Frontend UI Engineer Mastery

Complete 500-Page Course PDF

A complete A-to-Z curriculum for students who want to become professional frontend UI engineers. It covers UI design, HTML, CSS, JavaScript, TypeScript, React, Next.js, accessibility, testing, performance, deployment, portfolio projects and interview preparation.

English + Hinglish | Code + Output Preview | Exam + Practice | 20 Projects | 1-Year Roadmap

COURSE OUTCOMES

- Build pixel-perfect responsive screens from Figma or written requirements.
- Write semantic HTML, modern CSS and maintainable JavaScript/TypeScript.
- Create accessible reusable components with consistent states and variants.
- Integrate APIs with loading, error, empty and success states.
- Test, optimize, document and deploy production-ready frontend UI.

WHO THIS PDF IS FOR

- Absolute beginners who want a structured UI engineering path.
- Students who already know basics but need professional alignment and projects.
- Teachers creating classroom content with code-run examples and exams.
- Portfolio builders who want job-ready component and product screens.
- AIWorldAcademy learners preparing for certificate and interviews.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

How to Use This Course

This PDF is designed for a 1-year student journey. Each page is a classroom-ready lesson card.

DAILY STUDY ROUTINE

- Watch/read one topic and immediately run the code example.
- Maintain one notebook for English definitions and Hinglish explanations.
- Spend 70% time building, 20% debugging, 10% reading theory.
- Take screenshot after every successful output.
- Push code daily; small commits are better than one huge commit.

COURSE RULE

- Do not skip HTML/CSS basics; UI engineering depends on structure and layout.
- Do not use copied templates without understanding spacing and states.
- Every assignment needs source code, output screenshot and explanation.
- Every week must end with one mini project.
- Final month is for capstone, portfolio and interview preparation.

STUDENT ACTION PLAN

- Read the English explanation first, then read the Hinglish explanation in brackets to confirm the idea.
- Copy the code into the exact file name shown in the code box and run the command written below it.
- Compare your browser result with the output preview screen. Fix spacing, colors, headings and accessibility states.
- Complete the practice task, then answer the exam questions without looking at the notes.
- Push every lesson task to GitHub with a clear commit message and maintain one portfolio repository.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

1-Year Frontend UI Engineer Roadmap

Follow this roadmap sequentially. Do not jump to frameworks before HTML/CSS/JS basics are strong.

MONTH 1-3

- HTML, CSS, responsive layouts and UI basics.
- Git/GitHub, VS Code, browser DevTools and documentation habit.
- Mini projects: portfolio, landing page, pricing cards, forms.
- Focus on clean alignment, readable typography and accessibility foundations.

MONTH 4-12

- JavaScript, TypeScript, React, Next.js and API integration.
- Design systems, Storybook, testing, performance and deployment.
- Advanced projects: LMS dashboard, auth flow, component library, capstone.
- Interview prep, GitHub polishing, resume and portfolio presentation.

STUDENT ACTION PLAN

- Read the English explanation first, then read the Hinglish explanation in brackets to confirm the idea.
- Copy the code into the exact file name shown in the code box and run the command written below it.
- Compare your browser result with the output preview screen. Fix spacing, colors, headings and accessibility states.
- Complete the practice task, then answer the exam questions without looking at the notes.
- Push every lesson task to GitHub with a clear commit message and maintain one portfolio repository.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Professional UI Quality Standard

A frontend UI engineer is judged by both code quality and user experience quality.

VISUAL STANDARD

- Consistent spacing scale, type scale, radius and color tokens.
- No overlap, no accidental blank half-page, no unreadable low contrast.
- Clear hierarchy: title, description, action and feedback.
- Responsive behavior tested at common widths.
- Professional empty, loading, success and error states.

ENGINEERING STANDARD

- Reusable components with clear names and props.
- Separation of structure, style, behavior and data.
- Type safety where possible.
- Accessible keyboard and screen reader behavior.
- Tests, documentation and deployment readiness.

STUDENT ACTION PLAN

- Read the English explanation first, then read the Hinglish explanation in brackets to confirm the idea.
- Copy the code into the exact file name shown in the code box and run the command written below it.
- Compare your browser result with the output preview screen. Fix spacing, colors, headings and accessibility states.
- Complete the practice task, then answer the exam questions without looking at the notes.
- Push every lesson task to GitHub with a clear commit message and maintain one portfolio repository.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Course Assessment and Certificate Criteria

This structure can be used inside AIWorldAcademy LMS for progress tracking and certificate eligibility.

ASSESSMENT

- Module quiz after each major topic.
- Code-run assignment with screenshot proof.
- Weekly project review and debugging viva.
- Final capstone with deployed link.
- Interview-style explanation in English and Hinglish.

CERTIFICATE ELIGIBILITY

- 80% lesson completion.
- 70% assignment completion.
- All required project submissions.
- Final capstone passing score.
- GitHub and deployed portfolio link.

STUDENT ACTION PLAN

- Read the English explanation first, then read the Hinglish explanation in brackets to confirm the idea.
- Copy the code into the exact file name shown in the code box and run the command written below it.
- Compare your browser result with the output preview screen. Fix spacing, colors, headings and accessibility states.
- Complete the practice task, then answer the exam questions without looking at the notes.
- Push every lesson task to GitHub with a clear commit message and maintain one portfolio repository.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Table of Contents - Part 1

Module 01	Frontend UI Engineer Role and Mindset	Career Foundation
Module 02	How Browsers Render UI	Browser Core
Module 03	Professional Environment Setup	Tools
Module 04	Git, GitHub and Team Workflow	Workflow
Module 05	HTML Document Structure	HTML
Module 06	Semantic HTML and SEO Basics	HTML
Module 07	Links, Buttons, Images and Media	HTML UI
Module 08	Forms and Native Inputs	HTML Forms
Module 09	Tables, Metadata and Content Patterns	HTML Advanced
Module 10	CSS Selectors, Cascade and Specificity	CSS Core
Module 11	Box Model, Spacing and Sizing	CSS Core
Module 12	Typography, Color and Visual Hierarchy	UI Design
Module 13	Flexbox Layout Mastery	CSS Layout
Module 14	CSS Grid Layout Mastery	CSS Layout
Module 15	Responsive Design and Breakpoints	Responsive UI
Module 16	Modern CSS Variables and Cascade Layers	CSS Architecture
Module 17	Sass, SCSS and CSS Organization	CSS Tools

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Table of Contents - Part 2

Module 18	Tailwind CSS Utility-First UI	UI Framework
Module 19	UI Design Principles for Engineers	UI Engineering
Module 20	Design Systems and Figma Handoff	Design System
Module 21	Accessibility Foundations	A11y
Module 22	Advanced Accessibility Patterns	A11y Advanced
Module 23	JavaScript Fundamentals for UI	JavaScript
Module 24	DOM Manipulation and Events	JavaScript UI
Module 25	Form Validation with JavaScript	Forms
Module 26	Async JavaScript and Fetch API	Data UI
Module 27	Modules, Bundlers and Build Tools	Tooling
Module 28	TypeScript for UI Engineers	TypeScript
Module 29	React Component Fundamentals	React
Module 30	React State, Props and Data Flow	React
Module 31	React Hooks and Effects	React Advanced
Module 32	React Forms and Component Libraries	React UI
Module 33	React Accessibility and Performance	React Quality
Module 34	Next.js App Router UI Engineering	Next.js

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Table of Contents - Part 3

Module 35	API Integration and Auth UI	Integration
Module 36	State Management Patterns	State
Module 37	Routing, Navigation and Protected UI	Navigation
Module 38	Animation and Micro-Interactions	Motion UI
Module 39	SVG, Icons, Canvas and Visual Assets	Visuals
Module 40	Charts and Data Visualization UI	Data Viz
Module 41	Unit and Component Testing	Testing
Module 42	End-to-End Testing with Playwright	Testing E2E
Module 43	Storybook and UI Documentation	Docs
Module 44	Web Performance and Core Web Vitals	Performance
Module 45	SEO, Metadata and Social Sharing	SEO
Module 46	Frontend Security Essentials	Security
Module 47	PWA, Offline UI and Browser Storage	PWA
Module 48	Deployment, CI/CD and Production Checks	DevOps for UI
Module 49	Other Frontend Frameworks and Languages	Ecosystem
Module 50	Career, Interview and Capstone Delivery	Career

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Frontend Languages, Syntaxes and File Types

A frontend UI engineer does not use only one language. Real projects combine structure, styling, logic, types, configuration, documentation, data queries and framework templates.

HTML
Structure and semantic content (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

CSS
Visual style, layout and responsive UI (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

JavaScript
Browser logic and interaction (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

TypeScript
Safer JS with types (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

JSX/TSX
React component markup (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

Sass/SCSS
CSS preprocessor for large apps (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

JSON
Config and API data (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

YAML
CI/CD and workflow files (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

Markdown/MDX
Docs, content and component docs (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

GraphQL
Typed data queries (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

SVG
Vector graphics and icons (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

WebAssembly
High-performance browser modules from Rust/C/C++ (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

Vue SFC
Vue single-file components (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

Angular Templates
Angular UI markup (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

Svelte
Compiled UI components (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

Astro
Content-first frontend pages (Hinglish: real project me yeh part isi kaam ke liye use hota hai.)

Teacher note: HTML, CSS, JavaScript and TypeScript are the core. React/Next/Vue/Angular/Svelte add framework syntax. JSON/YAML/Markdown/GraphQL/SVG are supporting languages/file formats used daily in frontend UI engineering.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

UI Engineering Tools and Setup

Tools are not the skill; tools help you prove and improve the skill.

REQUIRED TOOLS

- VS Code with formatting and linting extensions.
- Node.js LTS, npm/pnpm, Git and modern browser.
- Chrome/Edge/Firefox DevTools for layout and performance.
- Figma or design viewer for handoff.
- GitHub account for code submission.

OPTIONAL PROFESSIONAL TOOLS

- Storybook for component documentation.
- Playwright for browser testing.
- Lighthouse for performance checks.
- Vercel/Netlify for preview deployments.
- Sentry or logging tool for production errors.

STUDENT ACTION PLAN

- Read the English explanation first, then read the Hinglish explanation in brackets to confirm the idea.
- Copy the code into the exact file name shown in the code box and run the command written below it.
- Compare your browser result with the output preview screen. Fix spacing, colors, headings and accessibility states.
- Complete the practice task, then answer the exam questions without looking at the notes.
- Push every lesson task to GitHub with a clear commit message and maintain one portfolio repository.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

One Running Project Used Throughout the Course

Instead of learning disconnected examples only, this PDF uses one running product idea: an AIWorldAcademy student course portal. Every topic improves one part of this project.

CODE RUN EXAMPLE project structure

```
aiworld-frontend-ui/
index.html
css/
base.css
layout.css
components.css
js/
main.js
api.js
validation.js
src/
components/
pages/
hooks/
styles/
tests/
package.json
README.md
```

```
$ mkdir aiworld-frontend-ui && cd aiworld-frontend-ui
```

Starter Output Preview

AIWorldAcademy Student Portal

Hero section, course card, navigation, progress and call-to-action will grow lesson by lesson.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

GitHub Submission Standard

Professional students learn to communicate work through GitHub, not only through screenshots.

EVERY SUBMISSION MUST INCLUDE

- Clean source code in correct folders.
- README with run command and project description.
- At least one output screenshot.
- Deployed link when applicable.
- Short learning reflection.

RECOMMENDED COMMIT STYLE

- feat: add responsive course card
- fix: correct mobile navbar overflow
- style: improve spacing tokens
- test: add login flow e2e test
- docs: add project screenshot and run steps

STUDENT ACTION PLAN

- Read the English explanation first, then read the Hinglish explanation in brackets to confirm the idea.
- Copy the code into the exact file name shown in the code box and run the command written below it.
- Compare your browser result with the output preview screen. Fix spacing, colors, headings and accessibility states.
- Complete the practice task, then answer the exam questions without looking at the notes.
- Push every lesson task to GitHub with a clear commit message and maintain one portfolio repository.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Starter Project Preview

The same project grows through the course so students see how individual skills combine into one product.

INITIAL SCREENS

- Landing page with hero and course CTA.
- Course catalog with search/filter.
- Student dashboard with progress.
- Lesson player UI.
- Profile and certificate eligibility UI.

QUALITY GOALS

- Consistent burgundy/cream brand theme.
- Reusable card and button components.
- Accessible labels and focus states.
- Responsive layout at all breakpoints.
- Clean deployment and README.

STUDENT ACTION PLAN

- Read the English explanation first, then read the Hinglish explanation in brackets to confirm the idea.
- Copy the code into the exact file name shown in the code box and run the command written below it.
- Compare your browser result with the output preview screen. Fix spacing, colors, headings and accessibility states.
- Complete the practice task, then answer the exam questions without looking at the notes.
- Push every lesson task to GitHub with a clear commit message and maintain one portfolio repository.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 01: Frontend UI Engineer Role and Mindset

Career Foundation | HTML CSS JS TS UX | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn understanding the role, delivery mindset, quality bar and collaboration with designers/product teams. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap understanding the role, delivery mindset, quality bar and collaboration with designers/product teams ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Frontend UI Engineer Role and Mindset: Detailed Notes 1

English: These notes explain the practical engineering decisions behind HTML CSS JS TS UX. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Frontend UI Engineer Role and Mindset means applying HTML CSS JS TS UX to build reliable user interfaces.
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where frontend ui engineer role and mindset matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-01: add career foundation improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Frontend UI Engineer Role and Mindset: Detailed Notes 2

English: These notes explain the practical engineering decisions behind HTML CSS JS TS UX. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where frontend ui engineer role and mindset matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-01: add career foundation improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Frontend UI Engineer Role and Mindset

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE style.css

```
:root {
--brand: #7a1227;
--cream: #fff8ea;
--text: #151515;
}
.card {
background: var(--cream);
color: var(--text);
border: 1px solid #e6cfc0;
border-radius: 24px;
padding: clamp(1rem, 2vw, 2rem);
}
```

```
$ Open page and inspect styles
```

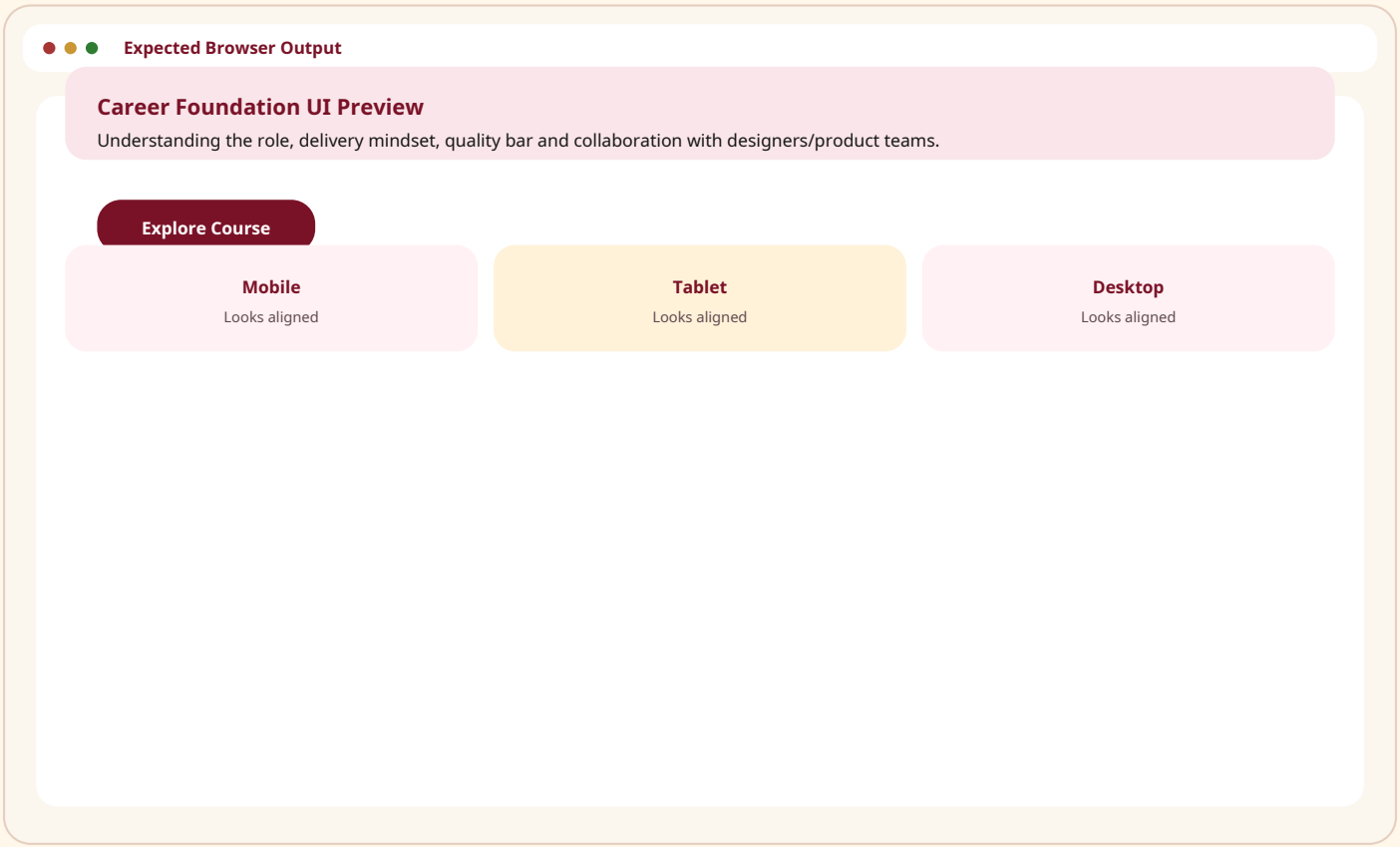
RUN AND UNDERSTAND

- Expected output: Card uses reusable tokens and scales spacing across devices.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Frontend UI Engineer Role and Mindset

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Frontend UI Engineer Role and Mindset

HANDS-ON TASK

- Build one screen section that demonstrates understanding the role, delivery mindset, quality bar and collaboration ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-01-frontend-ui-engineer-role-and-mind
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Frontend UI Engineer Role and Mindset

WRITTEN QUESTIONS

- Q1. What problem does Frontend UI Engineer Role and Mindset solve in real UI engineering?
- Q2. Which file/language is most commonly involved with HTML CSS JS TS UX?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where frontend ui engineer role and mindset is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Frontend UI Engineer Role and Mindset

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Frontend UI Engineer Role and Mindset

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using HTML CSS JS TS UX.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-01.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 02: How Browsers Render UI

Browser Core | DOM CSSOM Layout Paint Composi | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn rendering pipeline, browser devtools, reflow, repaint and cross-browser awareness. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap rendering pipeline, browser devtools, reflow, repaint and cross-browser awareness ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

How Browsers Render UI: Detailed Notes 1

English: These notes explain the practical engineering decisions behind DOM CSSOM Layout Paint Composite. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: How Browsers Render UI means applying DOM CSSOM Layout Paint Composite to build reliable user ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where how browsers render ui matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-02: add browser core improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

How Browsers Render UI: Detailed Notes 2

English: These notes explain the practical engineering decisions behind DOM CSSOM Layout Paint Composite. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where how browsers render ui matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-02: add browser core improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: How Browsers Render UI

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/main.js

```
const tabs = document.querySelectorAll('[data-tab]');
const panels = document.querySelectorAll('[data-panel]');

tabs.forEach((tab) => {
tab.addEventListener('click', () => {
panels.forEach((panel) => panel.hidden = panel.id !== tab.dataset.tab);
});
});
```

```
$ Open tabs.html and click tabs
```

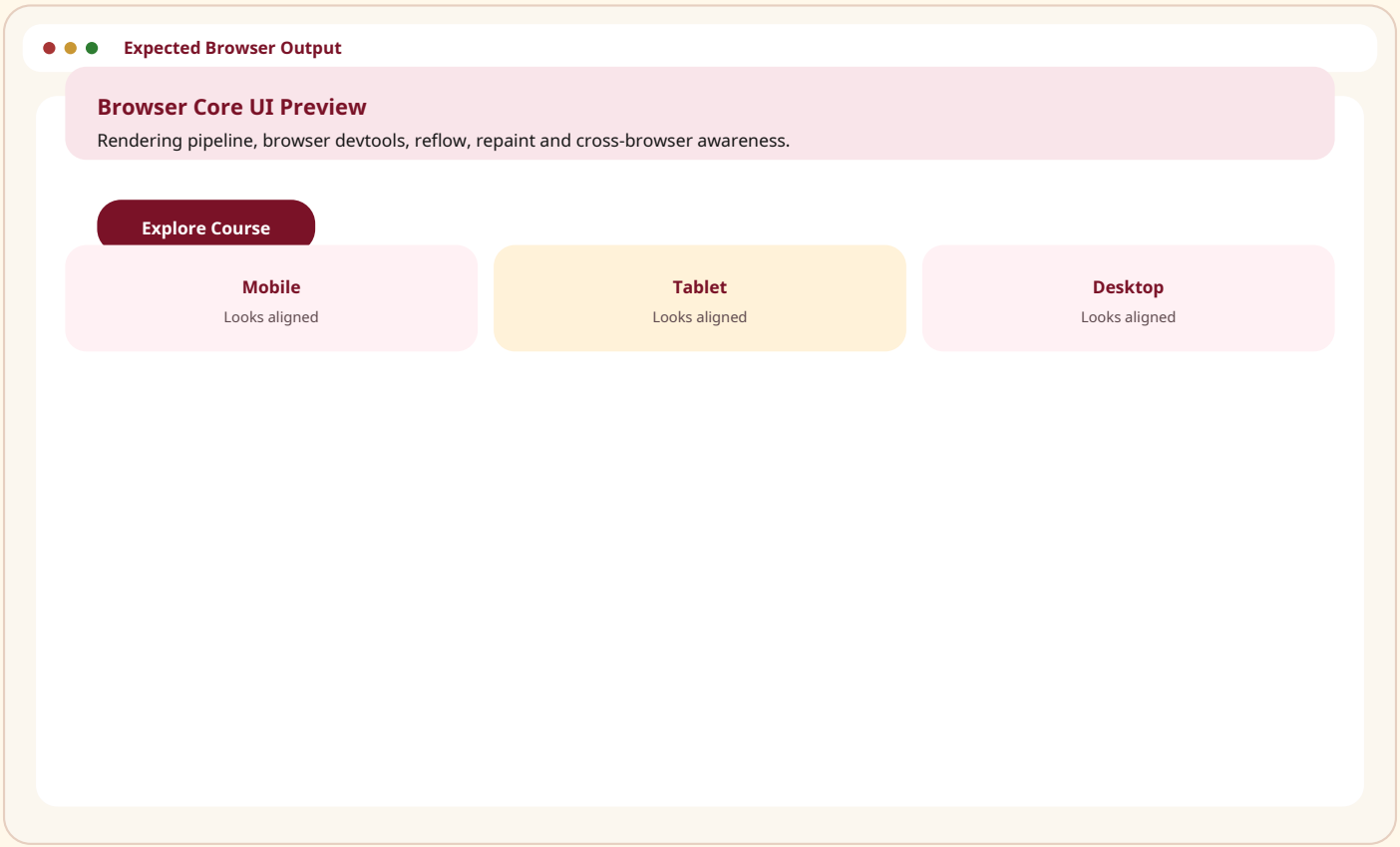
RUN AND UNDERSTAND

- Expected output: Only the selected tab panel remains visible.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: How Browsers Render UI

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: How Browsers Render UI

HANDS-ON TASK

- Build one screen section that demonstrates rendering pipeline, browser devtools, reflow, repaint and ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-02-how-browsers-render-ui
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: How Browsers Render UI

WRITTEN QUESTIONS

- Q1. What problem does How Browsers Render UI solve in real UI engineering?
- Q2. Which file/language is most commonly involved with DOM CSSOM Layout Paint Composite?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where how browsers render ui is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: How Browsers Render UI

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: How Browsers Render UI

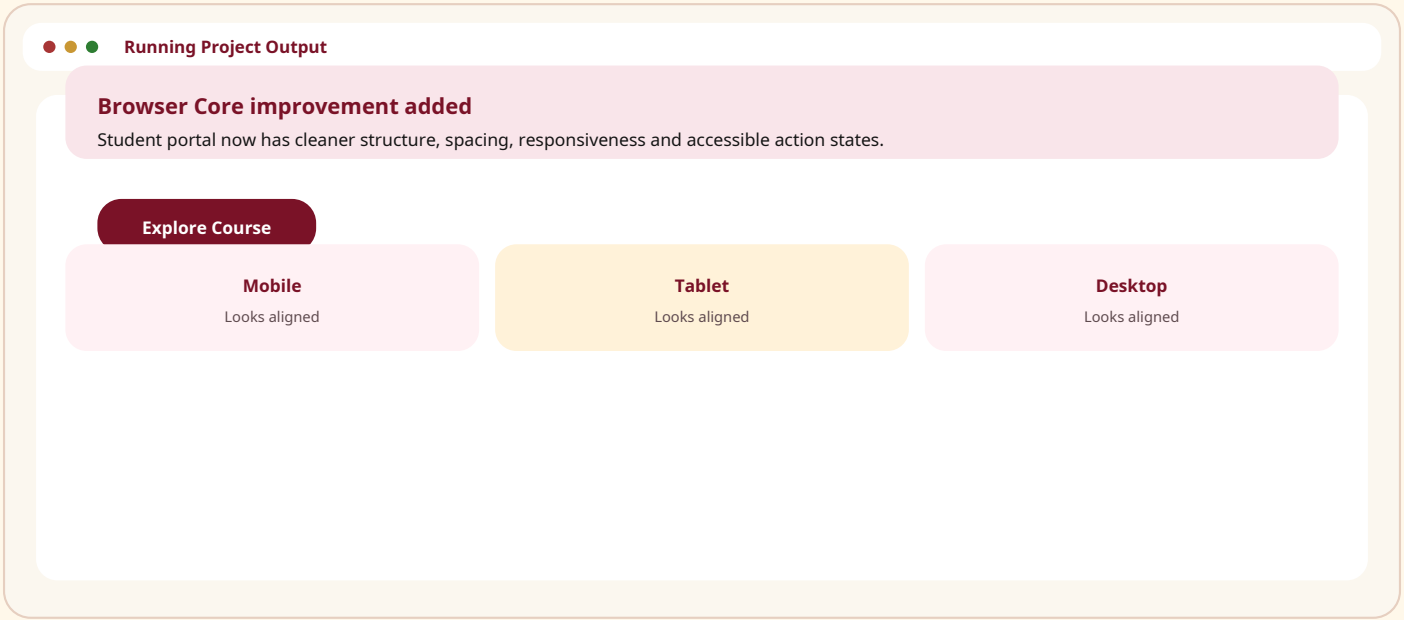
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using DOM CSSOM Layout Paint Composite.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-02.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 03: Professional Environment Setup

Tools | VS Code Node npm pnpm Vite Chr | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn developer machine setup, extensions, linting, formatters and project workspace. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap developer machine setup, extensions, linting, formatters and project workspace ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Professional Environment Setup: Detailed Notes 1

English: These notes explain the practical engineering decisions behind VS Code Node npm pnpm Vite Chrome DevTools. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Professional Environment Setup means applying VS Code Node npm pnpm Vite Chrome DevTools to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where professional environment setup matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-03: add tools improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Professional Environment Setup: Detailed Notes 2

English: These notes explain the practical engineering decisions behind VS Code Node npm pnpm Vite Chrome DevTools. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where professional environment setup matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-03: add tools improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Professional Environment Setup

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/api.ts

```
export async function getCourses() {
const res = await fetch('/api/courses');
if (!res.ok) throw new Error('Courses failed to load');
return res.json();
}

export async function renderCourses() {
const courses = await getCourses();
console.log('Loaded courses:', courses.length);
}
```

```
$ npm run dev
```

RUN AND UNDERSTAND

- Expected output: Course count logs after successful API call.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Professional Environment Setup

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.

Expected Browser Output

```
$ npm run dev
> frontend-ui@1.0.0 dev
Local: http://localhost:5173/
Status: compiled successfully
Module: Tools
No console errors found
Screenshot saved: output/module.png
```

VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Professional Environment Setup

HANDS-ON TASK

- Build one screen section that demonstrates developer machine setup, extensions, linting, formatters and project ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-03-professional-environment-setup
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Professional Environment Setup

WRITTEN QUESTIONS

- Q1. What problem does Professional Environment Setup solve in real UI engineering?
- Q2. Which file/language is most commonly involved with VS Code Node npm pnpm Vite Chrome DevTools?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where professional environment setup is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Professional Environment Setup

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Professional Environment Setup

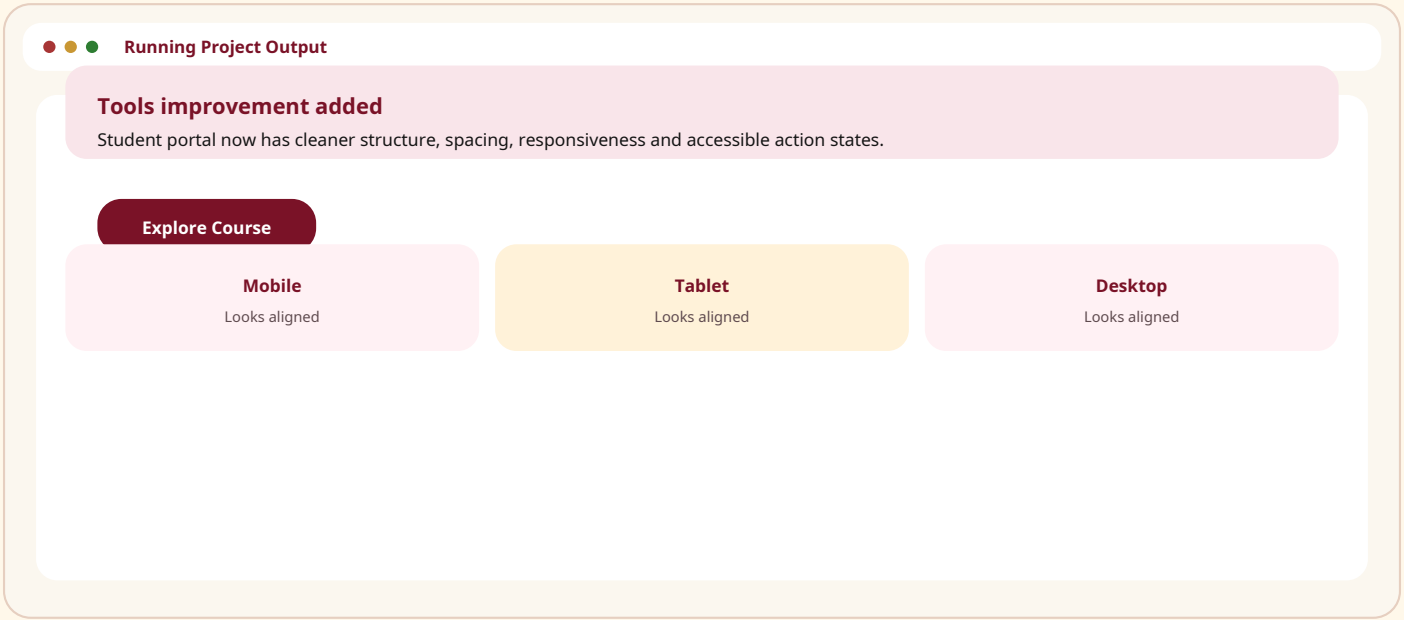
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using VS Code Node npm pnpm Vite Chrome DevTools.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-03.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 04: Git, GitHub and Team Workflow

Workflow | Git GitHub Branch PR README | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn version control, commit discipline, pull requests, issues and review-ready submissions. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap version control, commit discipline, pull requests, issues and review-ready submissions ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Git, GitHub and Team Workflow: Detailed Notes 1

English: These notes explain the practical engineering decisions behind Git GitHub Branch PR README. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Git, GitHub and Team Workflow means applying Git GitHub Branch PR README to build reliable user ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where git, github and team workflow matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-04: add workflow improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Git, GitHub and Team Workflow: Detailed Notes 2

English: These notes explain the practical engineering decisions behind Git GitHub Branch PR README. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where git, github and team workflow matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-04: add workflow improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Git, GitHub and Team Workflow

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE index.html

```
<section class="feature-card">
<p class="eyebrow">UI Engineer Skill</p>
<h2>Build clean interfaces</h2>
<p>Use structure, spacing, contrast and interaction states.</p>
<a href="/learn" class="btn">Learn More</a>
</section>
```

```
$ Open index.html
```

RUN AND UNDERSTAND

- Expected output: Feature card appears with heading, paragraph and link button.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Git, GitHub and Team Workflow

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.

Expected Browser Output

```
$ npm run dev
> frontend-ui@1.0.0 dev
Local: http://localhost:5173/
Status: compiled successfully
Module: Workflow
No console errors found
Screenshot saved: output/module.png
```

VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Git, GitHub and Team Workflow

HANDS-ON TASK

- Build one screen section that demonstrates version control, commit discipline, pull requests, issues and review-ready ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-04-git-github-and-team-workflow
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Git, GitHub and Team Workflow

WRITTEN QUESTIONS

- Q1. What problem does Git, GitHub and Team Workflow solve in real UI engineering?
- Q2. Which file/language is most commonly involved with Git GitHub Branch PR README?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where git, github and team workflow is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Git, GitHub and Team Workflow

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Git, GitHub and Team Workflow

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using Git GitHub Branch PR README.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-04.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Workflow improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 05: HTML Document Structure

HTML | HTML5 semantic elements | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn doctype, html head body, headings, lists, landmarks and clean page structure. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap doctype, html head body, headings, lists, landmarks and clean page structure ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

HTML Document Structure: Detailed Notes 1

English: These notes explain the practical engineering decisions behind HTML5 semantic elements. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: HTML Document Structure means applying HTML5 semantic elements to build reliable user interfaces.
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where html document structure matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-05: add html improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

HTML Document Structure: Detailed Notes 2

English: These notes explain the practical engineering decisions behind HTML5 semantic elements. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where html document structure matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-05: add html improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: HTML Document Structure

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE style.css

```
:root {
--brand: #7a1227;
--cream: #fff8ea;
--text: #151515;
}
.card {
background: var(--cream);
color: var(--text);
border: 1px solid #e6cfc0;
border-radius: 24px;
padding: clamp(1rem, 2vw, 2rem);
}
```

```
$ Open page and inspect styles
```

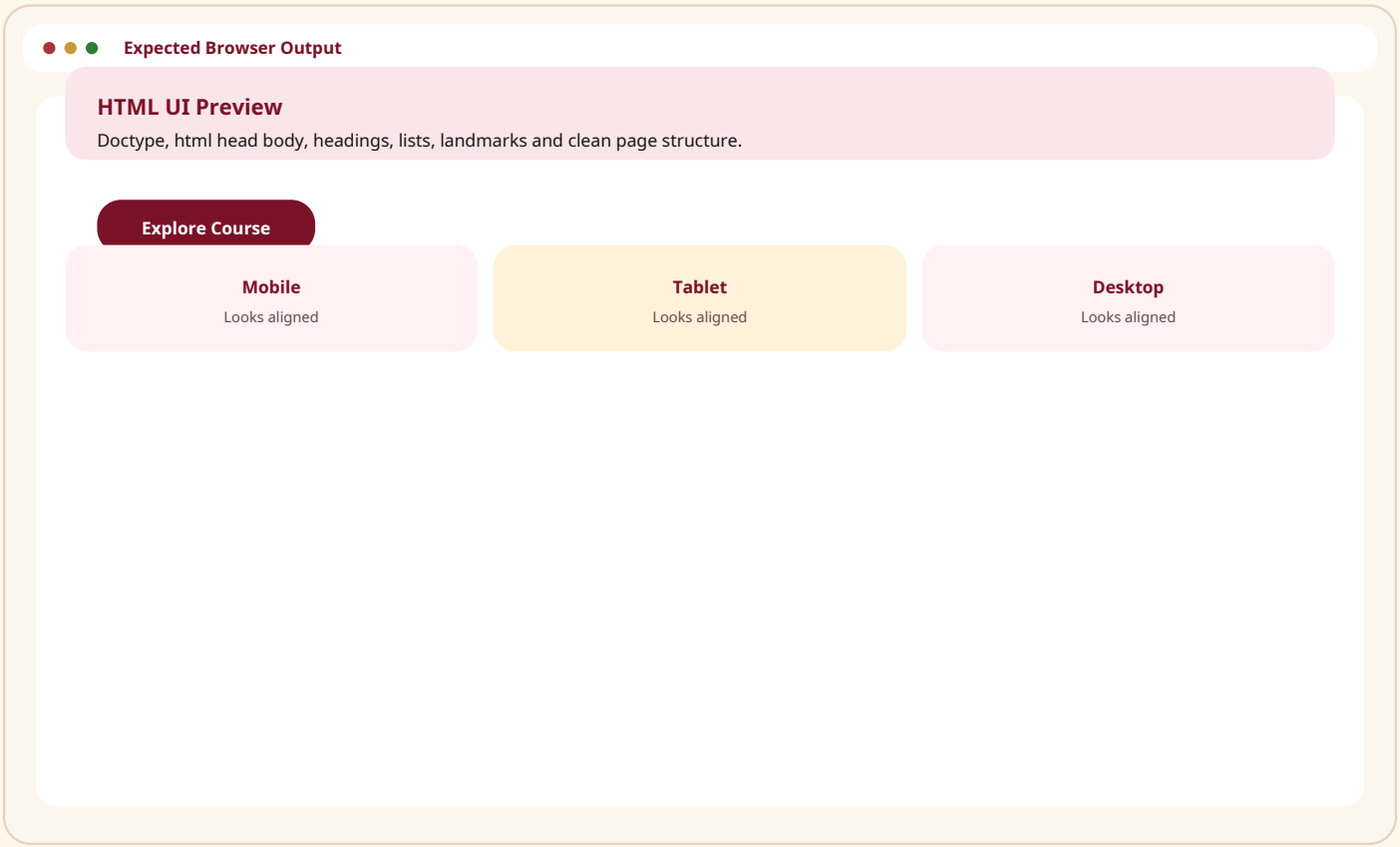
RUN AND UNDERSTAND

- Expected output: Card uses reusable tokens and scales spacing across devices.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: HTML Document Structure

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: HTML Document Structure

HANDS-ON TASK

- Build one screen section that demonstrates doctype, html head body, headings, lists, landmarks and clean page ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-05-html-document-structure
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: HTML Document Structure

WRITTEN QUESTIONS

- Q1. What problem does HTML Document Structure solve in real UI engineering?
- Q2. Which file/language is most commonly involved with HTML5 semantic elements?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where html document structure is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: HTML Document Structure

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: HTML Document Structure

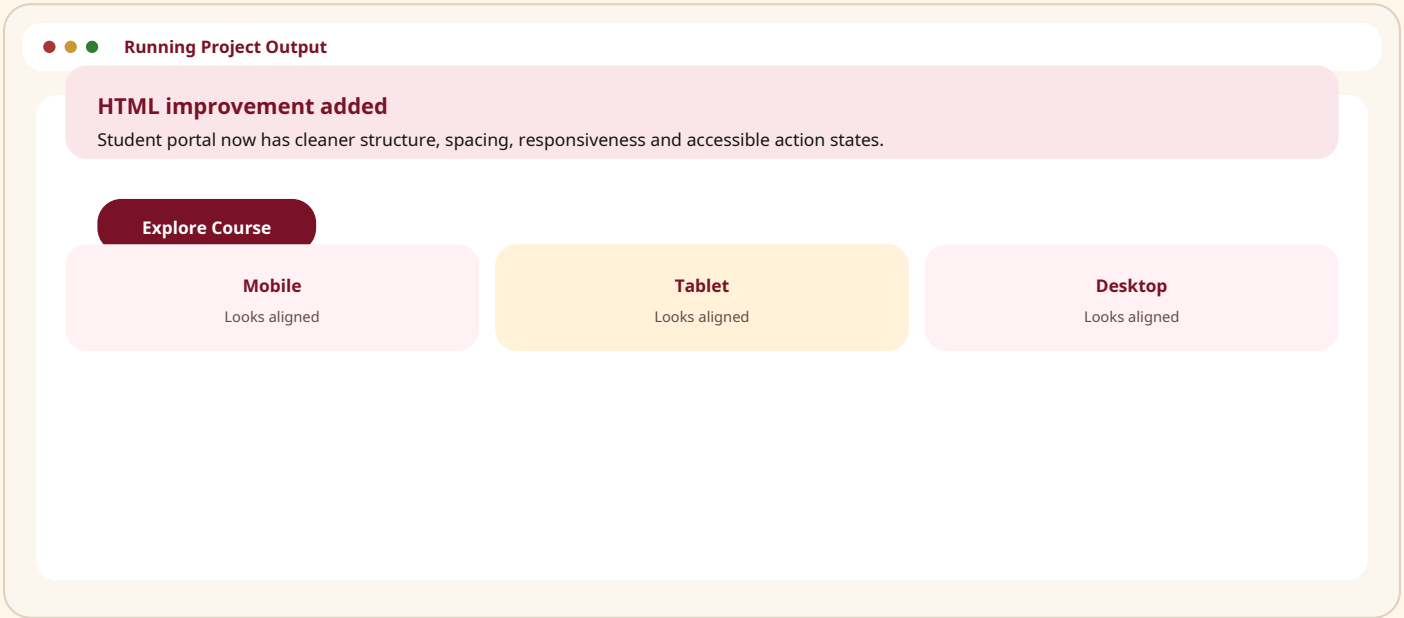
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using HTML5 semantic elements.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-05.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 06: Semantic HTML and SEO Basics

HTML | header nav main article sectio | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn meaningful markup, search-friendly headings, metadata and assistive tech support. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap meaningful markup, search-friendly headings, metadata and assistive tech support ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Semantic HTML and SEO Basics: Detailed Notes 1

English: These notes explain the practical engineering decisions behind header nav main article section footer meta. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Semantic HTML and SEO Basics means applying header nav main article section footer meta to build reliable ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where semantic html and seo basics matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-06: add html improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Semantic HTML and SEO Basics: Detailed Notes 2

English: These notes explain the practical engineering decisions behind header nav main article section footer meta. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where semantic html and seo basics matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-06: add html improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Semantic HTML and SEO Basics

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/main.js

```
const tabs = document.querySelectorAll('[data-tab]');
const panels = document.querySelectorAll('[data-panel]');

tabs.forEach((tab) => {
tab.addEventListener('click', () => {
panels.forEach((panel) => panel.hidden = panel.id !== tab.dataset.tab);
});
});
```

```
$ Open tabs.html and click tabs
```

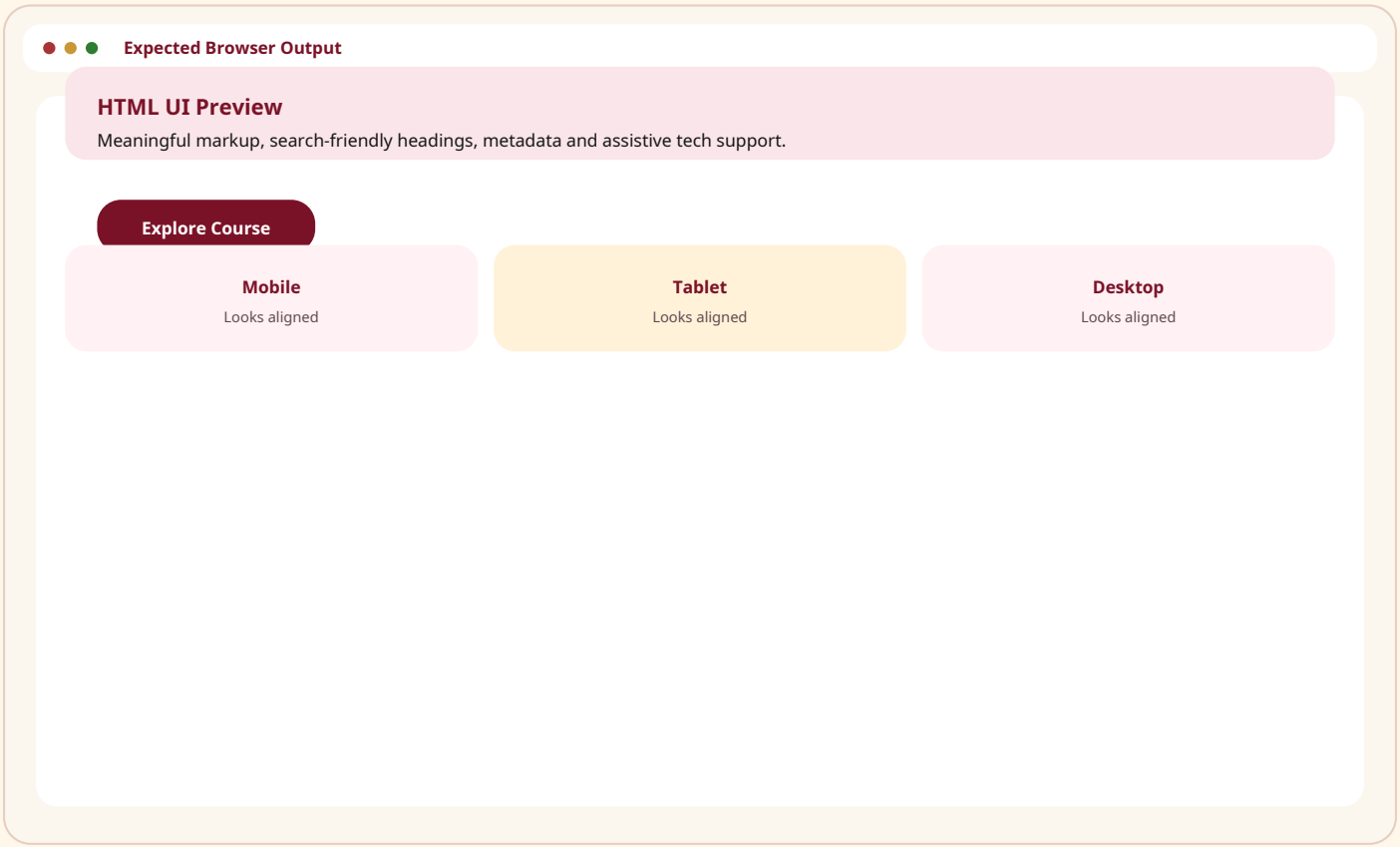
RUN AND UNDERSTAND

- Expected output: Only the selected tab panel remains visible.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Semantic HTML and SEO Basics

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Semantic HTML and SEO Basics

HANDS-ON TASK

- Build one screen section that demonstrates meaningful markup, search-friendly headings, metadata and assistive ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-06-semantic-html-and-seo-basics
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Semantic HTML and SEO Basics

WRITTEN QUESTIONS

- Q1. What problem does Semantic HTML and SEO Basics solve in real UI engineering?
- Q2. Which file/language is most commonly involved with header nav main article section footer meta?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where semantic html and seo basics is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Semantic HTML and SEO Basics

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Semantic HTML and SEO Basics

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using header nav main article section footer meta.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-06.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

HTML improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 07: Links, Buttons, Images and Media

HTML UI | a button img picture video SVG | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn correct href usage, accessible buttons, optimized images and media UI patterns. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap correct href usage, accessible buttons, optimized images and media UI patterns ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Links, Buttons, Images and Media: Detailed Notes 1

English: These notes explain the practical engineering decisions behind a button img picture video SVG. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Links, Buttons, Images and Media means applying a button img picture video SVG to build reliable ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where links, buttons, images and media matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-07: add html ui improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Links, Buttons, Images and Media: Detailed Notes 2

English: These notes explain the practical engineering decisions behind a button img picture video SVG. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where links, buttons, images and media matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-07: add html ui improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Links, Buttons, Images and Media

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE project/index.html

```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Course CTA</title>
<link rel="stylesheet" href="style.css" />
</head>
<body>
<main class="hero">
<h1>Frontend UI Engineer</h1>
<p>Learn by building real screens.</p>
<a class="btn" href="/courses/frontend-ui.html">Explore Course</a>
<button class="ghost" type="button">Watch Preview</button>
</main>
</body>
</html>
```

```
$ Open index.html with Live Server
```

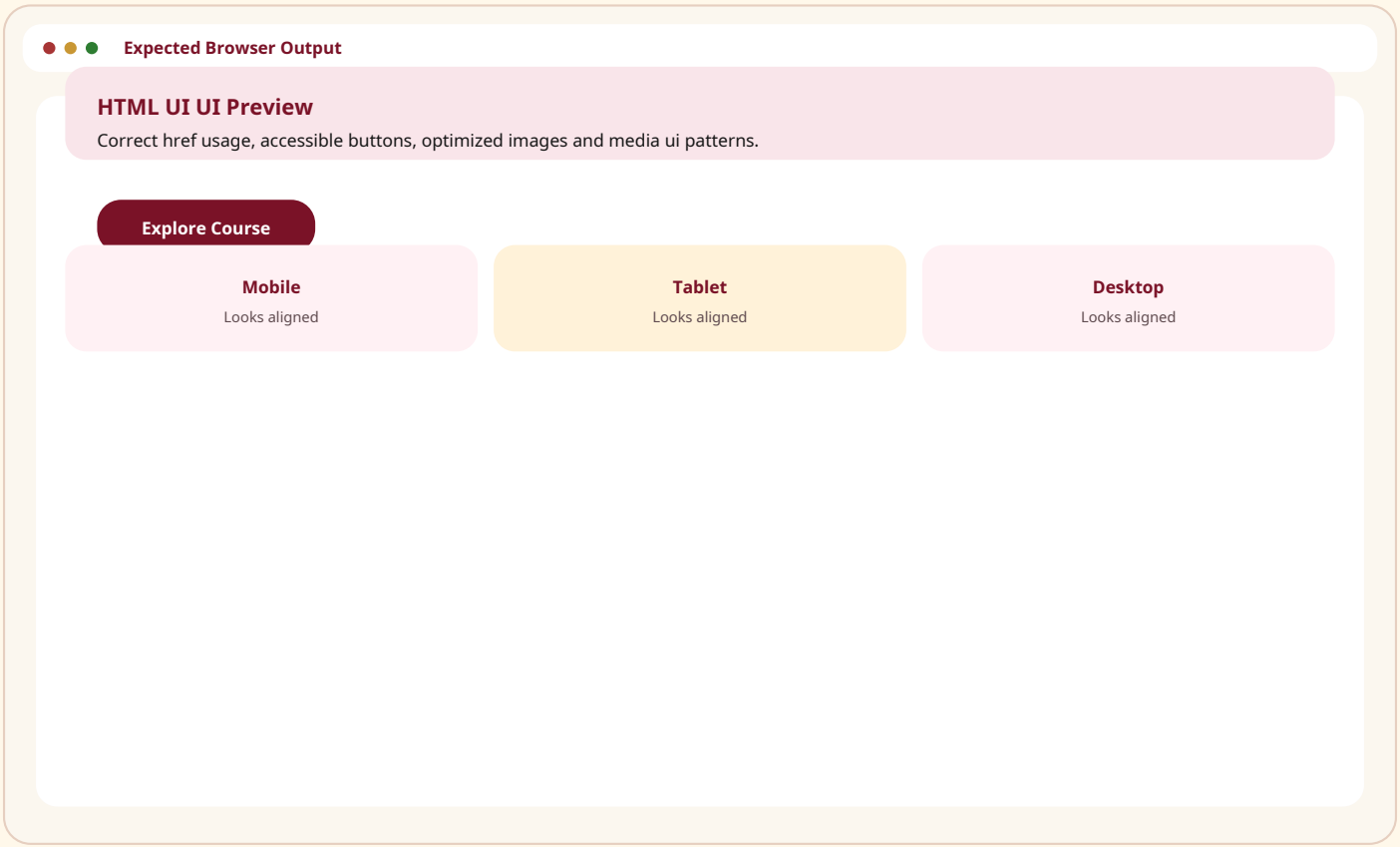
RUN AND UNDERSTAND

- Expected output: A styled link button opens the course page; the normal button performs an action.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Links, Buttons, Images and Media

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Links, Buttons, Images and Media

HANDS-ON TASK

- Build one screen section that demonstrates correct href usage, accessible buttons, optimized images and media UI ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-07-links-buttons-images-and-media
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Links, Buttons, Images and Media

WRITTEN QUESTIONS

- Q1. What problem does Links, Buttons, Images and Media solve in real UI engineering?
- Q2. Which file/language is most commonly involved with a button img picture video SVG?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where links, buttons, images and media is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Links, Buttons, Images and Media

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Links, Buttons, Images and Media

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using a button img picture video SVG.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-07.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

HTML UI improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 08: Forms and Native Inputs

HTML Forms | input label select textarea va | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn usable forms, labels, required fields, validation and error messages. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap usable forms, labels, required fields, validation and error messages ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Forms and Native Inputs: Detailed Notes 1

English: These notes explain the practical engineering decisions behind input label select textarea validation. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Forms and Native Inputs means applying input label select textarea validation to build reliable user ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where forms and native inputs matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-08: add html forms improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Forms and Native Inputs: Detailed Notes 2

English: These notes explain the practical engineering decisions behind input label select textarea validation. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where forms and native inputs matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-08: add html forms improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Forms and Native Inputs

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/forms/register.js

```
const form = document.querySelector('#registerForm');
const email = document.querySelector('#email');
const error = document.querySelector('#emailError');

form.addEventListener('submit', (event) => {
event.preventDefault();
if (!email.validity.valid) {
error.textContent = 'Please enter a valid email address.';
email.setAttribute('aria-invalid', 'true');
return;
}
error.textContent = '';
email.removeAttribute('aria-invalid');
form.submit();
});
```

```
$ Open form page and submit invalid email
```

RUN AND UNDERSTAND

- Expected output: Inline error appears and screen reader can identify invalid email field.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Forms and Native Inputs

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Forms and Native Inputs

HANDS-ON TASK

- Build one screen section that demonstrates usable forms, labels, required fields, validation and error messages.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-08-forms-and-native-inputs
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Forms and Native Inputs

WRITTEN QUESTIONS

- Q1. What problem does Forms and Native Inputs solve in real UI engineering?
- Q2. Which file/language is most commonly involved with input label select textarea validation?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where forms and native inputs is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Forms and Native Inputs

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Forms and Native Inputs

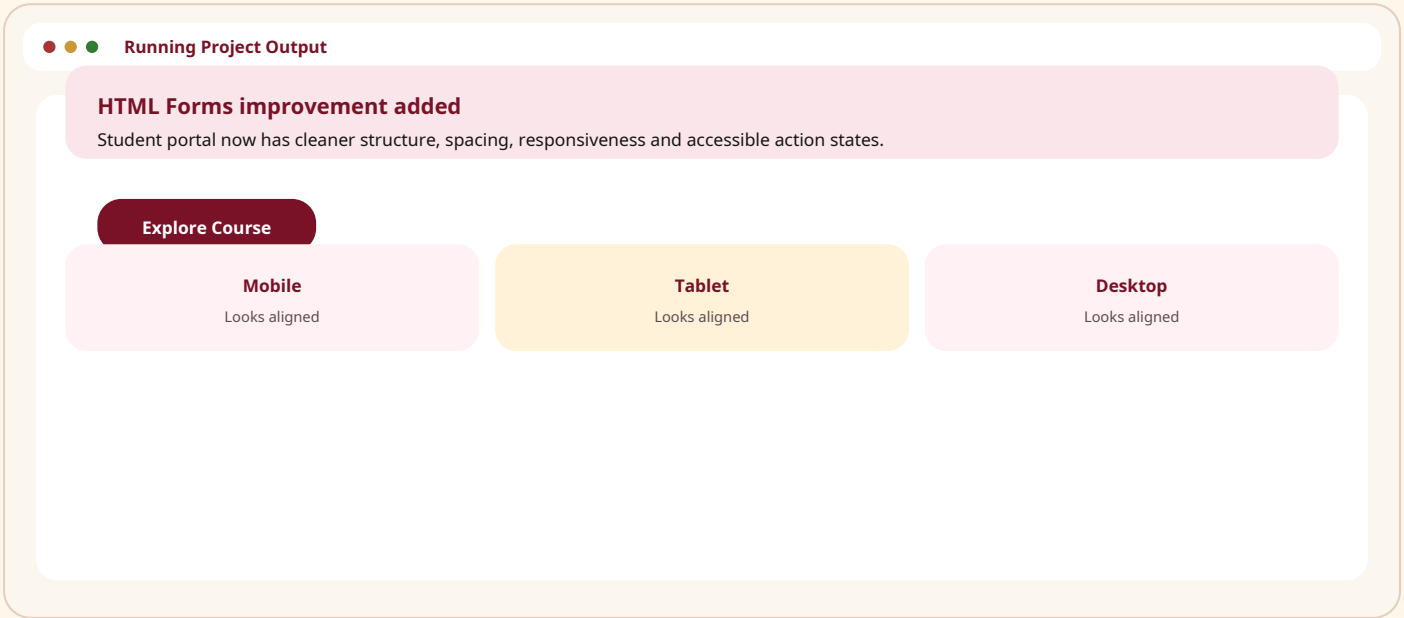
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using input label select textarea validation.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-08.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 09: Tables, Metadata and Content Patterns

HTML Advanced | table meta open graph schema | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn data tables, document metadata, content pages and SEO snippets. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap data tables, document metadata, content pages and SEO snippets ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Tables, Metadata and Content Patterns: Detailed Notes 1

English: These notes explain the practical engineering decisions behind table meta open graph schema. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Tables, Metadata and Content Patterns means applying table meta open graph schema to build reliable ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where tables, metadata and content patterns matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-09: add html advanced improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Tables, Metadata and Content Patterns: Detailed Notes 2

English: These notes explain the practical engineering decisions behind table meta open graph schema. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where tables, metadata and content patterns matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-09: add html advanced improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Tables, Metadata and Content Patterns

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE style.css

```
:root {
--brand: #7a1227;
--cream: #fff8ea;
--text: #151515;
}
.card {
background: var(--cream);
color: var(--text);
border: 1px solid #e6cfc0;
border-radius: 24px;
padding: clamp(1rem, 2vw, 2rem);
}
```

```
$ Open page and inspect styles
```

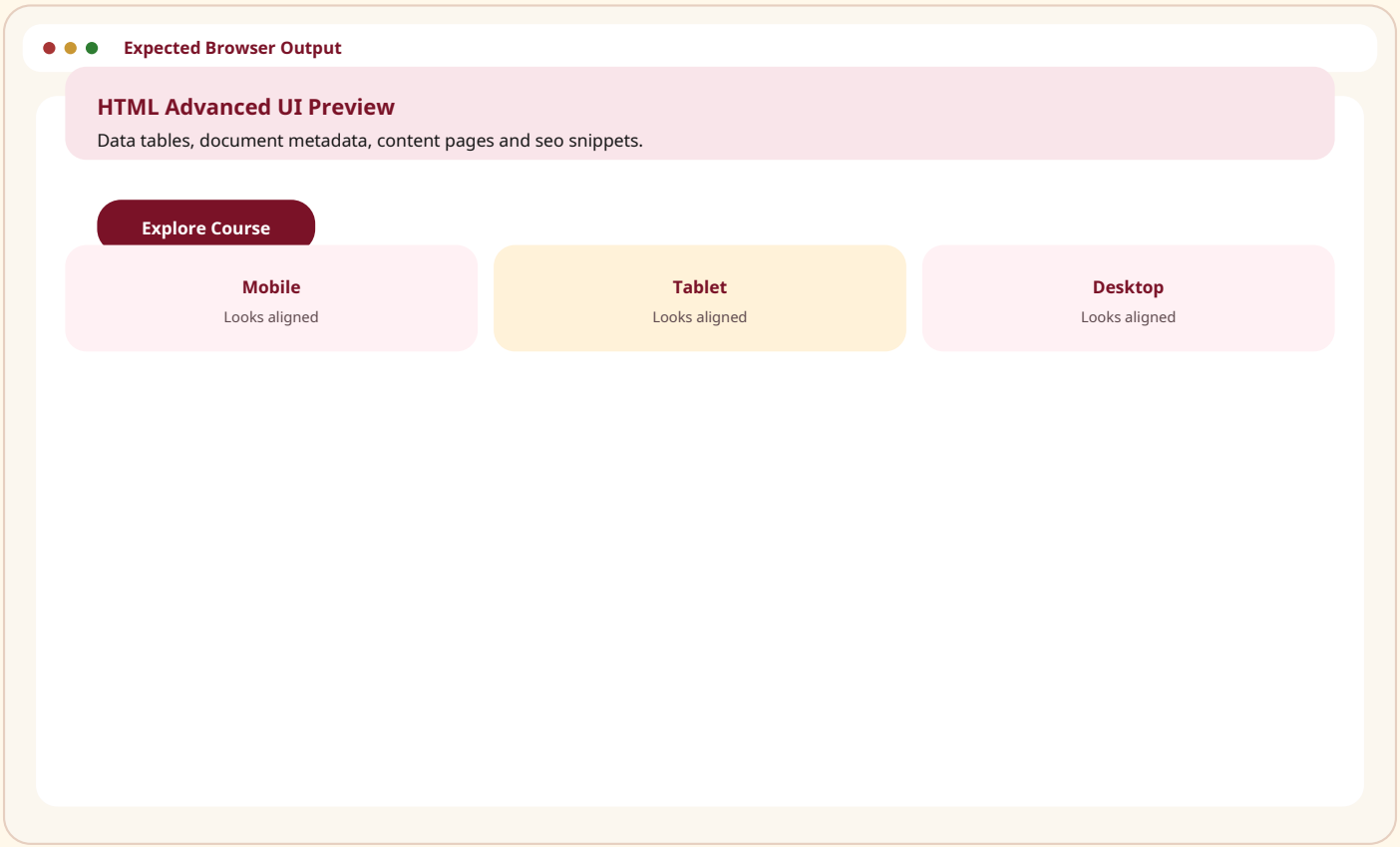
RUN AND UNDERSTAND

- Expected output: Card uses reusable tokens and scales spacing across devices.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Tables, Metadata and Content Patterns

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Tables, Metadata and Content Patterns

HANDS-ON TASK

- Build one screen section that demonstrates data tables, document metadata, content pages and SEO snippets.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-09-tables-metadata-and-content-patter
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Tables, Metadata and Content Patterns

WRITTEN QUESTIONS

- Q1. What problem does Tables, Metadata and Content Patterns solve in real UI engineering?
- Q2. Which file/language is most commonly involved with table meta open graph schema?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where tables, metadata and content patterns is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Tables, Metadata and Content Patterns

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Tables, Metadata and Content Patterns

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using table meta open graph schema.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-09.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

HTML Advanced improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 10: CSS Selectors, Cascade and Specificity

CSS Core | CSS selectors cascade layers s | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn selector strategy, cascade control, inheritance and maintainable styles. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap selector strategy, cascade control, inheritance and maintainable styles ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

CSS Selectors, Cascade and Specificity: Detailed Notes 1

English: These notes explain the practical engineering decisions behind CSS selectors cascade layers specificity. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: CSS Selectors, Cascade and Specificity means applying CSS selectors cascade layers specificity to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where css selectors, cascade and specificity matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-10: add css core improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

CSS Selectors, Cascade and Specificity: Detailed Notes 2

English: These notes explain the practical engineering decisions behind CSS selectors cascade layers specificity. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where css selectors, cascade and specificity matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-10: add css core improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: CSS Selectors, Cascade and Specificity

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/main.js

```
const tabs = document.querySelectorAll('[data-tab]');
const panels = document.querySelectorAll('[data-panel]');

tabs.forEach((tab) => {
tab.addEventListener('click', () => {
panels.forEach((panel) => panel.hidden = panel.id !== tab.dataset.tab);
});
});
```

```
$ Open tabs.html and click tabs
```

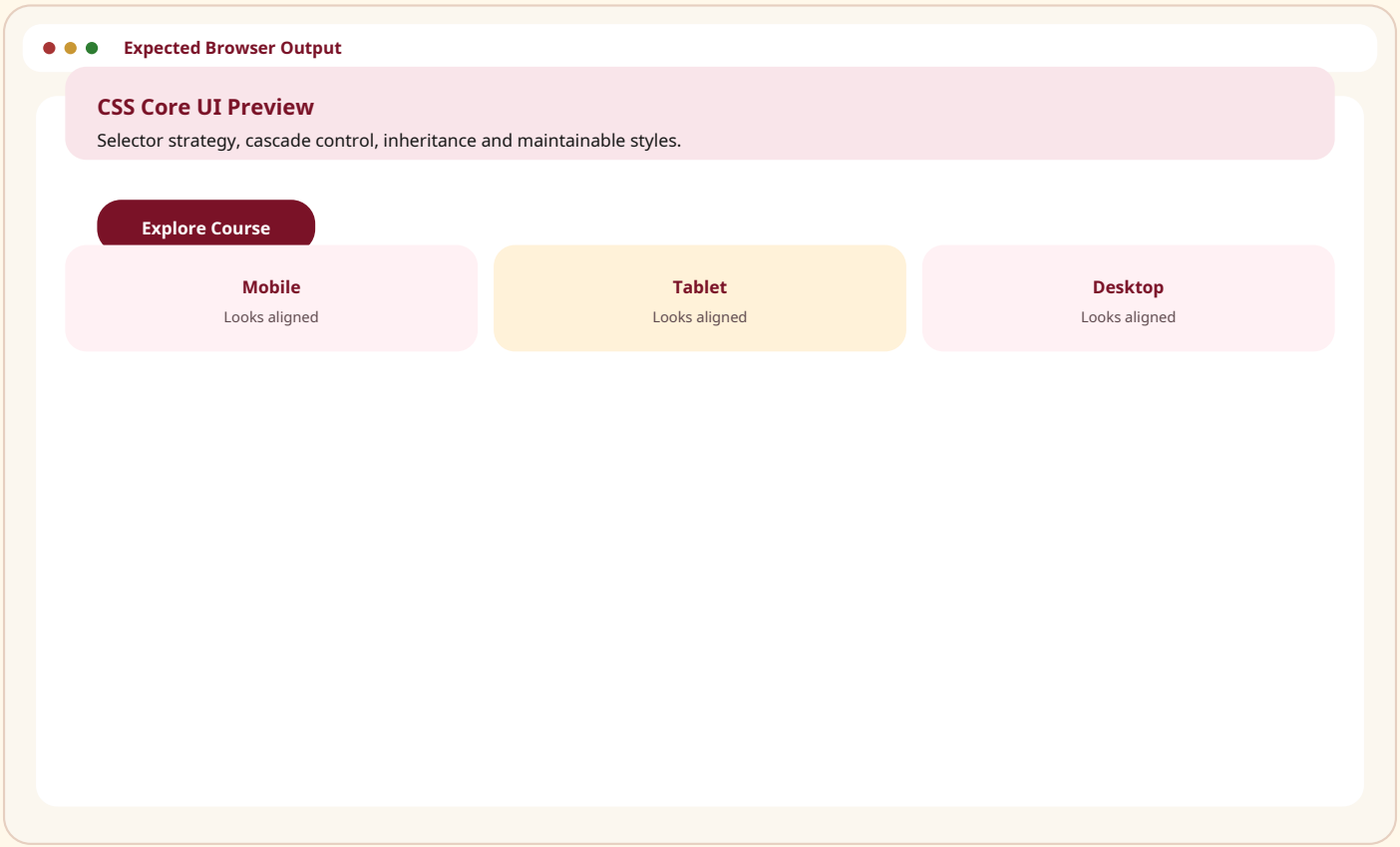
RUN AND UNDERSTAND

- Expected output: Only the selected tab panel remains visible.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: CSS Selectors, Cascade and Specificity

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: CSS Selectors, Cascade and Specificity

HANDS-ON TASK

- Build one screen section that demonstrates selector strategy, cascade control, inheritance and maintainable ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-10-css-selectors-cascade-and-specific
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: CSS Selectors, Cascade and Specificity

WRITTEN QUESTIONS

- Q1. What problem does CSS Selectors, Cascade and Specificity solve in real UI engineering?
- Q2. Which file/language is most commonly involved with CSS selectors cascade layers specificity?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where css selectors, cascade and specificity is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: CSS Selectors, Cascade and Specificity

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: CSS Selectors, Cascade and Specificity

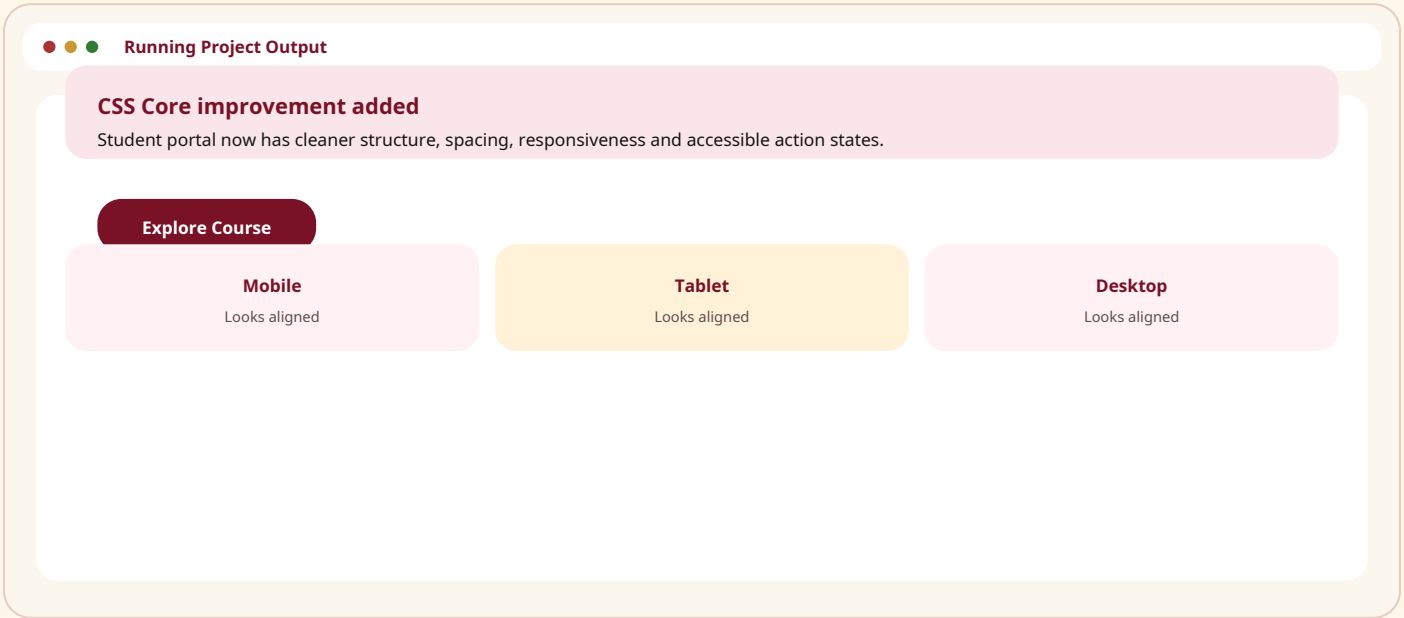
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using CSS selectors cascade layers specificity.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-10.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 11: Box Model, Spacing and Sizing

CSS Core | box-sizing margin padding bord | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn spacing system, rem, px, %, viewport units and predictable layouts. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap spacing system, rem, px, %, viewport units and predictable layouts ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Box Model, Spacing and Sizing: Detailed Notes 1

English: These notes explain the practical engineering decisions behind box-sizing margin padding border units. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Box Model, Spacing and Sizing means applying box-sizing margin padding border units to build reliable ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where box model, spacing and sizing matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-11: add css core improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Box Model, Spacing and Sizing: Detailed Notes 2

English: These notes explain the practical engineering decisions behind box-sizing margin padding border units. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where box model, spacing and sizing matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-11: add css core improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Box Model, Spacing and Sizing

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/api.ts

```
export async function getCourses() {
const res = await fetch('/api/courses');
if (!res.ok) throw new Error('Courses failed to load');
return res.json();
}

export async function renderCourses() {
const courses = await getCourses();
console.log('Loaded courses:', courses.length);
}
```

```
$ npm run dev
```

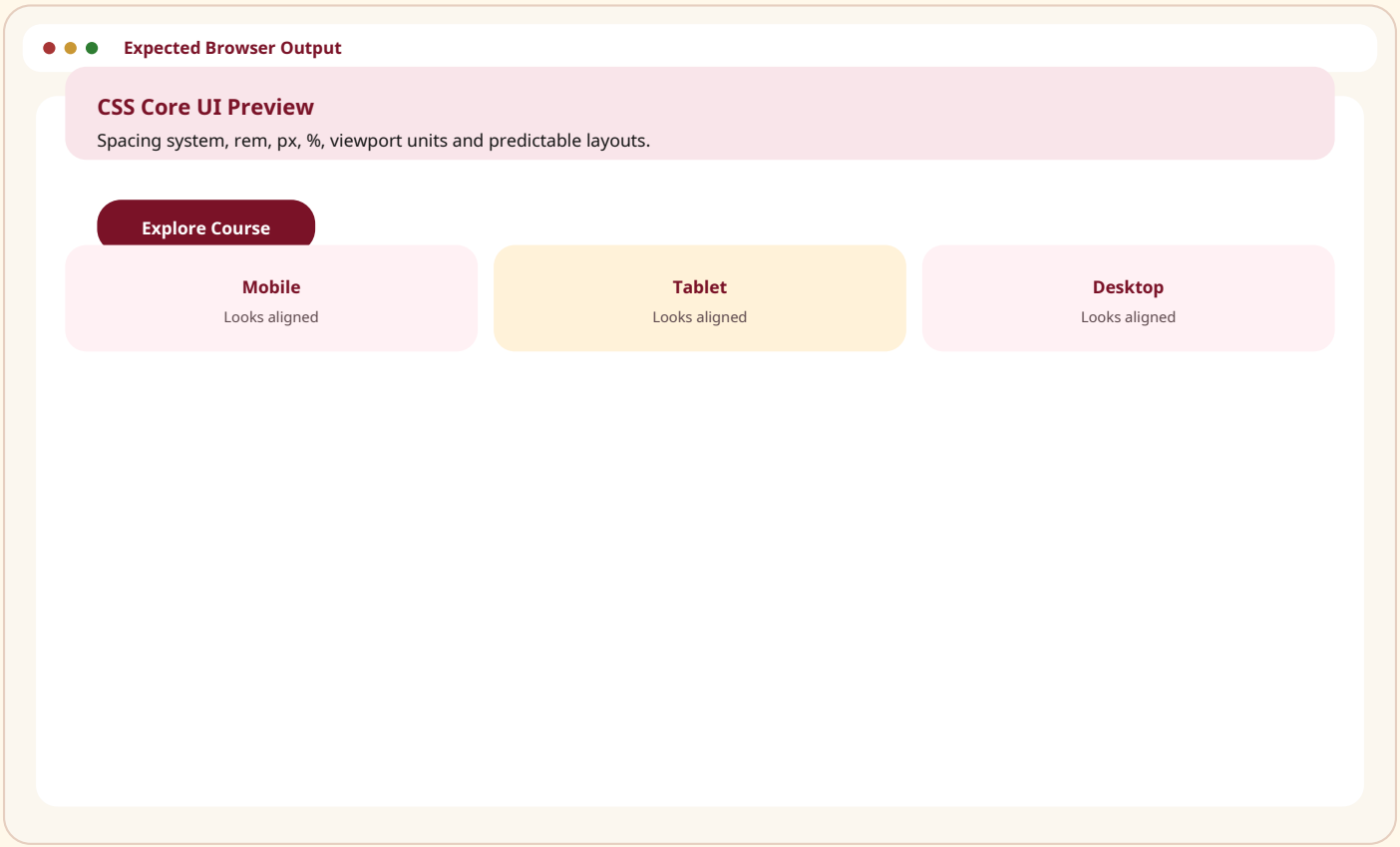
RUN AND UNDERSTAND

- Expected output: Course count logs after successful API call.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Box Model, Spacing and Sizing

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Box Model, Spacing and Sizing

HANDS-ON TASK

- Build one screen section that demonstrates spacing system, rem, px, %, viewport units and predictable layouts.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-11-box-model-spacing-and-sizing
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Box Model, Spacing and Sizing

WRITTEN QUESTIONS

- Q1. What problem does Box Model, Spacing and Sizing solve in real UI engineering?
- Q2. Which file/language is most commonly involved with box-sizing margin padding border units?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where box model, spacing and sizing is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Box Model, Spacing and Sizing

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Box Model, Spacing and Sizing

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using box-sizing margin padding border units.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-11.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

CSS Core improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 12: Typography, Color and Visual Hierarchy

UI Design | font scale line-height color t | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn readability, font pairing, color systems, hierarchy and UI polish. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap readability, font pairing, color systems, hierarchy and UI polish ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Typography, Color and Visual Hierarchy: Detailed Notes 1

English: These notes explain the practical engineering decisions behind font scale line-height color tokens contrast. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Typography, Color and Visual Hierarchy means applying font scale line-height color tokens contrast to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where typography, color and visual hierarchy matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-12: add ui design improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Typography, Color and Visual Hierarchy: Detailed Notes 2

English: These notes explain the practical engineering decisions behind font scale line-height color tokens contrast. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where typography, color and visual hierarchy matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-12: add ui design improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Typography, Color and Visual Hierarchy

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE index.html

```
<section class="feature-card">
<p class="eyebrow">UI Engineer Skill</p>
<h2>Build clean interfaces</h2>
<p>Use structure, spacing, contrast and interaction states.</p>
<a href="/learn" class="btn">Learn More</a>
</section>
```

```
$ Open index.html
```

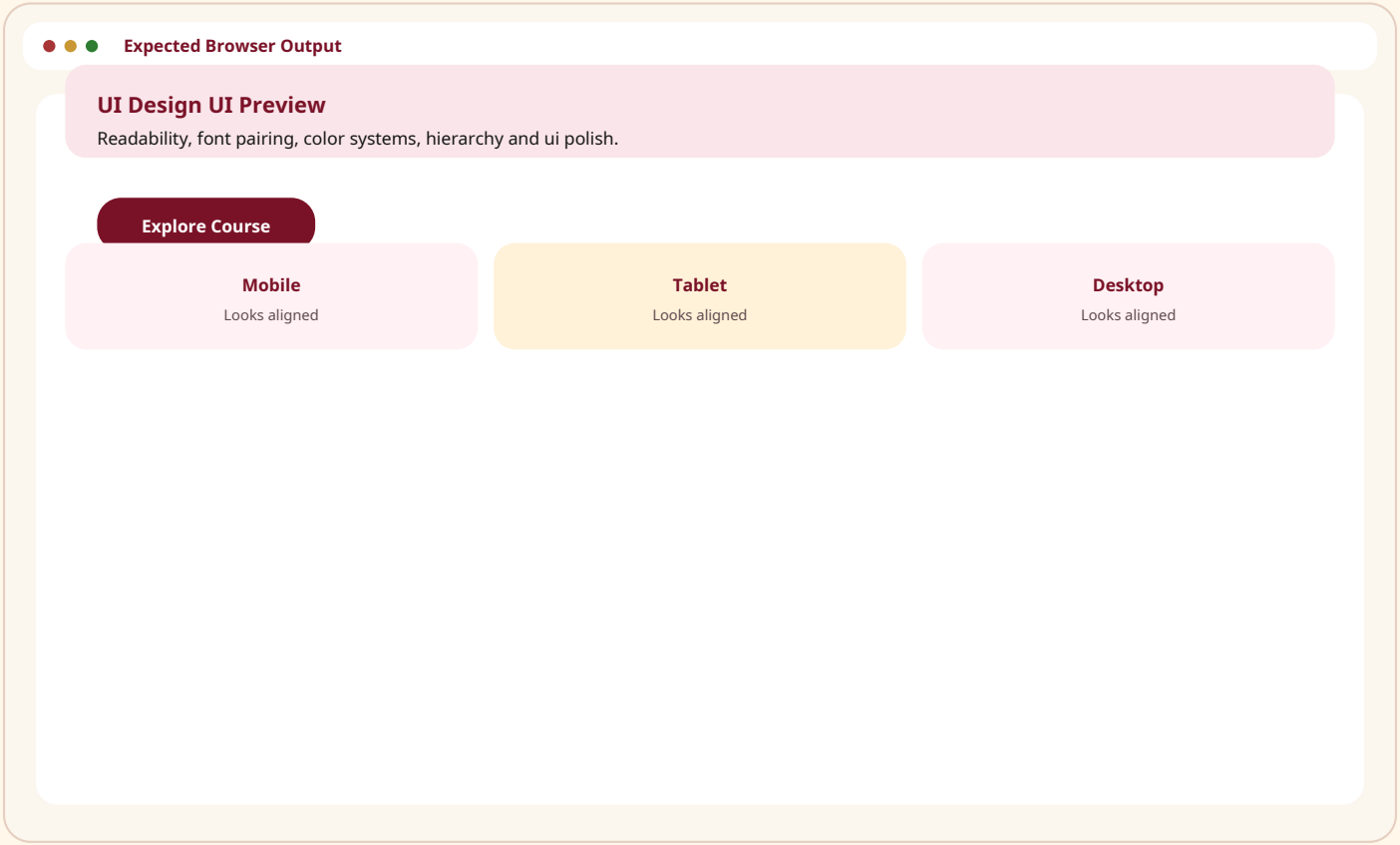
RUN AND UNDERSTAND

- Expected output: Feature card appears with heading, paragraph and link button.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Typography, Color and Visual Hierarchy

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Typography, Color and Visual Hierarchy

HANDS-ON TASK

- Build one screen section that demonstrates readability, font pairing, color systems, hierarchy and UI polish.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-12-typography-color-and-visual-hierar
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Typography, Color and Visual Hierarchy

WRITTEN QUESTIONS

- Q1. What problem does Typography, Color and Visual Hierarchy solve in real UI engineering?
- Q2. Which file/language is most commonly involved with font scale line-height color tokens contrast?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where typography, color and visual hierarchy is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Typography, Color and Visual Hierarchy

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Typography, Color and Visual Hierarchy

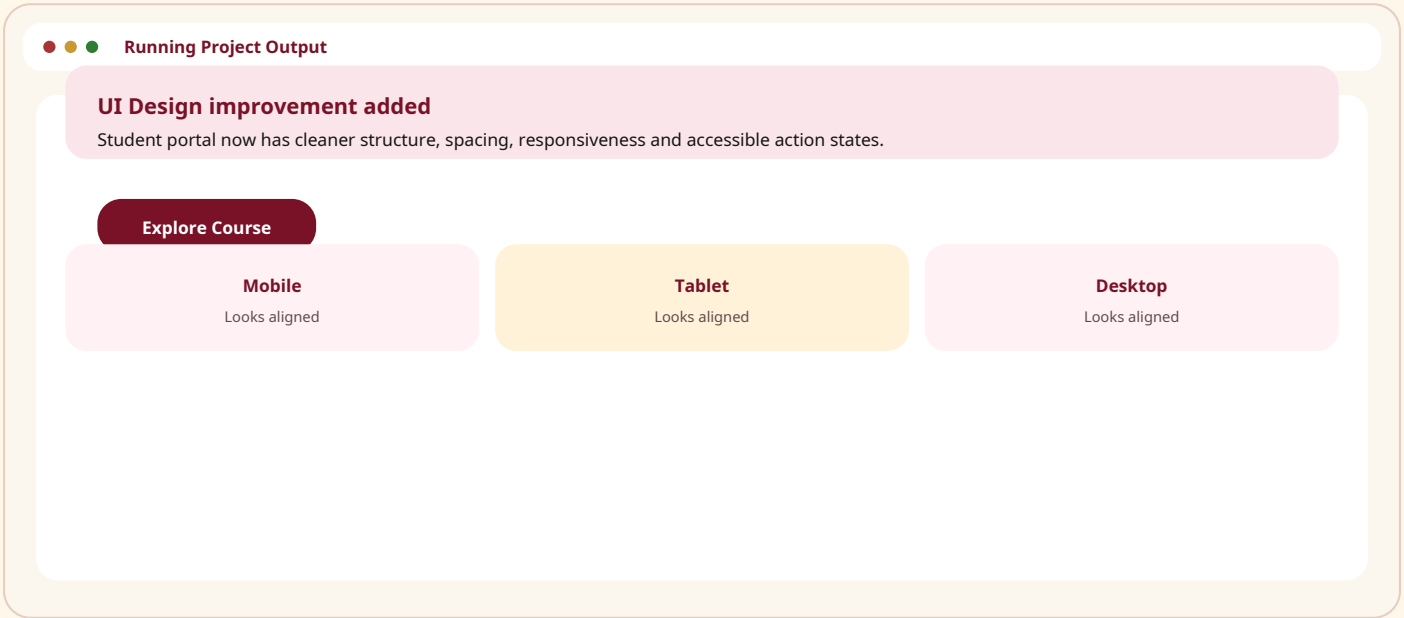
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using font scale line-height color tokens contrast.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-12.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 13: Flexbox Layout Mastery

CSS Layout | display flex gap align justify | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn navbars, cards, aligned controls, responsive rows and common flex bugs. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap navbars, cards, aligned controls, responsive rows and common flex bugs ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Flexbox Layout Mastery: Detailed Notes 1

English: These notes explain the practical engineering decisions behind display flex gap align justify wrap. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Flexbox Layout Mastery means applying display flex gap align justify wrap to build reliable user interfaces.
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where flexbox layout mastery matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-13: add css layout improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Flexbox Layout Mastery: Detailed Notes 2

English: These notes explain the practical engineering decisions behind display flex gap align justify wrap. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where flexbox layout mastery matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-13: add css layout improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Flexbox Layout Mastery

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE style.css

```
.topbar {
display: flex;
align-items: center;
justify-content: space-between;
gap: 1rem;
}
.menu {
display: flex;
flex-wrap: wrap;
gap: .75rem;
}
.actions { margin-left: auto; }
```

```
$ Open layout.html and inspect topbar
```

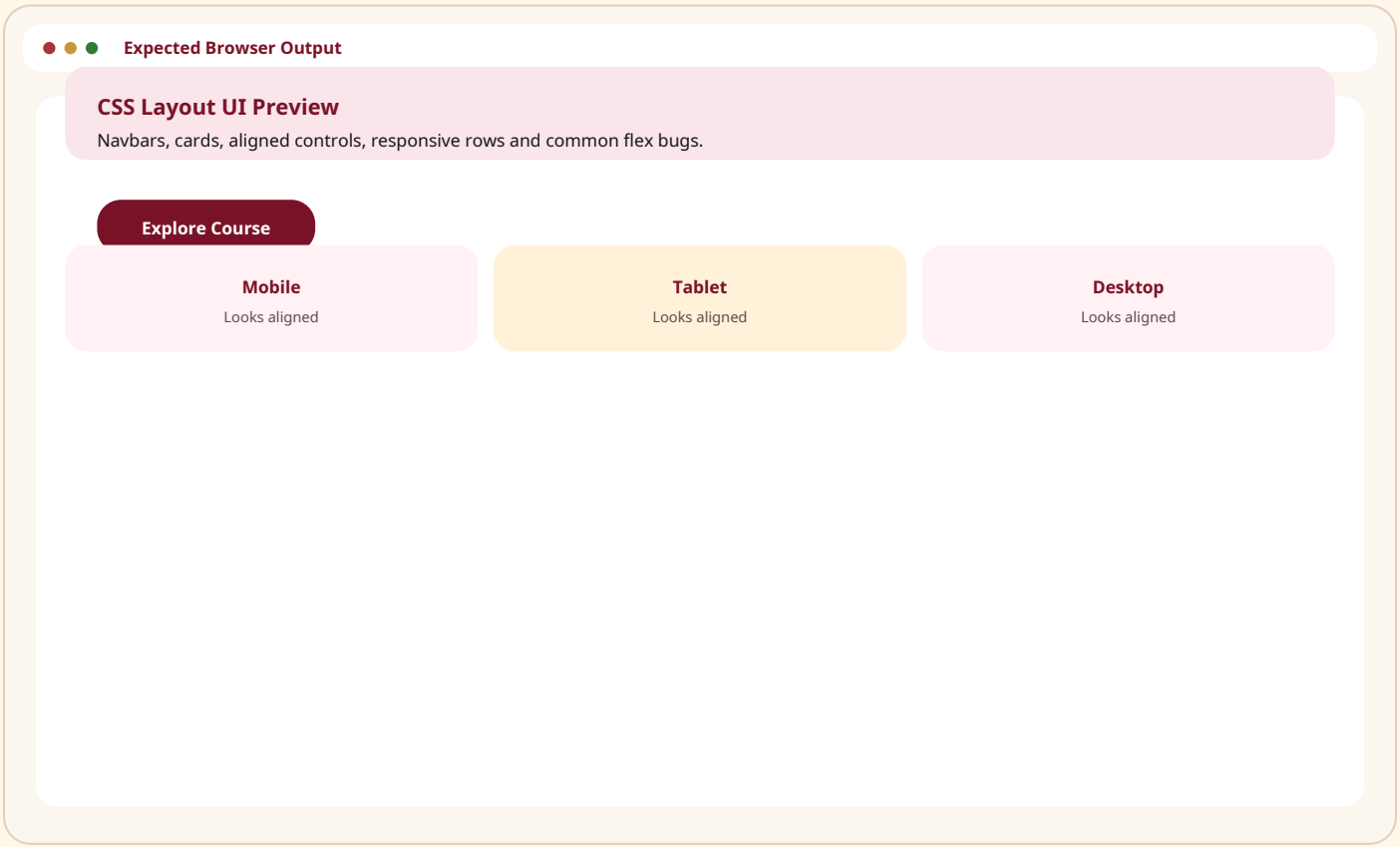
RUN AND UNDERSTAND

- Expected output: Logo, menu and actions align neatly without manual margin hacks.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Flexbox Layout Mastery

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Flexbox Layout Mastery

HANDS-ON TASK

- Build one screen section that demonstrates navbars, cards, aligned controls, responsive rows and common flex bugs.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-13-flexbox-layout-mastery
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Flexbox Layout Mastery

WRITTEN QUESTIONS

- Q1. What problem does Flexbox Layout Mastery solve in real UI engineering?
- Q2. Which file/language is most commonly involved with display flex gap align justify wrap?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where flexbox layout mastery is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Flexbox Layout Mastery

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Flexbox Layout Mastery

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using display flex gap align justify wrap.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-13.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

CSS Layout improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 14: CSS Grid Layout Mastery

CSS Layout | grid columns areas minmax auto | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn dashboard grids, galleries, responsive sections and page composition. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap dashboard grids, galleries, responsive sections and page composition ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

CSS Grid Layout Mastery: Detailed Notes 1

English: These notes explain the practical engineering decisions behind grid columns areas minmax auto-fit. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: CSS Grid Layout Mastery means applying grid columns areas minmax auto-fit to build reliable user ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where css grid layout mastery matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-14: add css layout improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

CSS Grid Layout Mastery: Detailed Notes 2

English: These notes explain the practical engineering decisions behind grid columns areas minmax auto-fit. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where css grid layout mastery matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-14: add css layout improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: CSS Grid Layout Mastery

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE style.css

```
.course-grid {
display: grid;
grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
gap: 1rem;
}
.course-card {
border: 1px solid #e6cfc0;
border-radius: 1.25rem;
padding: 1rem;
background: #fffdf8;
}
@media (max-width: 640px) {
.course-grid { grid-template-columns: 1fr; }
}
```

```
$ Open index.html and resize browser
```

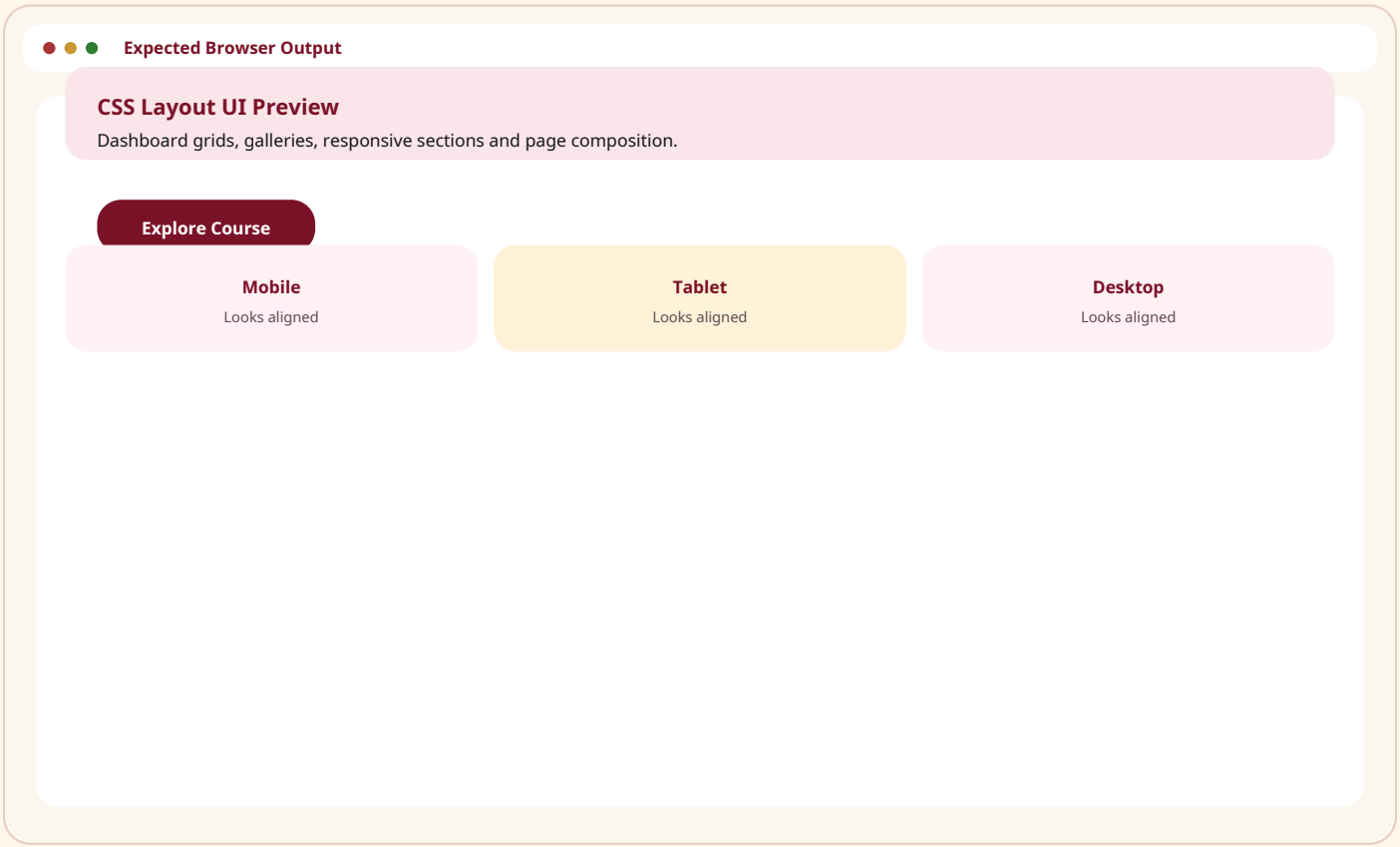
RUN AND UNDERSTAND

- Expected output: Cards automatically move from multi-column desktop grid to single-column mobile list.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: CSS Grid Layout Mastery

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: CSS Grid Layout Mastery

HANDS-ON TASK

- Build one screen section that demonstrates dashboard grids, galleries, responsive sections and page composition.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-14-css-grid-layout-mastery
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: CSS Grid Layout Mastery

WRITTEN QUESTIONS

- Q1. What problem does CSS Grid Layout Mastery solve in real UI engineering?
- Q2. Which file/language is most commonly involved with grid columns areas minmax auto-fit?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where css grid layout mastery is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: CSS Grid Layout Mastery

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: CSS Grid Layout Mastery

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using grid columns areas minmax auto-fit.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-14.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

CSS Layout improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 15: Responsive Design and Breakpoints

Responsive UI | media queries container querie | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn mobile first layout, adaptive typography, touch targets and device testing. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap mobile first layout, adaptive typography, touch targets and device testing ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Responsive Design and Breakpoints: Detailed Notes 1

English: These notes explain the practical engineering decisions behind media queries container queries mobile first. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Responsive Design and Breakpoints means applying media queries container queries mobile first to ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where responsive design and breakpoints matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-15: add responsive ui improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Responsive Design and Breakpoints: Detailed Notes 2

English: These notes explain the practical engineering decisions behind media queries container queries mobile first. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where responsive design and breakpoints matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-15: add responsive ui improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Responsive Design and Breakpoints

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/api.ts

```
export async function getCourses() {
const res = await fetch('/api/courses');
if (!res.ok) throw new Error('Courses failed to load');
return res.json();
}

export async function renderCourses() {
const courses = await getCourses();
console.log('Loaded courses:', courses.length);
}
```

```
$ npm run dev
```

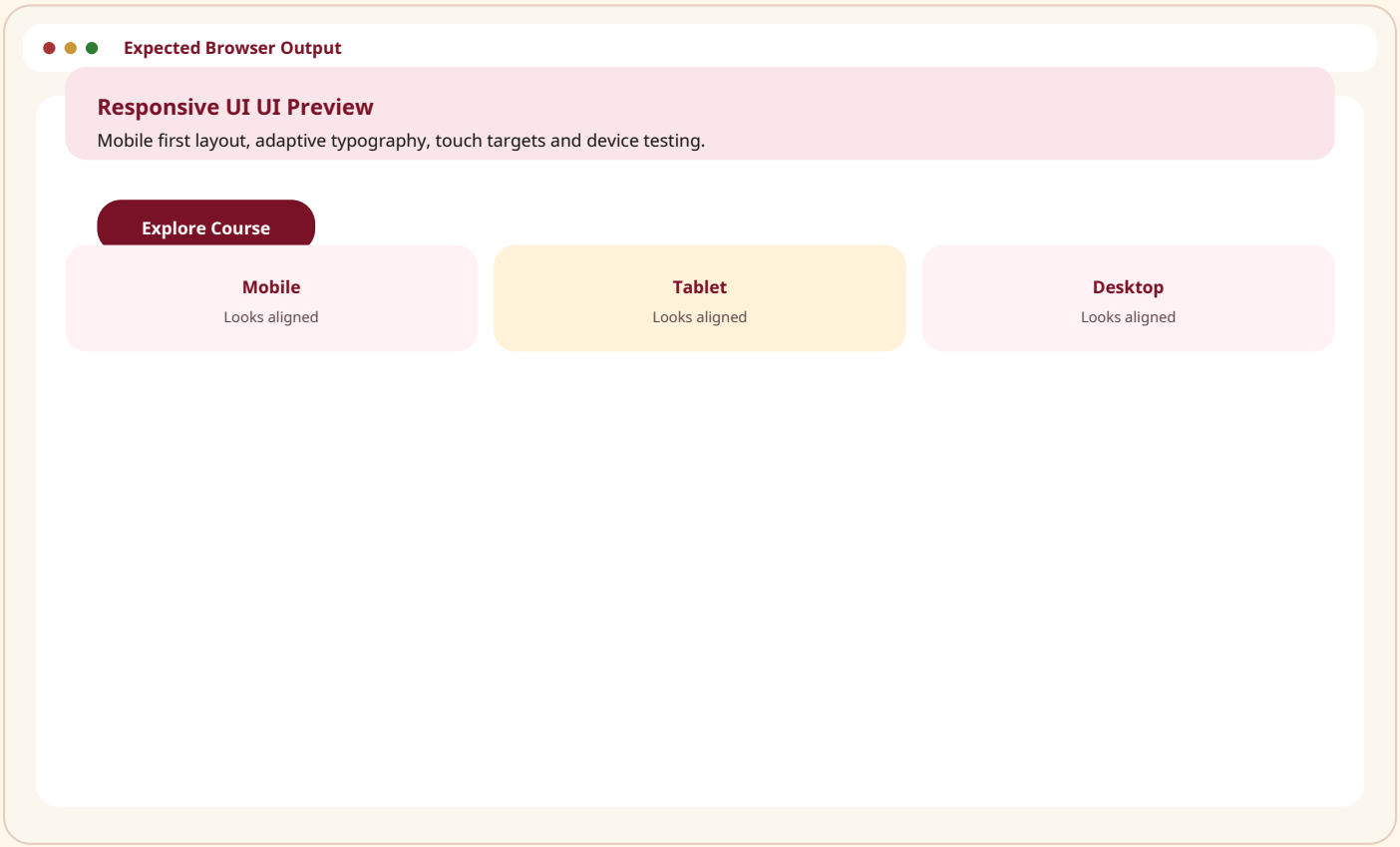
RUN AND UNDERSTAND

- Expected output: Course count logs after successful API call.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Responsive Design and Breakpoints

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Responsive Design and Breakpoints

HANDS-ON TASK

- Build one screen section that demonstrates mobile first layout, adaptive typography, touch targets and device ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-15-responsive-design-and-breakpoints
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Responsive Design and Breakpoints

WRITTEN QUESTIONS

- Q1. What problem does Responsive Design and Breakpoints solve in real UI engineering?
- Q2. Which file/language is most commonly involved with media queries container queries mobile first?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where responsive design and breakpoints is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Responsive Design and Breakpoints

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Responsive Design and Breakpoints

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using media queries container queries mobile first.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-15.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Responsive UI improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 16: Modern CSS Variables and Cascade Layers

CSS Architecture | custom properties @layer :root | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn design tokens, themes, reusable style architecture and dark/light systems. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap design tokens, themes, reusable style architecture and dark/light systems ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Modern CSS Variables and Cascade Layers: Detailed Notes 1

English: These notes explain the practical engineering decisions behind custom properties @layer :root themes. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Modern CSS Variables and Cascade Layers means applying custom properties @layer :root themes to ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where modern css variables and cascade layers matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-16: add css architecture improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Modern CSS Variables and Cascade Layers: Detailed Notes 2

English: These notes explain the practical engineering decisions behind custom properties @layer :root themes. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where modern css variables and cascade layers matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-16: add css architecture improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Modern CSS Variables and Cascade Layers

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE index.html

```
<section class="feature-card">
<p class="eyebrow">UI Engineer Skill</p>
<h2>Build clean interfaces</h2>
<p>Use structure, spacing, contrast and interaction states.</p>
<a href="/learn" class="btn">Learn More</a>
</section>
```

```
$ Open index.html
```

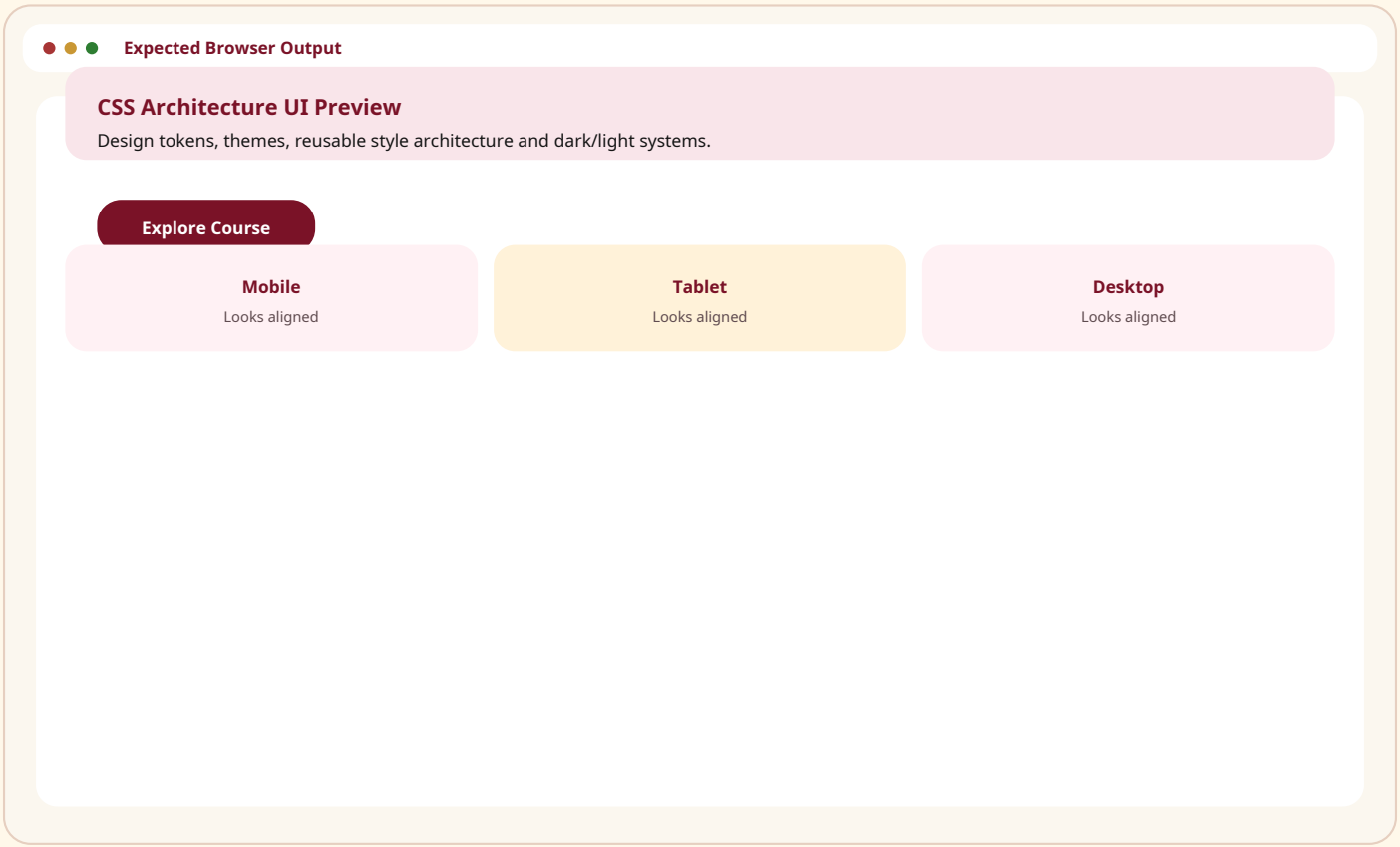
RUN AND UNDERSTAND

- Expected output: Feature card appears with heading, paragraph and link button.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Modern CSS Variables and Cascade Layers

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Modern CSS Variables and Cascade Layers

HANDS-ON TASK

- Build one screen section that demonstrates design tokens, themes, reusable style architecture and dark/light systems.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-16-modern-css-variables-and-cascade-l
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Modern CSS Variables and Cascade Layers

WRITTEN QUESTIONS

- Q1. What problem does Modern CSS Variables and Cascade Layers solve in real UI engineering?
- Q2. Which file/language is most commonly involved with custom properties @layer :root themes?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where modern css variables and cascade layers is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Modern CSS Variables and Cascade Layers

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Modern CSS Variables and Cascade Layers

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using custom properties @layer :root themes.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-16.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

CSS Architecture improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 17: Sass, SCSS and CSS Organization

CSS Tools | Sass SCSS nesting mixins parti | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn structured stylesheets, variables, mixins and refactoring old CSS safely. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap structured stylesheets, variables, mixins and refactoring old CSS safely ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Sass, SCSS and CSS Organization: Detailed Notes 1

English: These notes explain the practical engineering decisions behind Sass SCSS nesting mixins partials. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Sass, SCSS and CSS Organization means applying Sass SCSS nesting mixins partials to build reliable user ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where sass, scss and css organization matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-17: add css tools improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Sass, SCSS and CSS Organization: Detailed Notes 2

English: These notes explain the practical engineering decisions behind Sass SCSS nesting mixins partials. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where sass, scss and css organization matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-17: add css tools improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Sass, SCSS and CSS Organization

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE style.css

```
:root {
--brand: #7a1227;
--cream: #fff8ea;
--text: #151515;
}
.card {
background: var(--cream);
color: var(--text);
border: 1px solid #e6cfc0;
border-radius: 24px;
padding: clamp(1rem, 2vw, 2rem);
}
```

```
$ Open page and inspect styles
```

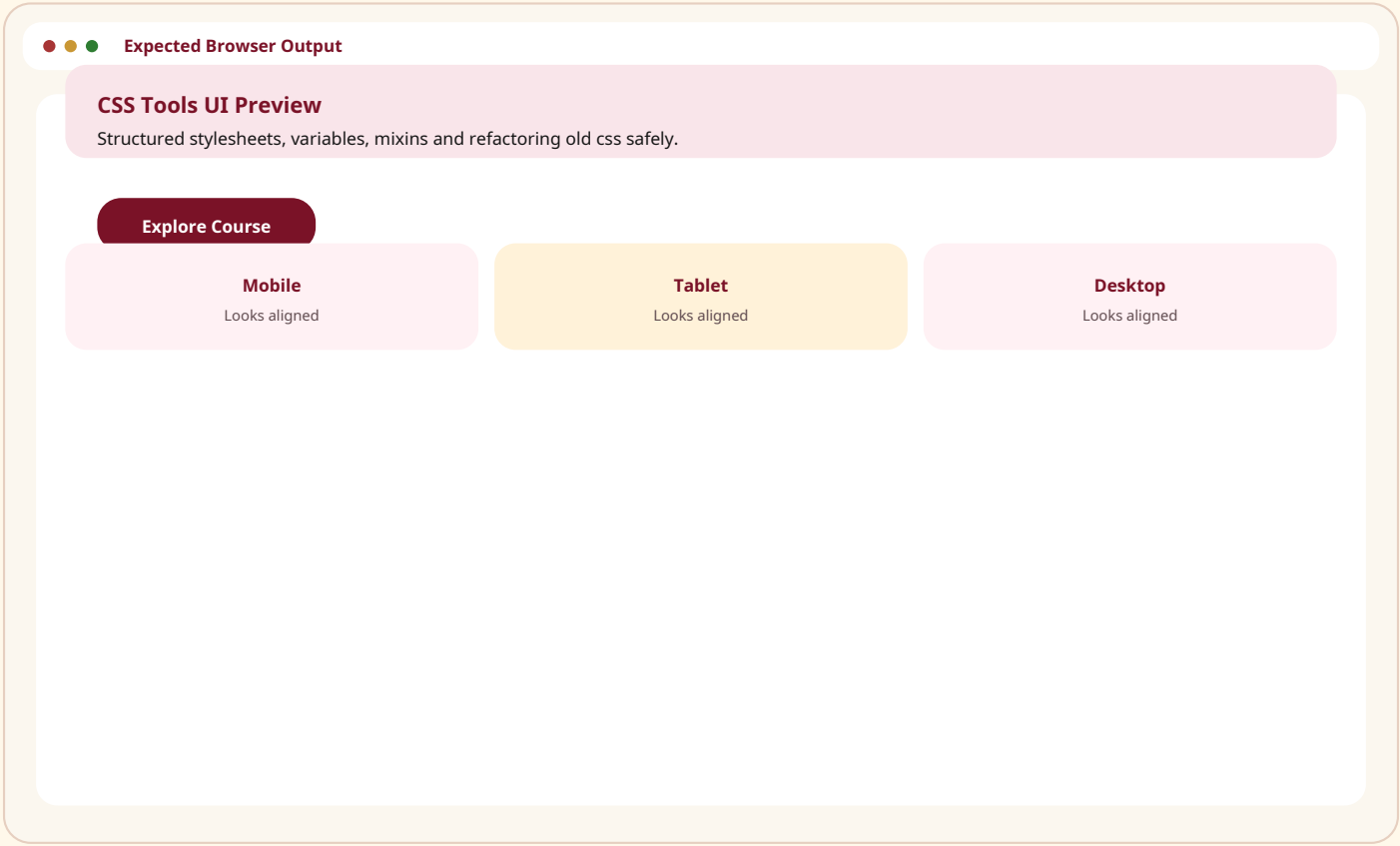
RUN AND UNDERSTAND

- Expected output: Card uses reusable tokens and scales spacing across devices.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Sass, SCSS and CSS Organization

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Sass, SCSS and CSS Organization

HANDS-ON TASK

- Build one screen section that demonstrates structured stylesheets, variables, mixins and refactoring old CSS safely.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-17-sass-scss-and-css-organization
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Sass, SCSS and CSS Organization

WRITTEN QUESTIONS

- Q1. What problem does Sass, SCSS and CSS Organization solve in real UI engineering?
- Q2. Which file/language is most commonly involved with Sass SCSS nesting mixins partials?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where sass, scss and css organization is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Sass, SCSS and CSS Organization

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Sass, SCSS and CSS Organization

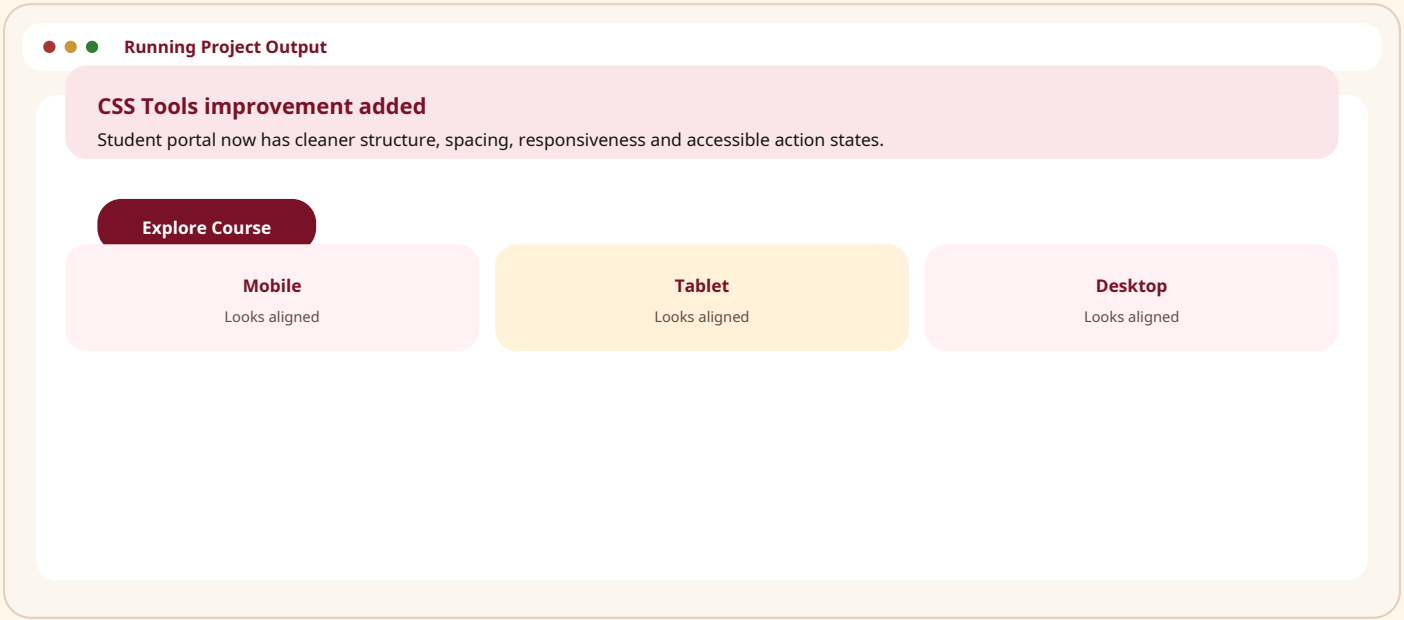
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using Sass SCSS nesting mixins partials.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-17.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 18: Tailwind CSS Utility-First UI

UI Framework | Tailwind responsive utilities | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn utility classes, responsive variants, reusable components and design consistency. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap utility classes, responsive variants, reusable components and design consistency ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Tailwind CSS Utility-First UI: Detailed Notes 1

English: These notes explain the practical engineering decisions behind Tailwind responsive utilities components. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Tailwind CSS Utility-First UI means applying Tailwind responsive utilities components to build reliable ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where tailwind css utility-first ui matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-18: add ui framework improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Tailwind CSS Utility-First UI: Detailed Notes 2

English: These notes explain the practical engineering decisions behind Tailwind responsive utilities components. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where tailwind css utility-first ui matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-18: add ui framework improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Tailwind CSS Utility-First UI

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE app/page.tsx

```
export default function Page() {
return (
<main className="min-h-screen bg-[#fff8ea] p-6">
<section className="mx-auto max-w-4xl rounded-3xl border bg-white p-8 shadow-sm">
<p className="text-sm font-bold text-[#7a1227]">AiWorldAcademy</p>
<h1 className="mt-3 text-4xl font-black text-neutral-950">Frontend UI Engineer</h1>
<p className="mt-3 max-w-xl text-neutral-700">Build responsive, accessible, production-ready interfaces.</p>
<button className="mt-6 rounded-full bg-[#7a1227] px-6 py-3 font-bold text-white">Start Learning</button>
</section>
</main>
);
}
```

```
$ npm run dev
```

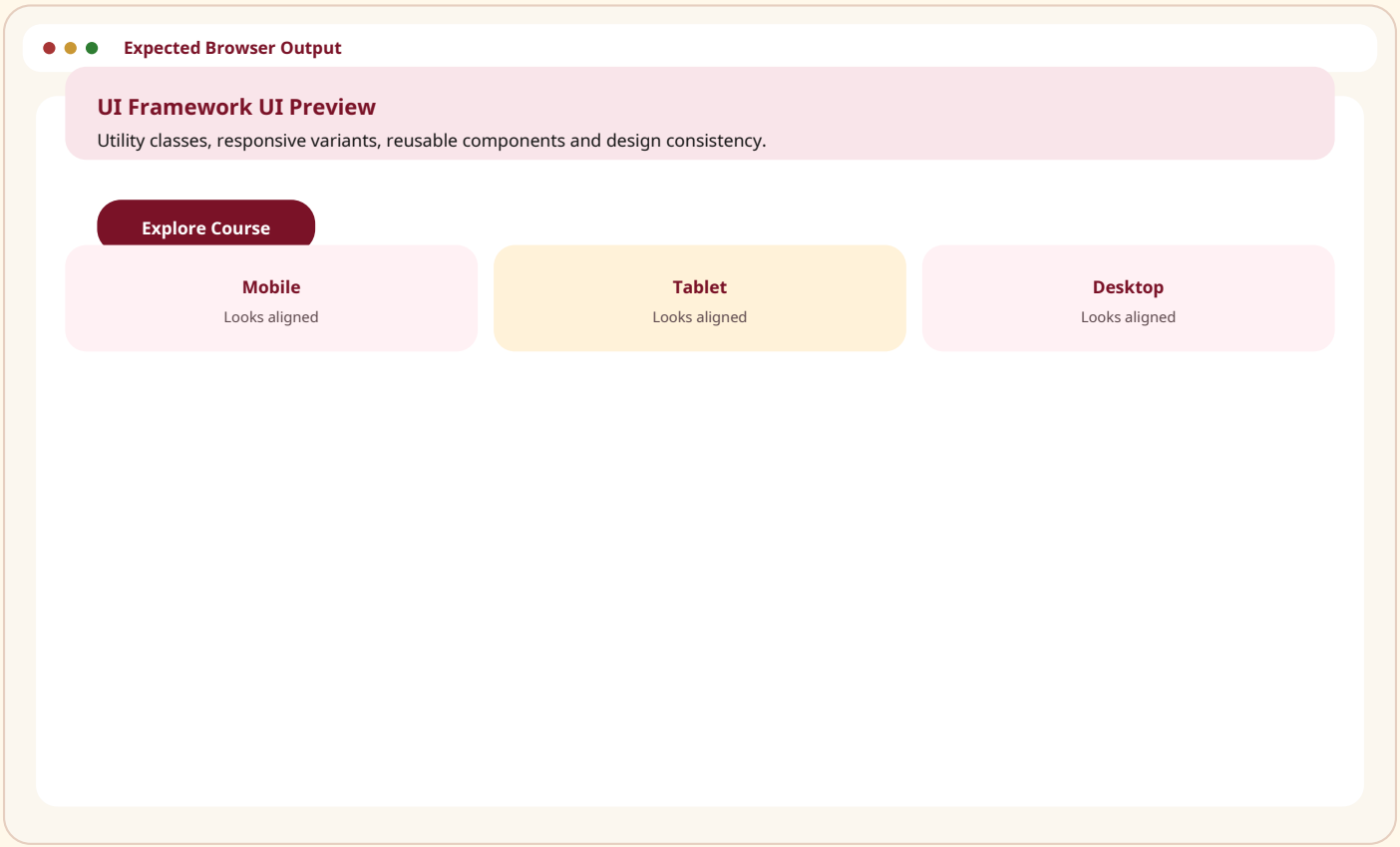
RUN AND UNDERSTAND

- Expected output: Premium creamy/burgundy hero card renders with responsive spacing.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Tailwind CSS Utility-First UI

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Tailwind CSS Utility-First UI

HANDS-ON TASK

- Build one screen section that demonstrates utility classes, responsive variants, reusable components and design ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-18-tailwind-css-utility-first-ui
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Tailwind CSS Utility-First UI

WRITTEN QUESTIONS

- Q1. What problem does Tailwind CSS Utility-First UI solve in real UI engineering?
- Q2. Which file/language is most commonly involved with Tailwind responsive utilities components?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where tailwind css utility-first ui is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Tailwind CSS Utility-First UI

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Tailwind CSS Utility-First UI

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using Tailwind responsive utilities components.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-18.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

UI Framework improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 19: UI Design Principles for Engineers

UI Engineering | layout spacing contrast states | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn design quality, alignment, whitespace, user flow and polished product screens. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap design quality, alignment, whitespace, user flow and polished product screens ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

UI Design Principles for Engineers: Detailed Notes 1

English: These notes explain the practical engineering decisions behind layout spacing contrast states visual hierarchy. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: UI Design Principles for Engineers means applying layout spacing contrast states visual hierarchy to ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where ui design principles for engineers matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-19: add ui engineering improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

UI Design Principles for Engineers: Detailed Notes 2

English: These notes explain the practical engineering decisions behind layout spacing contrast states visual hierarchy. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where ui design principles for engineers matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-19: add ui engineering improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: UI Design Principles for Engineers

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/api.ts

```
export async function getCourses() {
const res = await fetch('/api/courses');
if (!res.ok) throw new Error('Courses failed to load');
return res.json();
}

export async function renderCourses() {
const courses = await getCourses();
console.log('Loaded courses:', courses.length);
}
```

```
$ npm run dev
```

RUN AND UNDERSTAND

- Expected output: Course count logs after successful API call.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: UI Design Principles for Engineers

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.

Expected Browser Output

UI Engineering UI Preview

Design quality, alignment, whitespace, user flow and polished product screens.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: UI Design Principles for Engineers

HANDS-ON TASK

- Build one screen section that demonstrates design quality, alignment, whitespace, user flow and polished product ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-19-ui-design-principles-for-engineers
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: UI Design Principles for Engineers

WRITTEN QUESTIONS

- Q1. What problem does UI Design Principles for Engineers solve in real UI engineering?
- Q2. Which file/language is most commonly involved with layout spacing contrast states visual hierarchy?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where ui design principles for engineers is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: UI Design Principles for Engineers

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: UI Design Principles for Engineers

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using layout spacing contrast states visual hierarchy.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-19.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

UI Engineering improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 20: Design Systems and Figma Handoff

Design System | Figma tokens components varian | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn tokens, atomic design, reusable components and designer-developer handoff. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap tokens, atomic design, reusable components and designer-developer handoff ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Design Systems and Figma Handoff: Detailed Notes 1

English: These notes explain the practical engineering decisions behind Figma tokens components variants Storybook. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Design Systems and Figma Handoff means applying Figma tokens components variants Storybook to ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where design systems and figma handoff matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-20: add design system improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Design Systems and Figma Handoff: Detailed Notes 2

English: These notes explain the practical engineering decisions behind Figma tokens components variants Storybook. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where design systems and figma handoff matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-20: add design system improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Design Systems and Figma Handoff

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE index.html

```
<section class="feature-card">
<p class="eyebrow">UI Engineer Skill</p>
<h2>Build clean interfaces</h2>
<p>Use structure, spacing, contrast and interaction states.</p>
<a href="/learn" class="btn">Learn More</a>
</section>
```

```
$ Open index.html
```

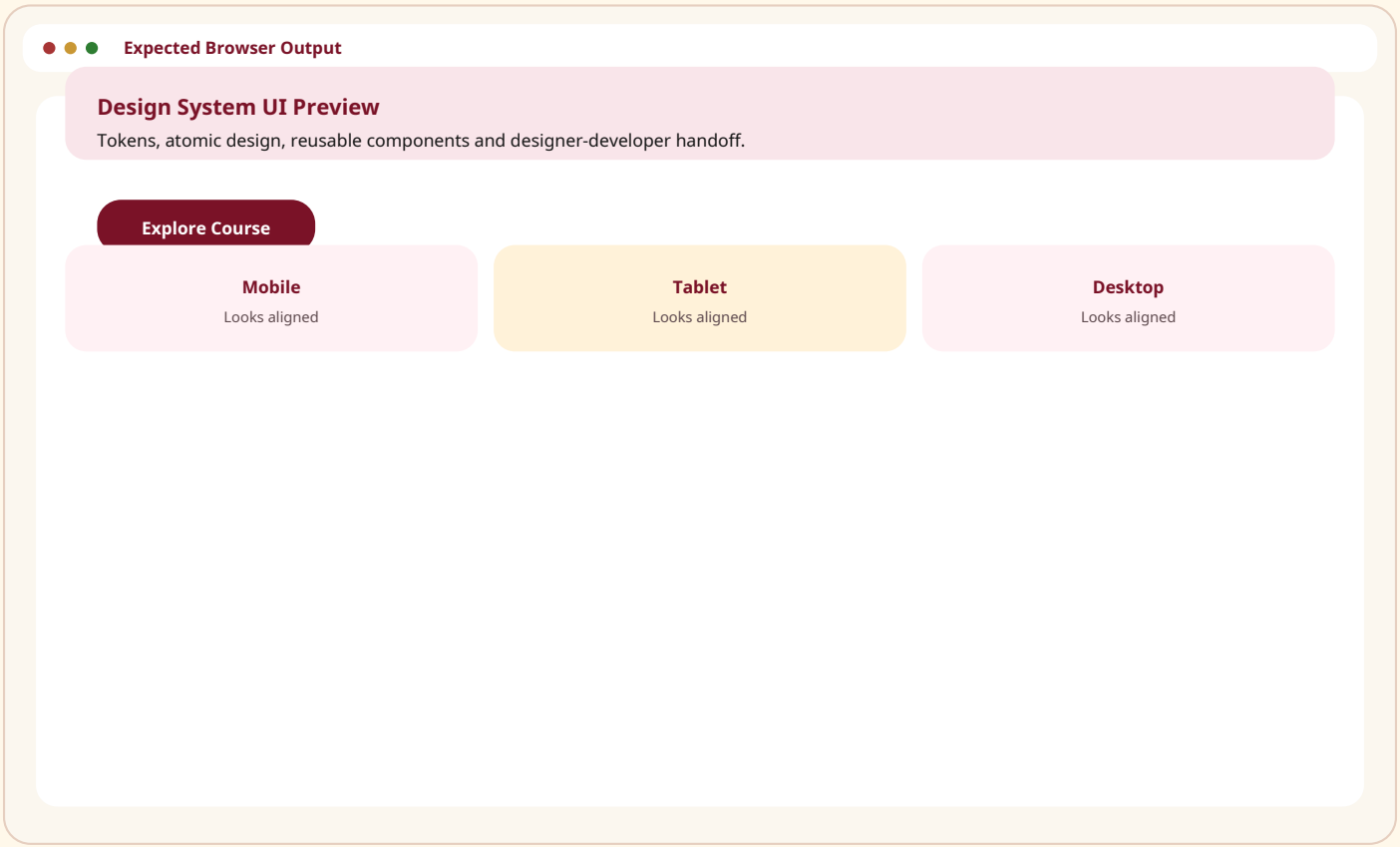
RUN AND UNDERSTAND

- Expected output: Feature card appears with heading, paragraph and link button.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Design Systems and Figma Handoff

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Design Systems and Figma Handoff

HANDS-ON TASK

- Build one screen section that demonstrates tokens, atomic design, reusable components and designer-developer ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-20-design-systems-and-figma-handoff
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Design Systems and Figma Handoff

WRITTEN QUESTIONS

- Q1. What problem does Design Systems and Figma Handoff solve in real UI engineering?
- Q2. Which file/language is most commonly involved with Figma tokens components variants Storybook?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where design systems and figma handoff is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Design Systems and Figma Handoff

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Design Systems and Figma Handoff

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using Figma tokens components variants Storybook.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-20.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Design System improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 21: Accessibility Foundations

A11y | WCAG semantic labels keyboard | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn keyboard support, readable contrast, alt text and accessible component behavior. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap keyboard support, readable contrast, alt text and accessible component behavior ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Accessibility Foundations: Detailed Notes 1

English: These notes explain the practical engineering decisions behind WCAG semantic labels keyboard focus. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Accessibility Foundations means applying WCAG semantic labels keyboard focus to build reliable user ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where accessibility foundations matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-21: add a11y improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Accessibility Foundations: Detailed Notes 2

English: These notes explain the practical engineering decisions behind WCAG semantic labels keyboard focus. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where accessibility foundations matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-21: add a11y improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Accessibility Foundations

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE components/Modal.tsx

```
export function Modal({ open, onClose, children }) {
if (!open) return null;
return (
<div className="overlay" role="presentation" onMouseDown={onClose}>
<section
role="dialog"
aria-modal="true"
aria-labelledby="modal-title"
className="modal"
onMouseDown={(event) => event.stopPropagation()}
>
<h2 id="modal-title">Course Preview</h2>
{children}
<button type="button" onClick={onClose}>Close</button>
</section>
</div>
);
}
```

```
$ npm run test:a11y
```

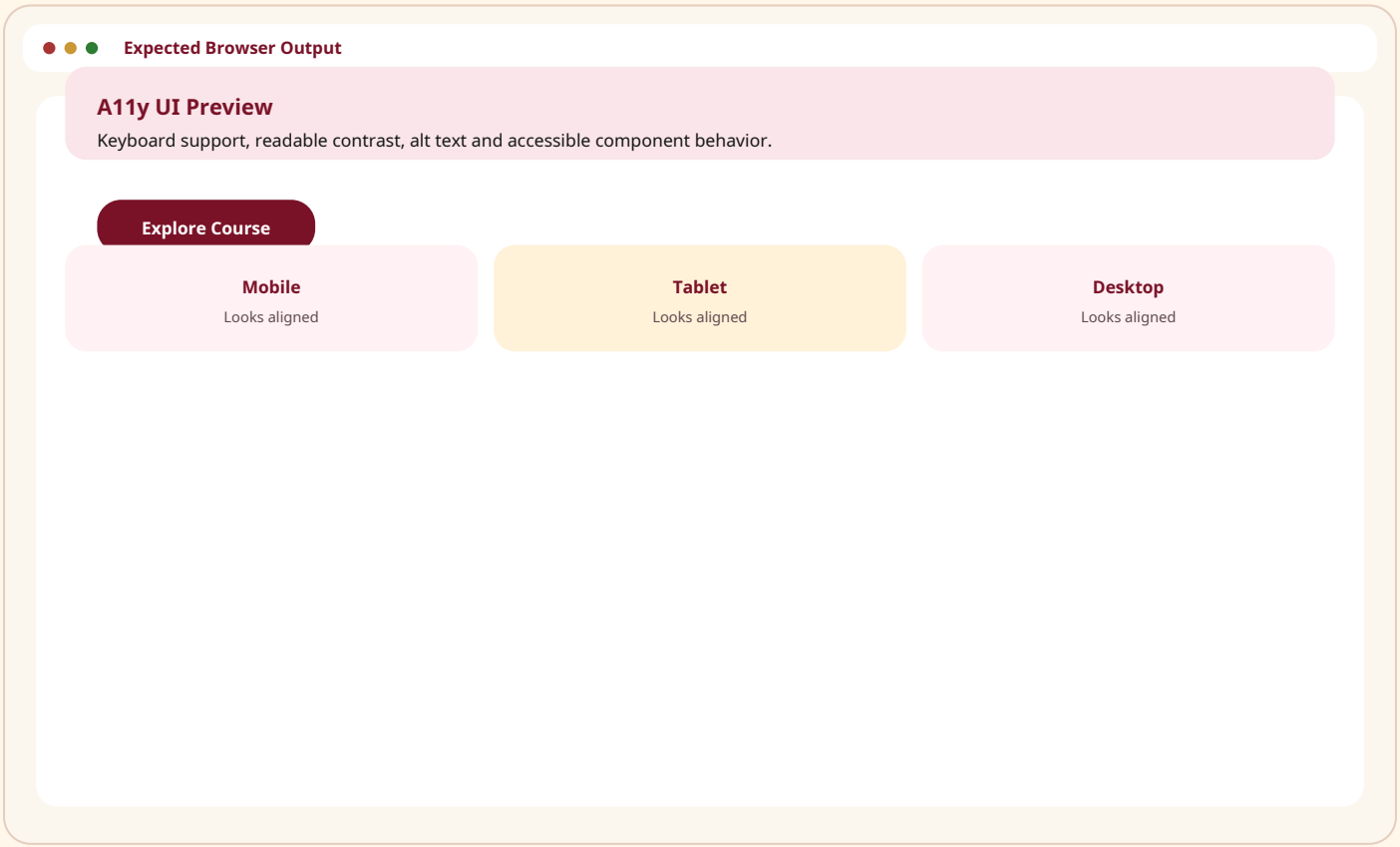
RUN AND UNDERSTAND

- Expected output: Dialog has correct role, title, close button and keyboard-friendly structure.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Accessibility Foundations

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Accessibility Foundations

HANDS-ON TASK

- Build one screen section that demonstrates keyboard support, readable contrast, alt text and accessible ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-21-accessibility-foundations
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Accessibility Foundations

WRITTEN QUESTIONS

- Q1. What problem does Accessibility Foundations solve in real UI engineering?
- Q2. Which file/language is most commonly involved with WCAG semantic labels keyboard focus?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where accessibility foundations is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Accessibility Foundations

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Accessibility Foundations

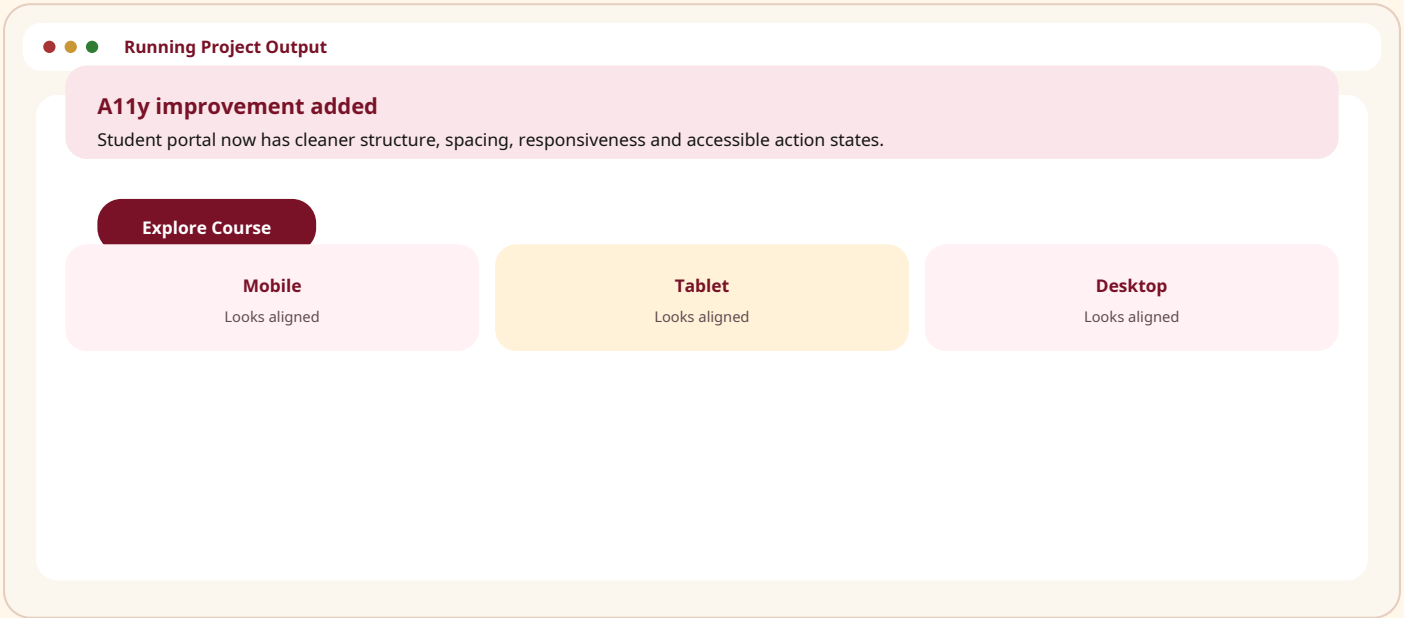
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using WCAG semantic labels keyboard focus.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-21.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 22: Advanced Accessibility Patterns

A11y Advanced | ARIA dialogs menus tabs live r | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn modals, dropdowns, tabs, announcements, focus trap and screen reader UX. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap modals, dropdowns, tabs, announcements, focus trap and screen reader UX ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Advanced Accessibility Patterns: Detailed Notes 1

English: These notes explain the practical engineering decisions behind ARIA dialogs menus tabs live regions. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Advanced Accessibility Patterns means applying ARIA dialogs menus tabs live regions to build reliable user ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where advanced accessibility patterns matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-22: add a11y advanced improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Advanced Accessibility Patterns: Detailed Notes 2

English: These notes explain the practical engineering decisions behind ARIA dialogs menus tabs live regions. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where advanced accessibility patterns matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-22: add a11y advanced improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Advanced Accessibility Patterns

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE components/Modal.tsx

```
export function Modal({ open, onClose, children }) {
if (!open) return null;
return (
<div className="overlay" role="presentation" onMouseDown={onClose}>
<section
role="dialog"
aria-modal="true"
aria-labelledby="modal-title"
className="modal"
onMouseDown={(event) => event.stopPropagation()}
>
<h2 id="modal-title">Course Preview</h2>
{children}
<button type="button" onClick={onClose}>Close</button>
</section>
</div>
);
}
```

```
$ npm run test:a11y
```

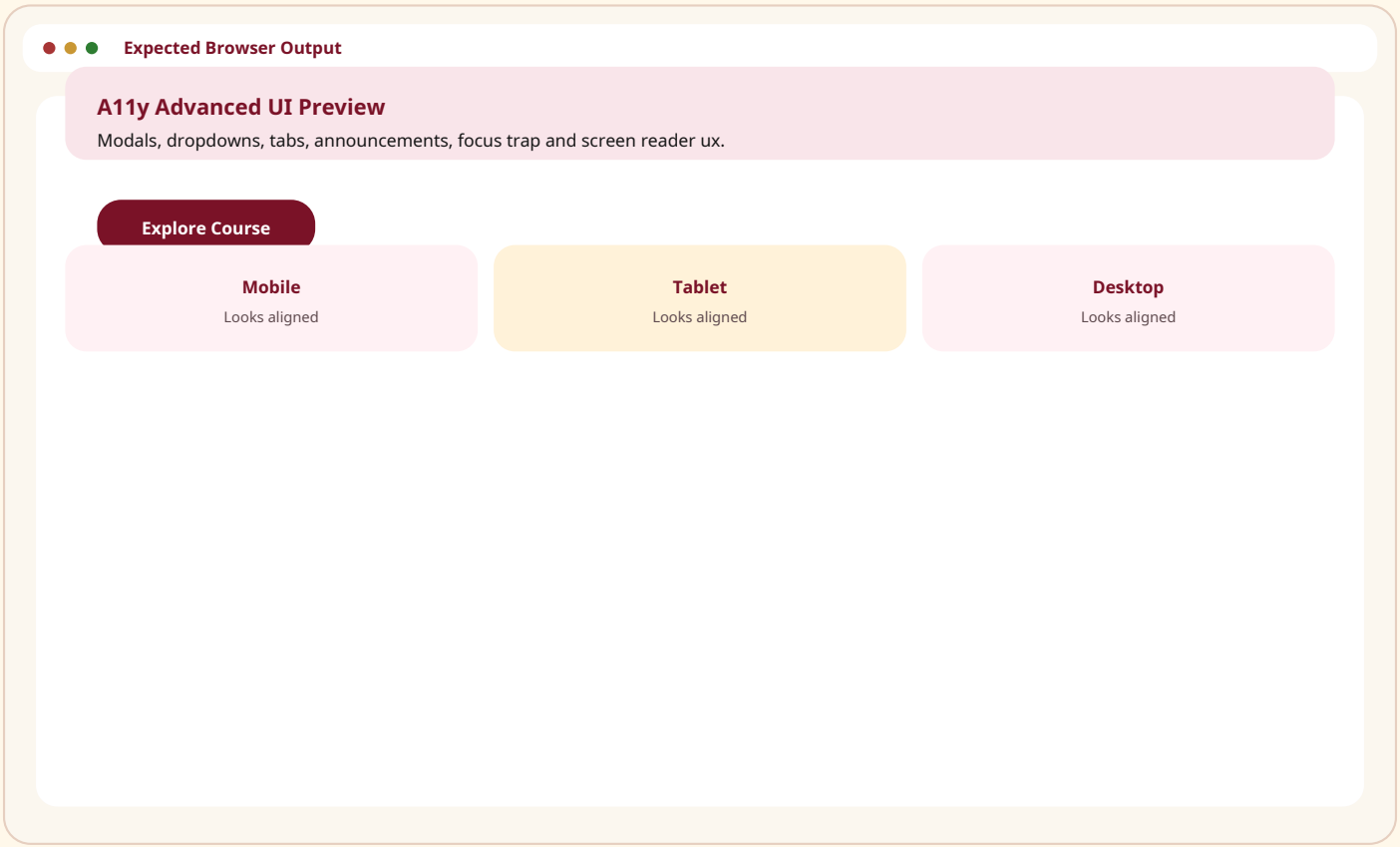
RUN AND UNDERSTAND

- Expected output: Dialog has correct role, title, close button and keyboard-friendly structure.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Advanced Accessibility Patterns

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Advanced Accessibility Patterns

HANDS-ON TASK

- Build one screen section that demonstrates modals, dropdowns, tabs, announcements, focus trap and screen ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-22-advanced-accessibility-patterns
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Advanced Accessibility Patterns

WRITTEN QUESTIONS

- Q1. What problem does Advanced Accessibility Patterns solve in real UI engineering?
- Q2. Which file/language is most commonly involved with ARIA dialogs menus tabs live regions?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where advanced accessibility patterns is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Advanced Accessibility Patterns

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Advanced Accessibility Patterns

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using ARIA dialogs menus tabs live regions.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-22.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

A11y Advanced improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 23: JavaScript Fundamentals for UI

JavaScript | variables arrays objects funct | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn logic foundations, UI state thinking and clean JavaScript syntax. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap logic foundations, UI state thinking and clean JavaScript syntax ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

JavaScript Fundamentals for UI: Detailed Notes 1

English: These notes explain the practical engineering decisions behind variables arrays objects functions conditions. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: JavaScript Fundamentals for UI means applying variables arrays objects functions conditions to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where javascript fundamentals for ui matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-23: add javascript improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

JavaScript Fundamentals for UI: Detailed Notes 2

English: These notes explain the practical engineering decisions behind variables arrays objects functions conditions. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where javascript fundamentals for ui matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-23: add javascript improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: JavaScript Fundamentals for UI

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/api.ts

```
export async function getCourses() {
const res = await fetch('/api/courses');
if (!res.ok) throw new Error('Courses failed to load');
return res.json();
}

export async function renderCourses() {
const courses = await getCourses();
console.log('Loaded courses:', courses.length);
}
```

```
$ npm run dev
```

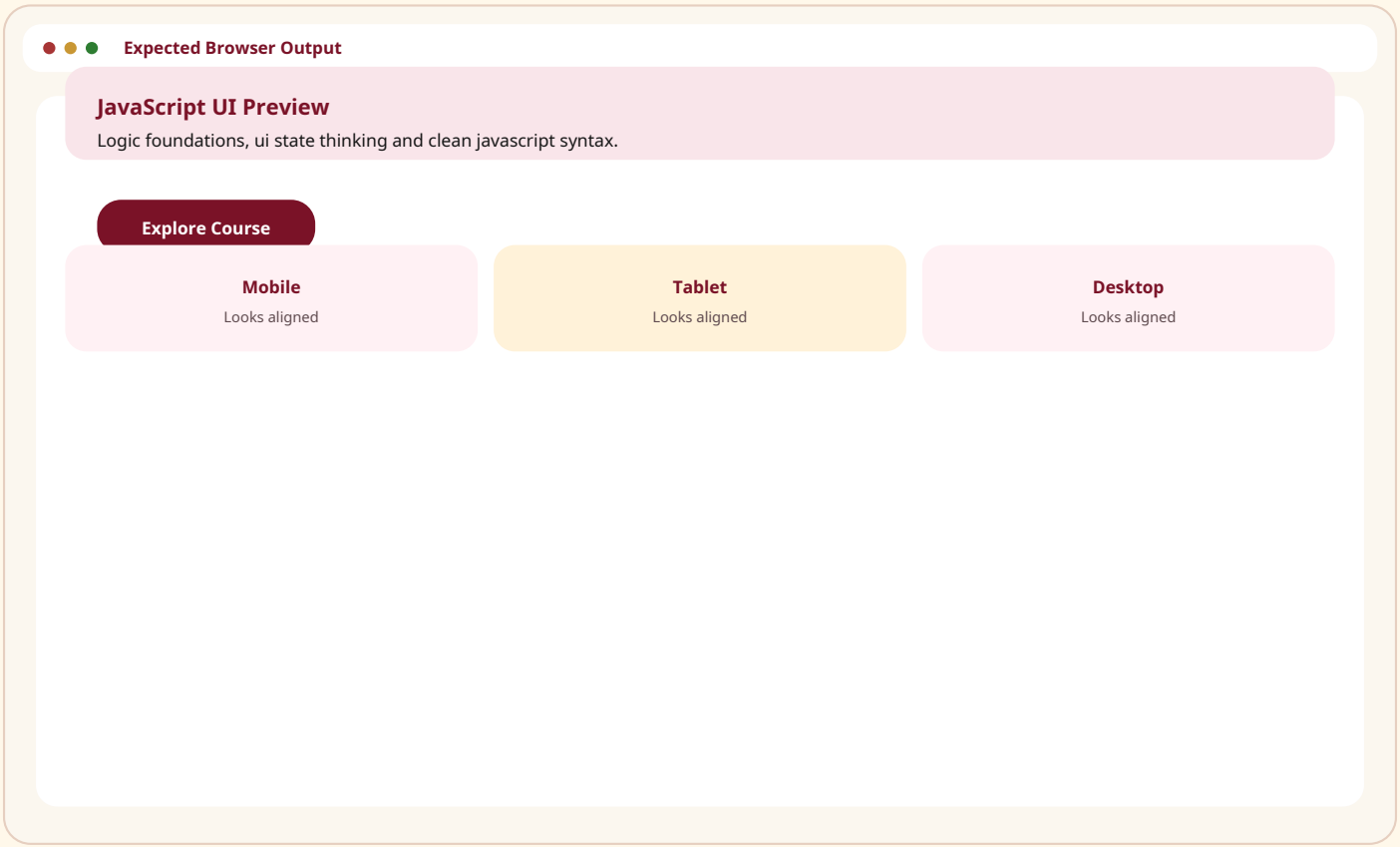
RUN AND UNDERSTAND

- Expected output: Course count logs after successful API call.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: JavaScript Fundamentals for UI

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: JavaScript Fundamentals for UI

HANDS-ON TASK

- Build one screen section that demonstrates logic foundations, UI state thinking and clean JavaScript syntax.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-23-javascript-fundamentals-for-ui
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: JavaScript Fundamentals for UI

WRITTEN QUESTIONS

- Q1. What problem does JavaScript Fundamentals for UI solve in real UI engineering?
- Q2. Which file/language is most commonly involved with variables arrays objects functions conditions?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where javascript fundamentals for ui is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: JavaScript Fundamentals for UI

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: JavaScript Fundamentals for UI

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using variables arrays objects functions conditions.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-23.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

JavaScript improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 24: DOM Manipulation and Events

JavaScript UI | querySelector addEventListener | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn interactive buttons, menus, tabs, accordions and event delegation. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap interactive buttons, menus, tabs, accordions and event delegation ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

DOM Manipulation and Events: Detailed Notes 1

English: These notes explain the practical engineering decisions behind querySelector addEventListener classList dataset. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: DOM Manipulation and Events means applying querySelector addEventListener classList dataset to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where dom manipulation and events matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-24: add javascript ui improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

DOM Manipulation and Events: Detailed Notes 2

English: These notes explain the practical engineering decisions behind querySelector addEventListener classList dataset. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where dom manipulation and events matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-24: add javascript ui improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: DOM Manipulation and Events

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE index.html

```
<section class="feature-card">
<p class="eyebrow">UI Engineer Skill</p>
<h2>Build clean interfaces</h2>
<p>Use structure, spacing, contrast and interaction states.</p>
<a href="/learn" class="btn">Learn More</a>
</section>
```

```
$ Open index.html
```

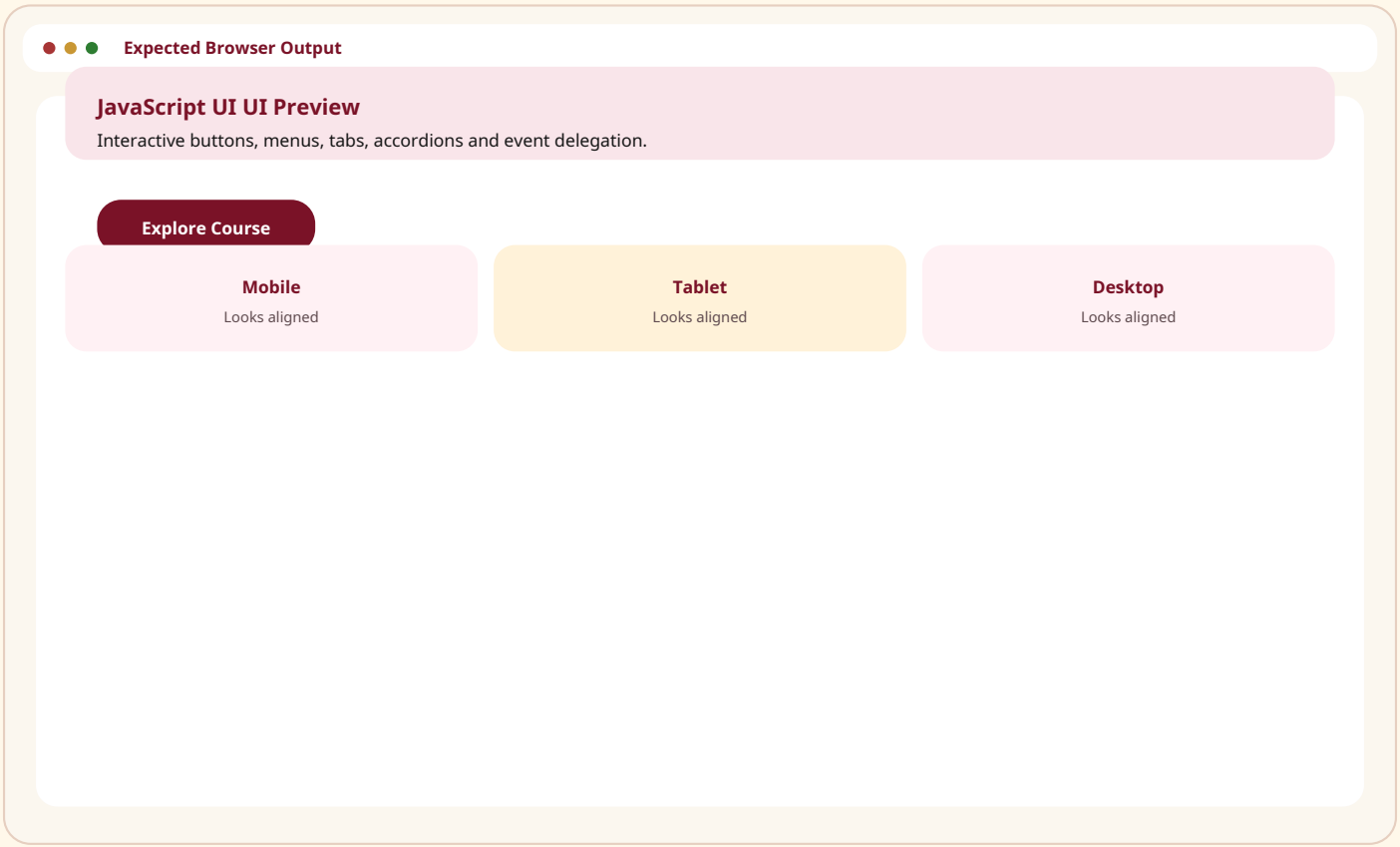
RUN AND UNDERSTAND

- Expected output: Feature card appears with heading, paragraph and link button.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: DOM Manipulation and Events

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: DOM Manipulation and Events

HANDS-ON TASK

- Build one screen section that demonstrates interactive buttons, menus, tabs, accordions and event delegation.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-24-dom-manipulation-and-events
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: DOM Manipulation and Events

WRITTEN QUESTIONS

- Q1. What problem does DOM Manipulation and Events solve in real UI engineering?
- Q2. Which file/language is most commonly involved with querySelector addEventListener classList dataset?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where dom manipulation and events is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: DOM Manipulation and Events

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: DOM Manipulation and Events

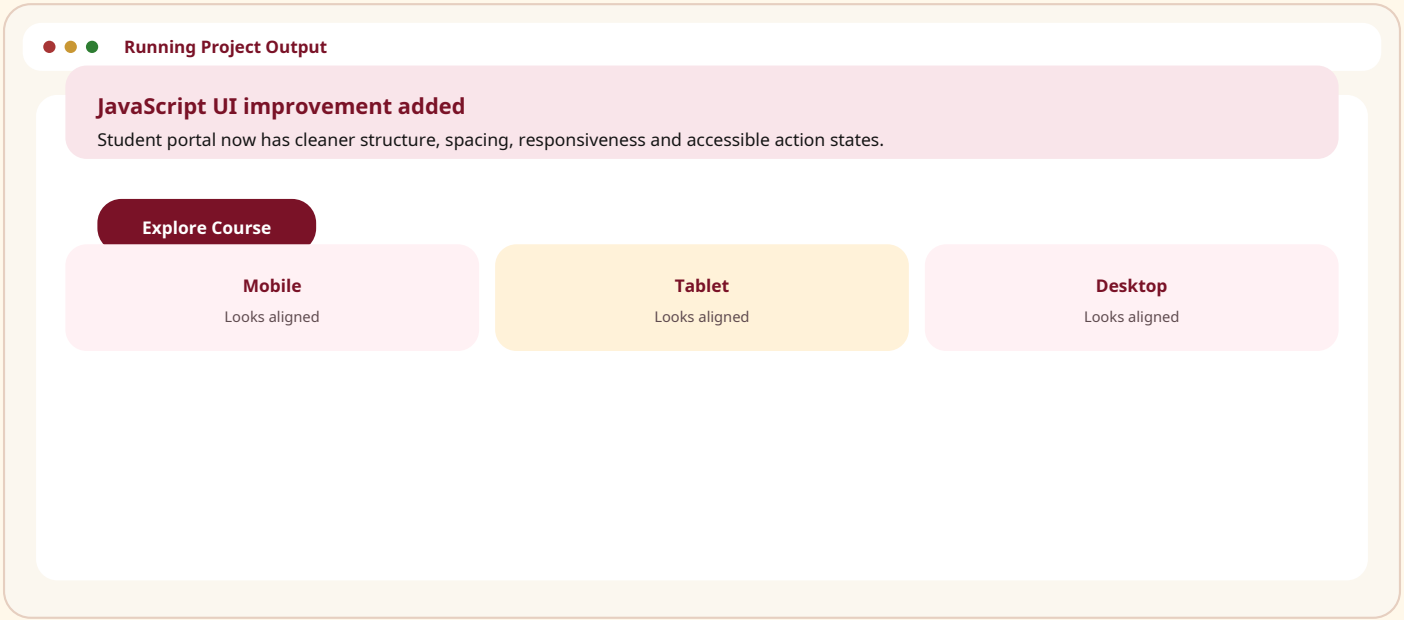
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using querySelector addEventListener classList dataset.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-24.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 25: Form Validation with JavaScript

Forms | constraint validation regex cu | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn client validation, inline errors, submit handling and user-friendly feedback. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap client validation, inline errors, submit handling and user-friendly feedback ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Form Validation with JavaScript: Detailed Notes 1

English: These notes explain the practical engineering decisions behind constraint validation regex custom error. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Form Validation with JavaScript means applying constraint validation regex custom error to build reliable ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where form validation with javascript matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-25: add forms improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Form Validation with JavaScript: Detailed Notes 2

English: These notes explain the practical engineering decisions behind constraint validation regex custom error. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where form validation with javascript matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-25: add forms improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Form Validation with JavaScript

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/forms/register.js

```
const form = document.querySelector('#registerForm');
const email = document.querySelector('#email');
const error = document.querySelector('#emailError');

form.addEventListener('submit', (event) => {
event.preventDefault();
if (!email.validity.valid) {
error.textContent = 'Please enter a valid email address.';
email.setAttribute('aria-invalid', 'true');
return;
}
error.textContent = '';
email.removeAttribute('aria-invalid');
form.submit();
});
```

```
$ Open form page and submit invalid email
```

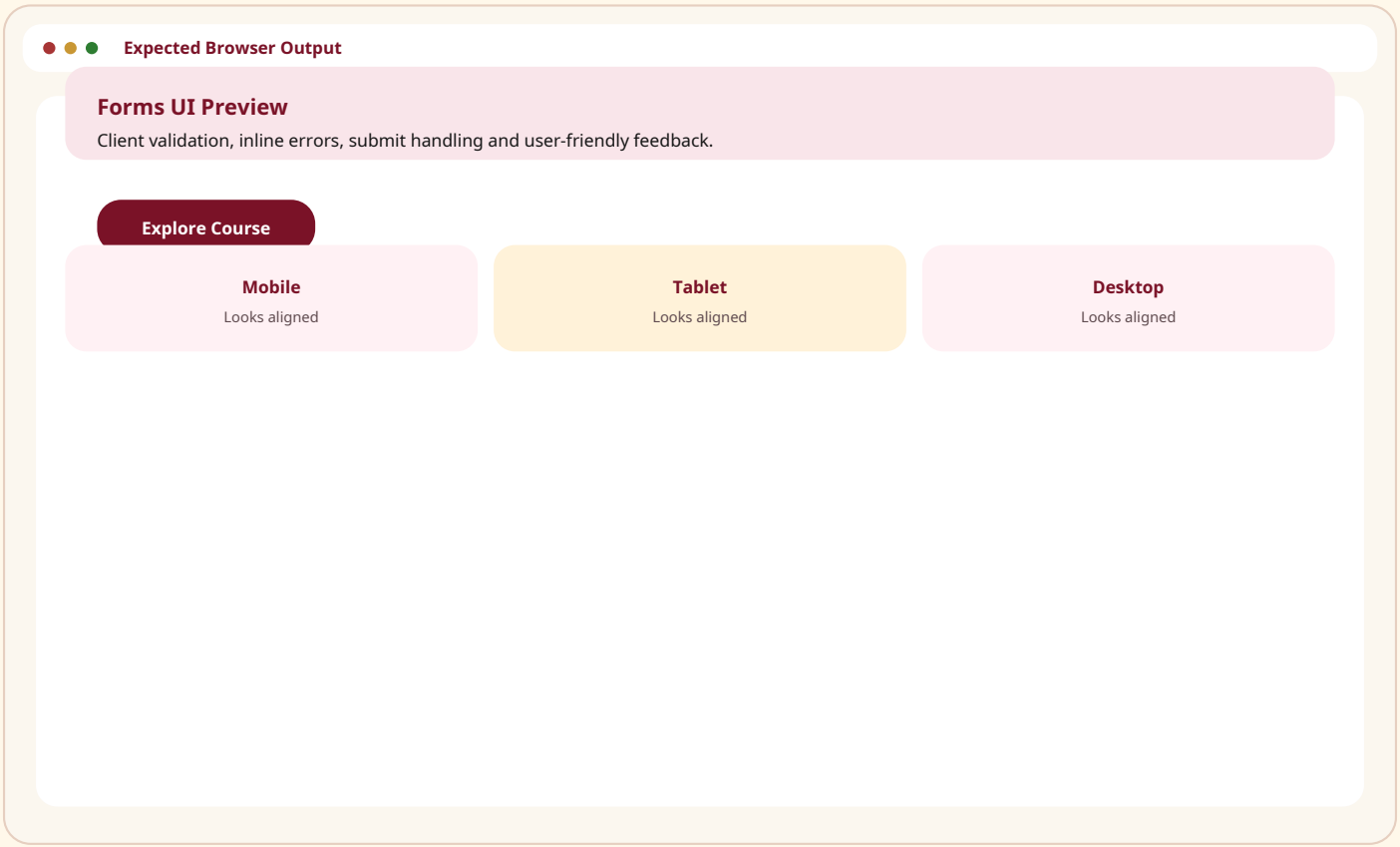
RUN AND UNDERSTAND

- Expected output: Inline error appears and screen reader can identify invalid email field.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Form Validation with JavaScript

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Form Validation with JavaScript

HANDS-ON TASK

- Build one screen section that demonstrates client validation, inline errors, submit handling and user-friendly ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-25-form-validation-with-javascript
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Form Validation with JavaScript

WRITTEN QUESTIONS

- Q1. What problem does Form Validation with JavaScript solve in real UI engineering?
- Q2. Which file/language is most commonly involved with constraint validation regex custom error?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where form validation with javascript is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Form Validation with JavaScript

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Form Validation with JavaScript

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using constraint validation regex custom error.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-25.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Forms improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 26: Async JavaScript and Fetch API

Data UI | Promise async await fetch JSON | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn loading states, error states, API data rendering and retry-friendly UI. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap loading states, error states, API data rendering and retry-friendly UI ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Async JavaScript and Fetch API: Detailed Notes 1

English: These notes explain the practical engineering decisions behind Promise async await fetch JSON. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Async JavaScript and Fetch API means applying Promise async await fetch JSON to build reliable user ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where async javascript and fetch api matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-26: add data ui improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Async JavaScript and Fetch API: Detailed Notes 2

English: These notes explain the practical engineering decisions behind Promise async await fetch JSON. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where async javascript and fetch api matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-26: add data ui improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Async JavaScript and Fetch API

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/main.js

```
const tabs = document.querySelectorAll('[data-tab]');
const panels = document.querySelectorAll('[data-panel]');

tabs.forEach((tab) => {
tab.addEventListener('click', () => {
panels.forEach((panel) => panel.hidden = panel.id !== tab.dataset.tab);
});
});
```

```
$ Open tabs.html and click tabs
```

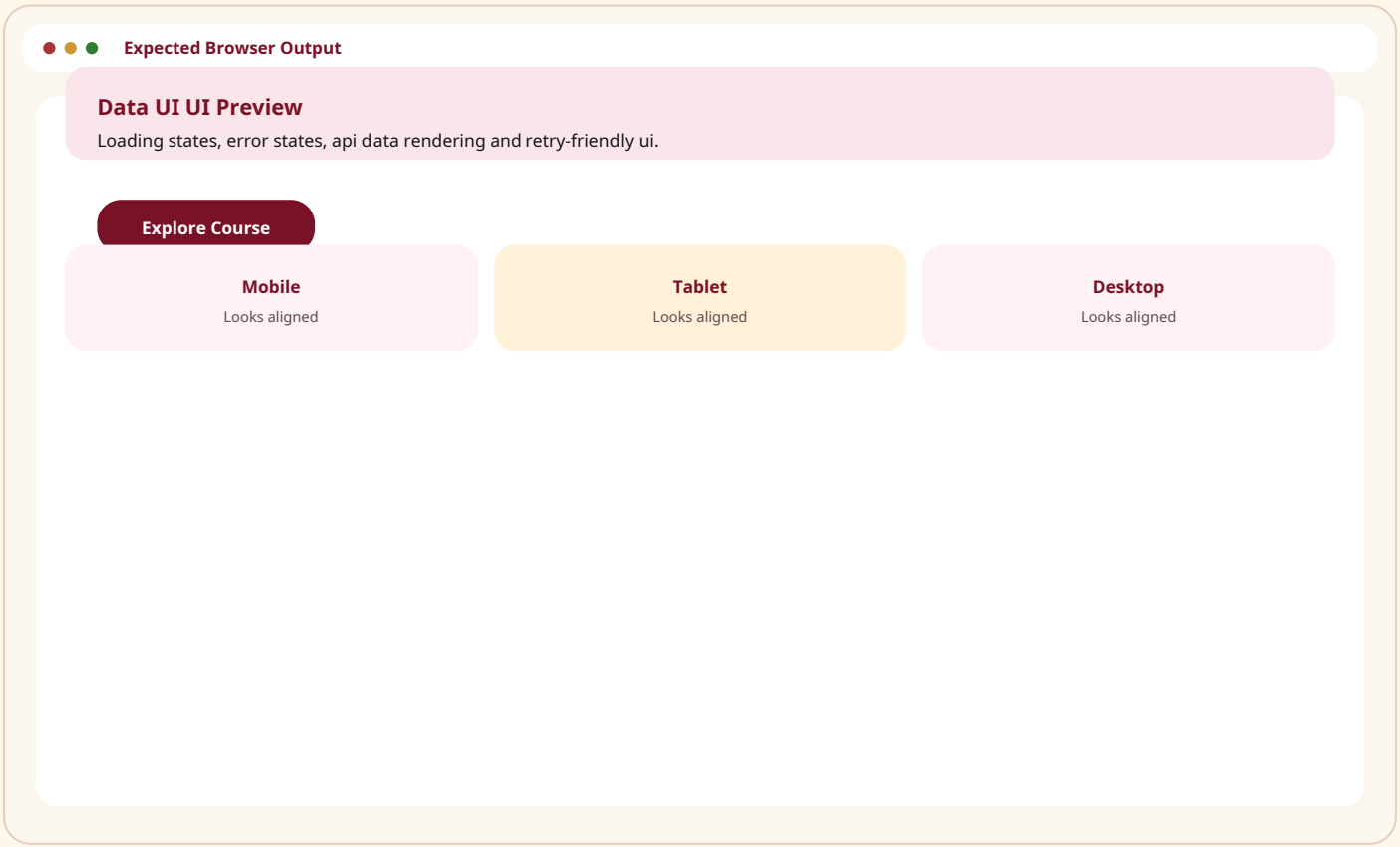
RUN AND UNDERSTAND

- Expected output: Only the selected tab panel remains visible.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Async JavaScript and Fetch API

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Async JavaScript and Fetch API

HANDS-ON TASK

- Build one screen section that demonstrates loading states, error states, API data rendering and retry-friendly UI.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-26-async-javascript-and-fetch-api
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Async JavaScript and Fetch API

WRITTEN QUESTIONS

- Q1. What problem does Async JavaScript and Fetch API solve in real UI engineering?
- Q2. Which file/language is most commonly involved with Promise async await fetch JSON?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where async javascript and fetch api is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Async JavaScript and Fetch API

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Async JavaScript and Fetch API

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using Promise async await fetch JSON.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-26.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Data UI improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 27: Modules, Bundlers and Build Tools

Tooling | ES modules Vite npm scripts pa | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn imports, exports, build pipeline, dependency management and app structure. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap imports, exports, build pipeline, dependency management and app structure ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Modules, Bundlers and Build Tools: Detailed Notes 1

English: These notes explain the practical engineering decisions behind ES modules Vite npm scripts package.json. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Modules, Bundlers and Build Tools means applying ES modules Vite npm scripts package.json to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where modules, bundlers and build tools matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-27: add tooling improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Modules, Bundlers and Build Tools: Detailed Notes 2

English: These notes explain the practical engineering decisions behind ES modules Vite npm scripts package.json. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where modules, bundlers and build tools matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-27: add tooling improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Modules, Bundlers and Build Tools

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/api.ts

```
export async function getCourses() {
const res = await fetch('/api/courses');
if (!res.ok) throw new Error('Courses failed to load');
return res.json();
}

export async function renderCourses() {
const courses = await getCourses();
console.log('Loaded courses:', courses.length);
}
```

```
$ npm run dev
```

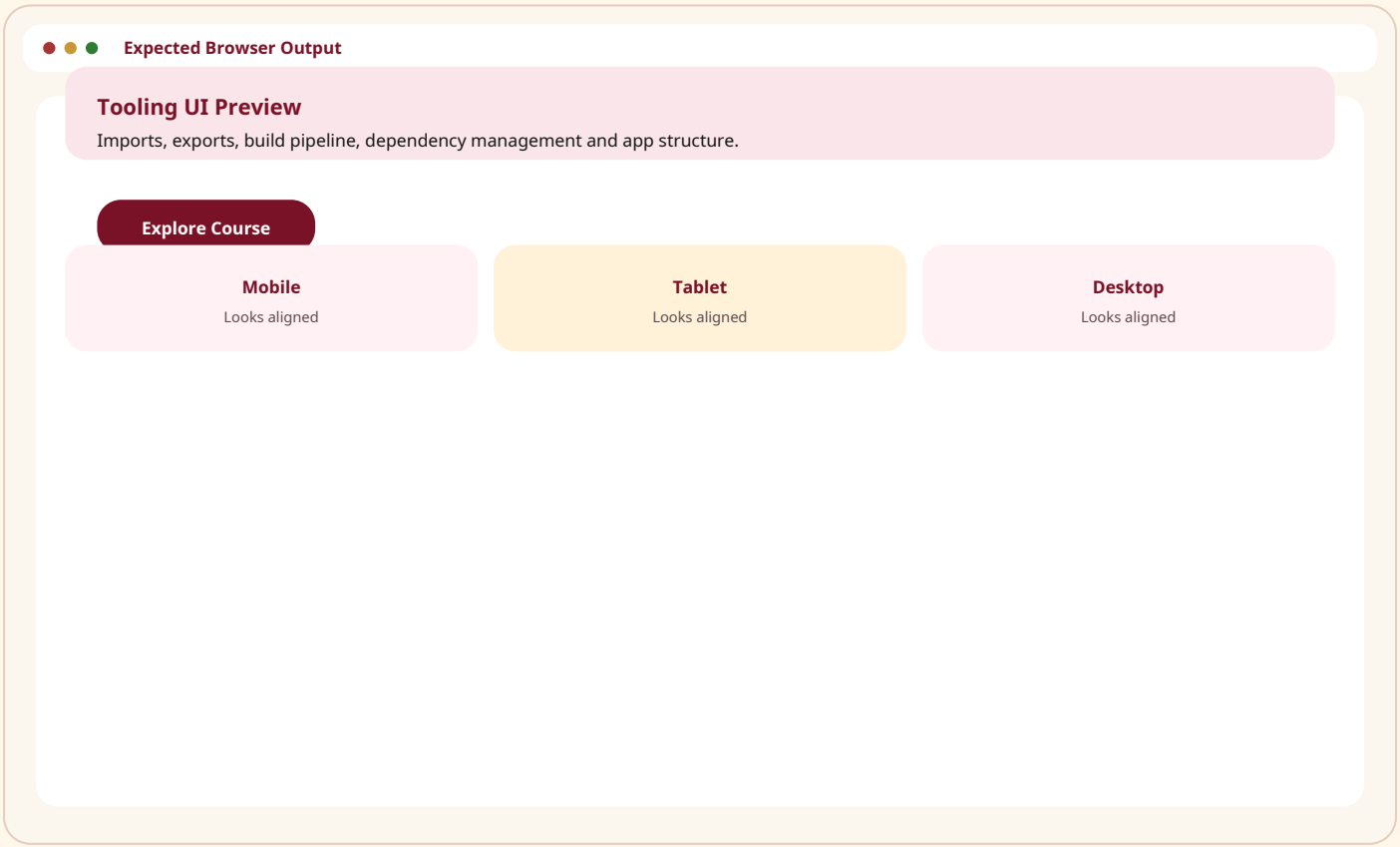
RUN AND UNDERSTAND

- Expected output: Course count logs after successful API call.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Modules, Bundlers and Build Tools

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Modules, Bundlers and Build Tools

HANDS-ON TASK

- Build one screen section that demonstrates imports, exports, build pipeline, dependency management and app ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-27-modules-bundlers-and-build-tools
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Modules, Bundlers and Build Tools

WRITTEN QUESTIONS

- Q1. What problem does Modules, Bundlers and Build Tools solve in real UI engineering?
- Q2. Which file/language is most commonly involved with ES modules Vite npm scripts package.json?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where modules, bundlers and build tools is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Modules, Bundlers and Build Tools

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Modules, Bundlers and Build Tools

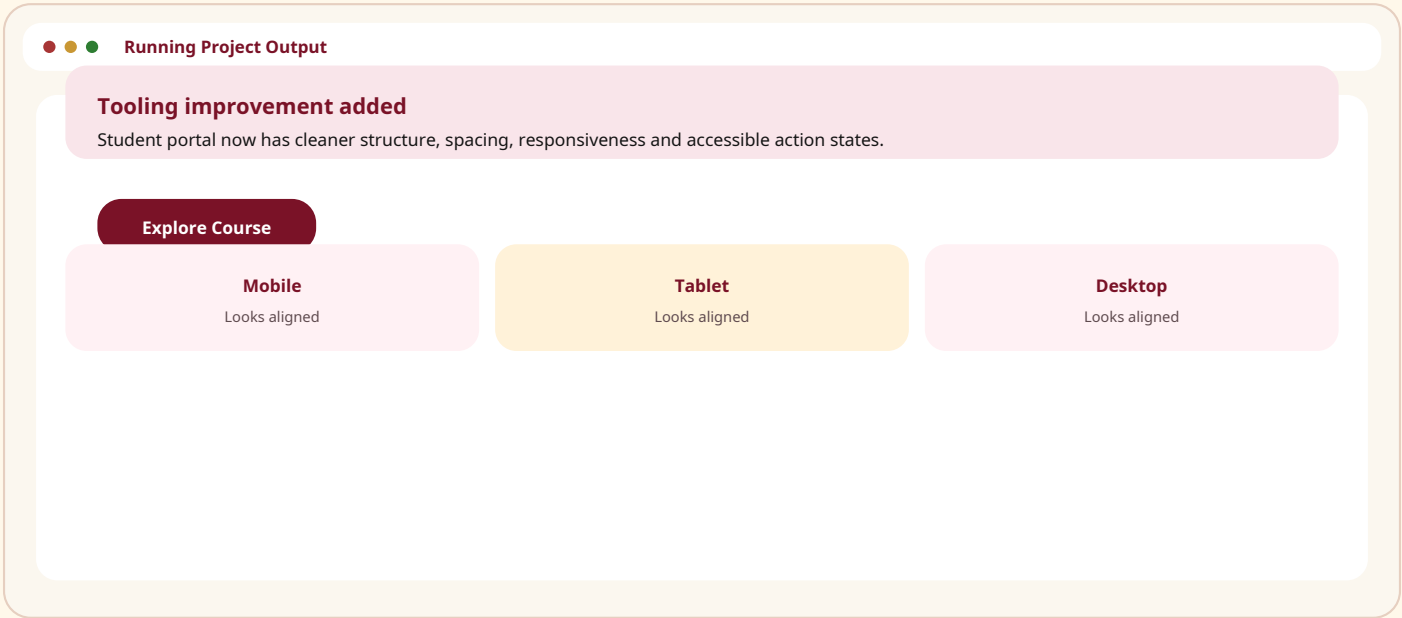
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using ES modules Vite npm scripts package.json.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-27.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 28: TypeScript for UI Engineers

TypeScript | types interfaces generics unio | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn safer components, API typing, props typing and error prevention. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap safer components, API typing, props typing and error prevention ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

TypeScript for UI Engineers: Detailed Notes 1

English: These notes explain the practical engineering decisions behind types interfaces generics unions TSX. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: TypeScript for UI Engineers means applying types interfaces generics unions TSX to build reliable user ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where typescript for ui engineers matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-28: add typescript improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

TypeScript for UI Engineers: Detailed Notes 2

English: These notes explain the practical engineering decisions behind types interfaces generics unions TSX. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where typescript for ui engineers matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-28: add typescript improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: TypeScript for UI Engineers

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE types/course.ts

```
export type CourseLevel = 'Basic' | 'Premium' | 'Pro';

export interface Course {
id: string;
title: string;
level: CourseLevel;
lessons: number;
completedLessons: number;
}

export function progress(course: Course): number {
return Math.round((course.completedLessons / course.lessons) * 100);
}
```

```
$ npm run typecheck
```

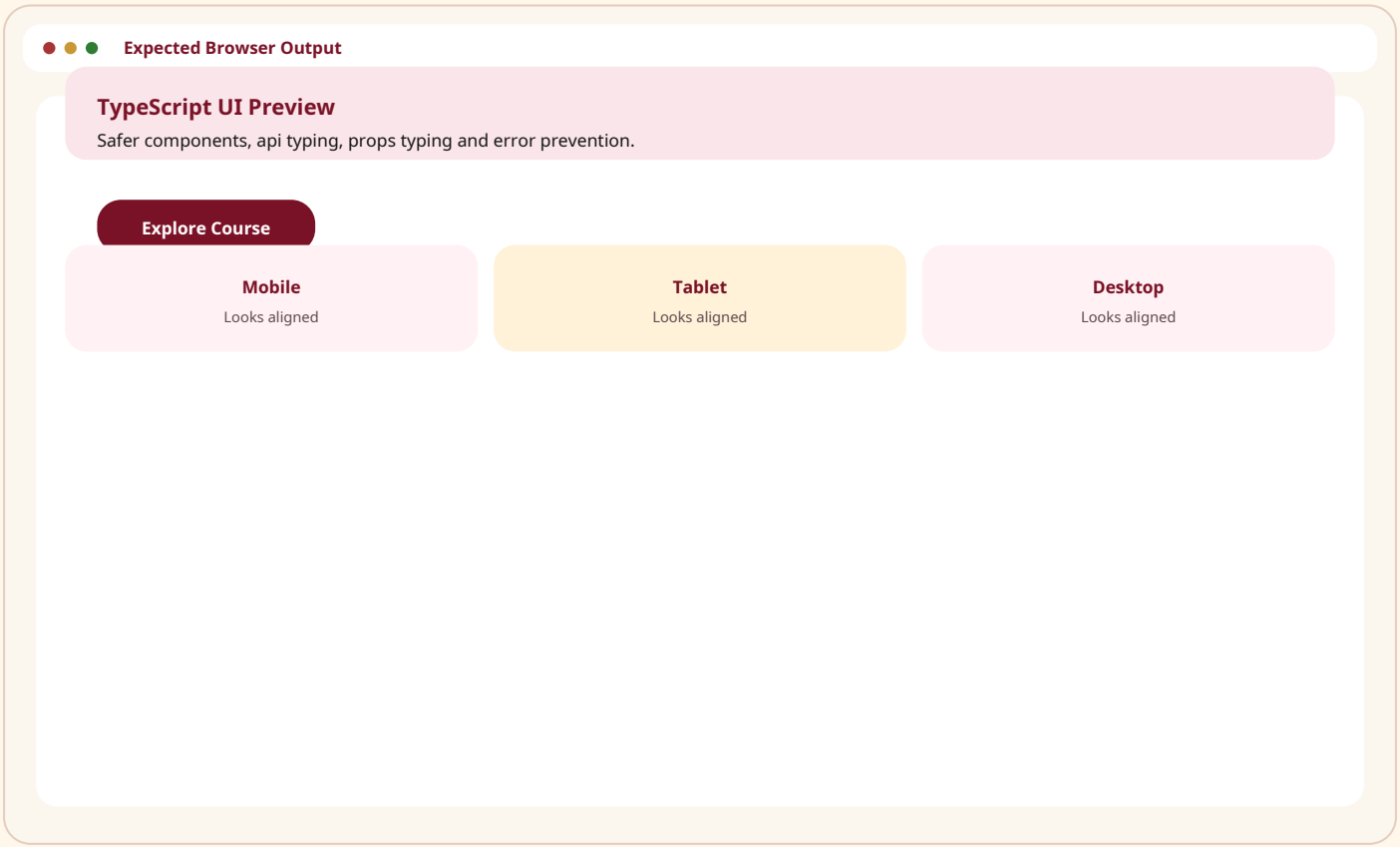
RUN AND UNDERSTAND

- Expected output: TypeScript checks course data and prevents wrong level or missing fields.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: TypeScript for UI Engineers

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: TypeScript for UI Engineers

HANDS-ON TASK

- Build one screen section that demonstrates safer components, API typing, props typing and error prevention.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-28-typescript-for-ui-engineers
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: TypeScript for UI Engineers

WRITTEN QUESTIONS

- Q1. What problem does TypeScript for UI Engineers solve in real UI engineering?
- Q2. Which file/language is most commonly involved with types interfaces generics unions TSX?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where typescript for ui engineers is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: TypeScript for UI Engineers

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: TypeScript for UI Engineers

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using types interfaces generics unions TSX.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-28.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

TypeScript improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 29: React Component Fundamentals

React | React JSX components props chi | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn component thinking, composition, reusable UI units and JSX rules. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap component thinking, composition, reusable UI units and JSX rules ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

React Component Fundamentals: Detailed Notes 1

English: These notes explain the practical engineering decisions behind React JSX components props children. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: React Component Fundamentals means applying React JSX components props children to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where react component fundamentals matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-29: add react improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

React Component Fundamentals: Detailed Notes 2

English: These notes explain the practical engineering decisions behind React JSX components props children. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where react component fundamentals matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-29: add react improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: React Component Fundamentals

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE components/CourseCard.tsx

```
type CourseCardProps = {
title: string;
level: 'Beginner' | 'Intermediate' | 'Advanced';
progress: number;
};

export function CourseCard({ title, level, progress }: CourseCardProps) {
return (
<article className="course-card" aria-label={`${title} course`}>
<span className="badge">{level}</span>
<h2>{title}</h2>
<progress value={progress} max={100} aria-label="Course progress" />
<a href="/student/courses/frontend-ui">Continue learning</a>
</article>
);
}
```

```
$ npm run dev
```

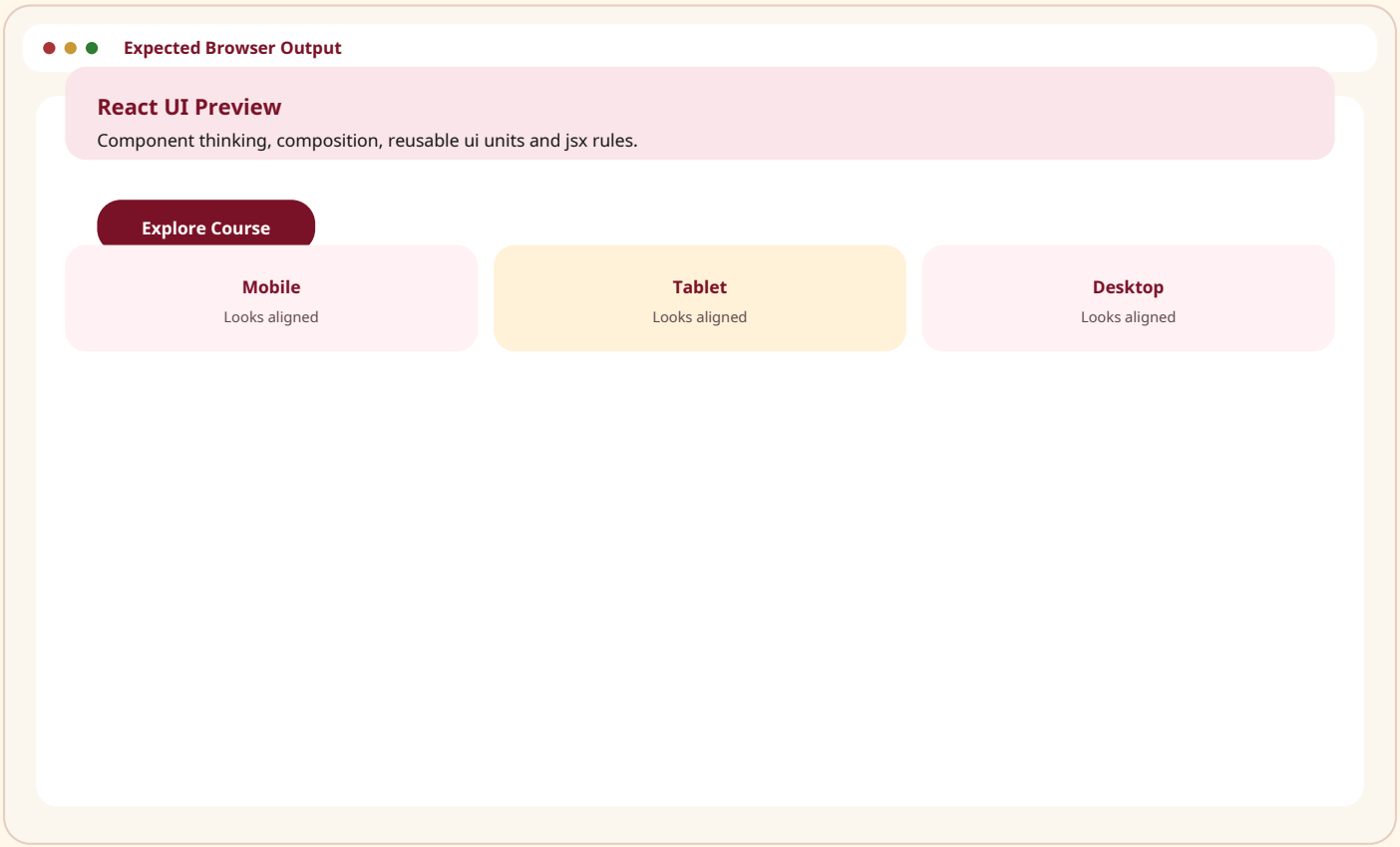
RUN AND UNDERSTAND

- Expected output: Reusable typed course card shows badge, title, progress and continue link.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: React Component Fundamentals

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: React Component Fundamentals

HANDS-ON TASK

- Build one screen section that demonstrates component thinking, composition, reusable UI units and JSX rules.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-29-react-component-fundamentals
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: React Component Fundamentals

WRITTEN QUESTIONS

- Q1. What problem does React Component Fundamentals solve in real UI engineering?
- Q2. Which file/language is most commonly involved with React JSX components props children?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where react component fundamentals is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: React Component Fundamentals

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: React Component Fundamentals

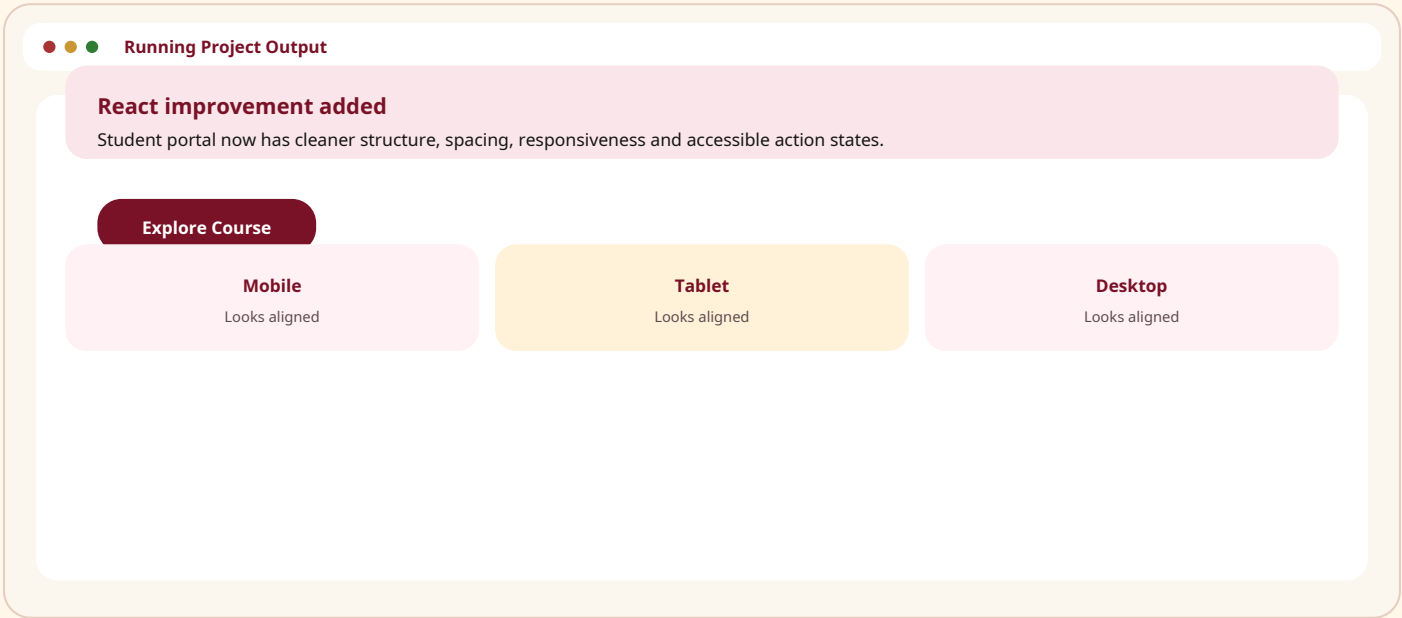
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using React JSX components props children.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-29.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 30: React State, Props and Data Flow

React | useState props derived state l | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn interactive UI, parent-child data flow and predictable state updates. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap interactive UI, parent-child data flow and predictable state updates ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

React State, Props and Data Flow: Detailed Notes 1

English: These notes explain the practical engineering decisions behind useState props derived state lifting state. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: React State, Props and Data Flow means applying useState props derived state lifting state to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where react state, props and data flow matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-30: add react improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

React State, Props and Data Flow: Detailed Notes 2

English: These notes explain the practical engineering decisions behind useState props derived state lifting state. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where react state, props and data flow matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-30: add react improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: React State, Props and Data Flow

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE components/CourseCard.tsx

```
type CourseCardProps = {
title: string;
level: 'Beginner' | 'Intermediate' | 'Advanced';
progress: number;
};

export function CourseCard({ title, level, progress }: CourseCardProps) {
return (
<article className="course-card" aria-label={`${title} course`}>
<span className="badge">{level}</span>
<h2>{title}</h2>
<progress value={progress} max={100} aria-label="Course progress" />
<a href="/student/courses/frontend-ui">Continue learning</a>
</article>
);
}
```

```
$ npm run dev
```

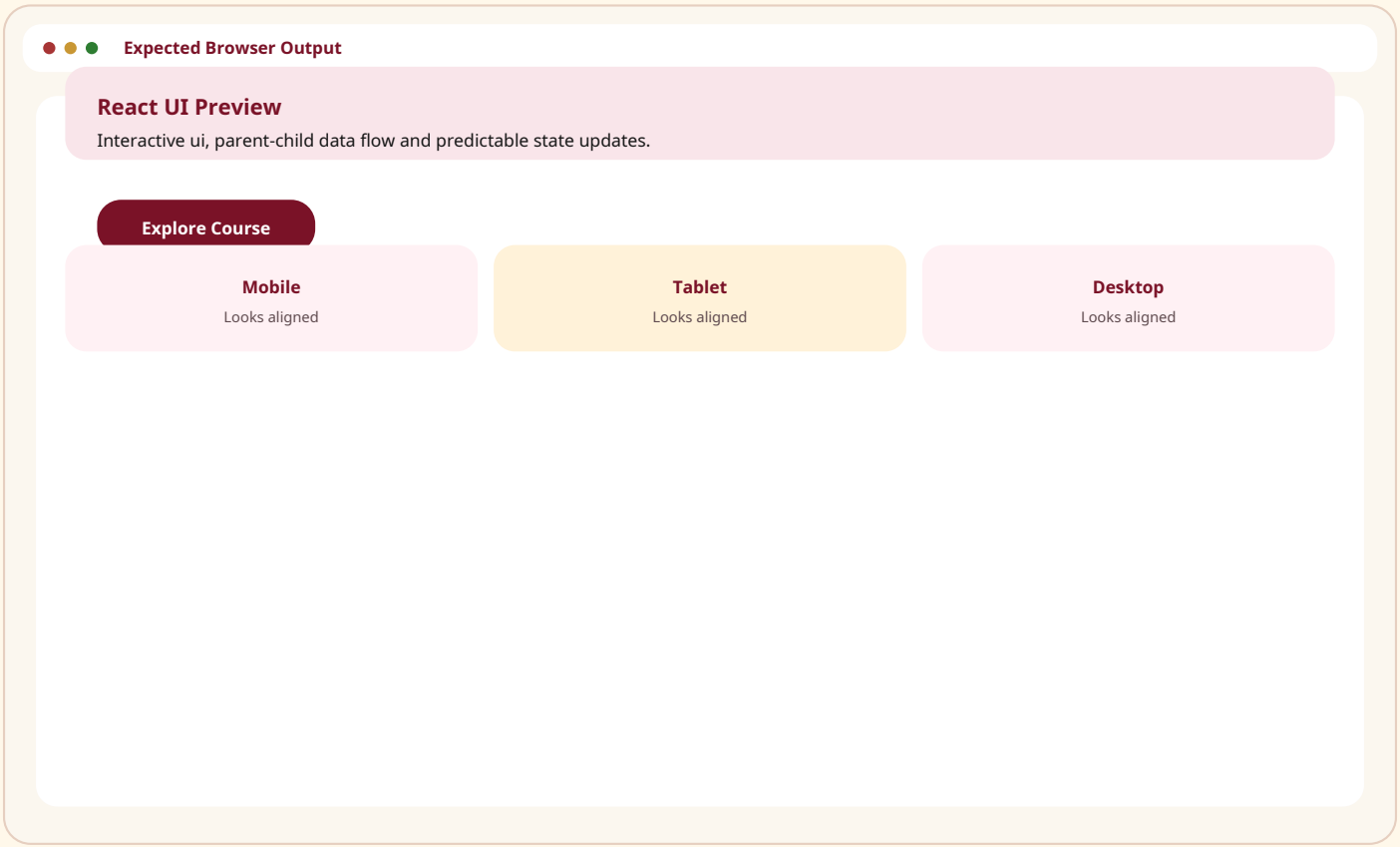
RUN AND UNDERSTAND

- Expected output: Reusable typed course card shows badge, title, progress and continue link.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: React State, Props and Data Flow

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: React State, Props and Data Flow

HANDS-ON TASK

- Build one screen section that demonstrates interactive UI, parent-child data flow and predictable state updates.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-30-react-state-props-and-data-flow
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: React State, Props and Data Flow

WRITTEN QUESTIONS

- Q1. What problem does React State, Props and Data Flow solve in real UI engineering?
- Q2. Which file/language is most commonly involved with useState props derived state lifting state?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where react state, props and data flow is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: React State, Props and Data Flow

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: React State, Props and Data Flow

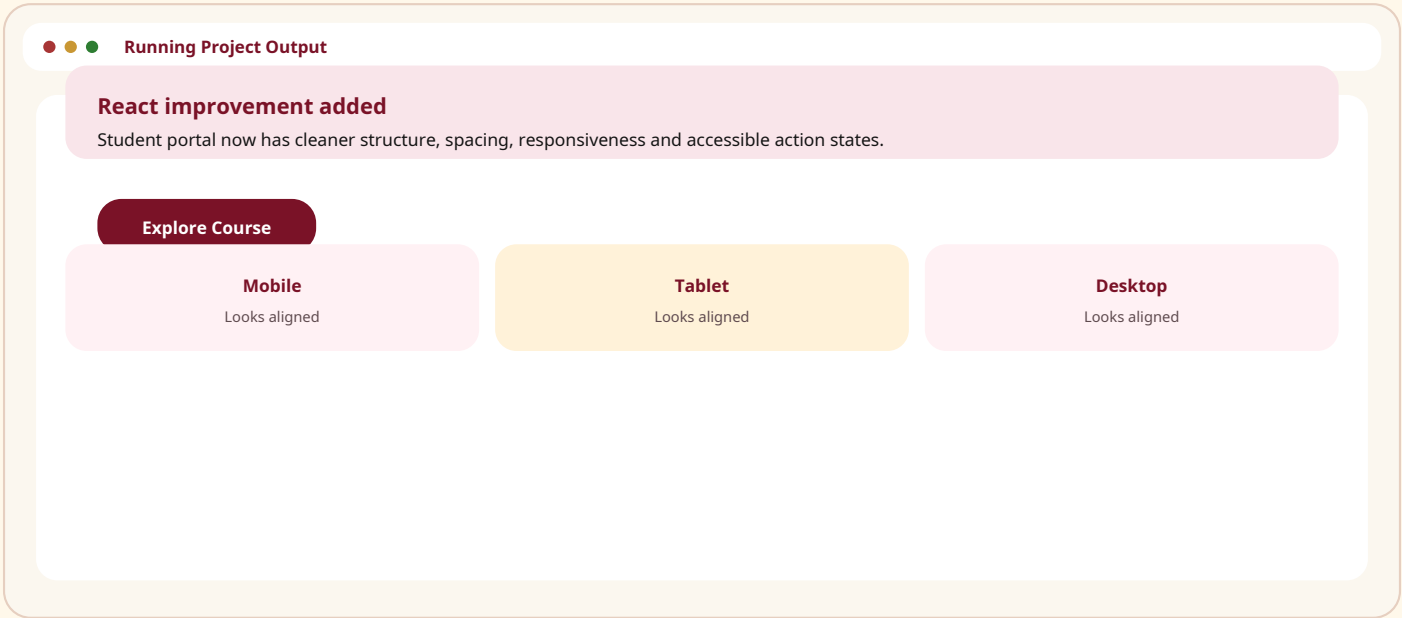
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using useState props derived state lifting state.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-30.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 31: React Hooks and Effects

React Advanced | useEffect useRef useMemo useCa | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn effects, refs, reusable hooks, performance and side-effect cleanup. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap effects, refs, reusable hooks, performance and side-effect cleanup ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

React Hooks and Effects: Detailed Notes 1

English: These notes explain the practical engineering decisions behind useEffect useRef useMemo useCallback custom hooks. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: React Hooks and Effects means applying useEffect useRef useMemo useCallback custom hooks to ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where react hooks and effects matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-31: add react advanced improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

React Hooks and Effects: Detailed Notes 2

English: These notes explain the practical engineering decisions behind useEffect useRef useMemo useCallback custom hooks. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where react hooks and effects matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-31: add react advanced improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: React Hooks and Effects

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE components/CourseCard.tsx

```
type CourseCardProps = {
title: string;
level: 'Beginner' | 'Intermediate' | 'Advanced';
progress: number;
};

export function CourseCard({ title, level, progress }: CourseCardProps) {
return (
<article className="course-card" aria-label={`${title} course`}>
<span className="badge">{level}</span>
<h2>{title}</h2>
<progress value={progress} max={100} aria-label="Course progress" />
<a href="/student/courses/frontend-ui">Continue learning</a>
</article>
);
}
```

```
$ npm run dev
```

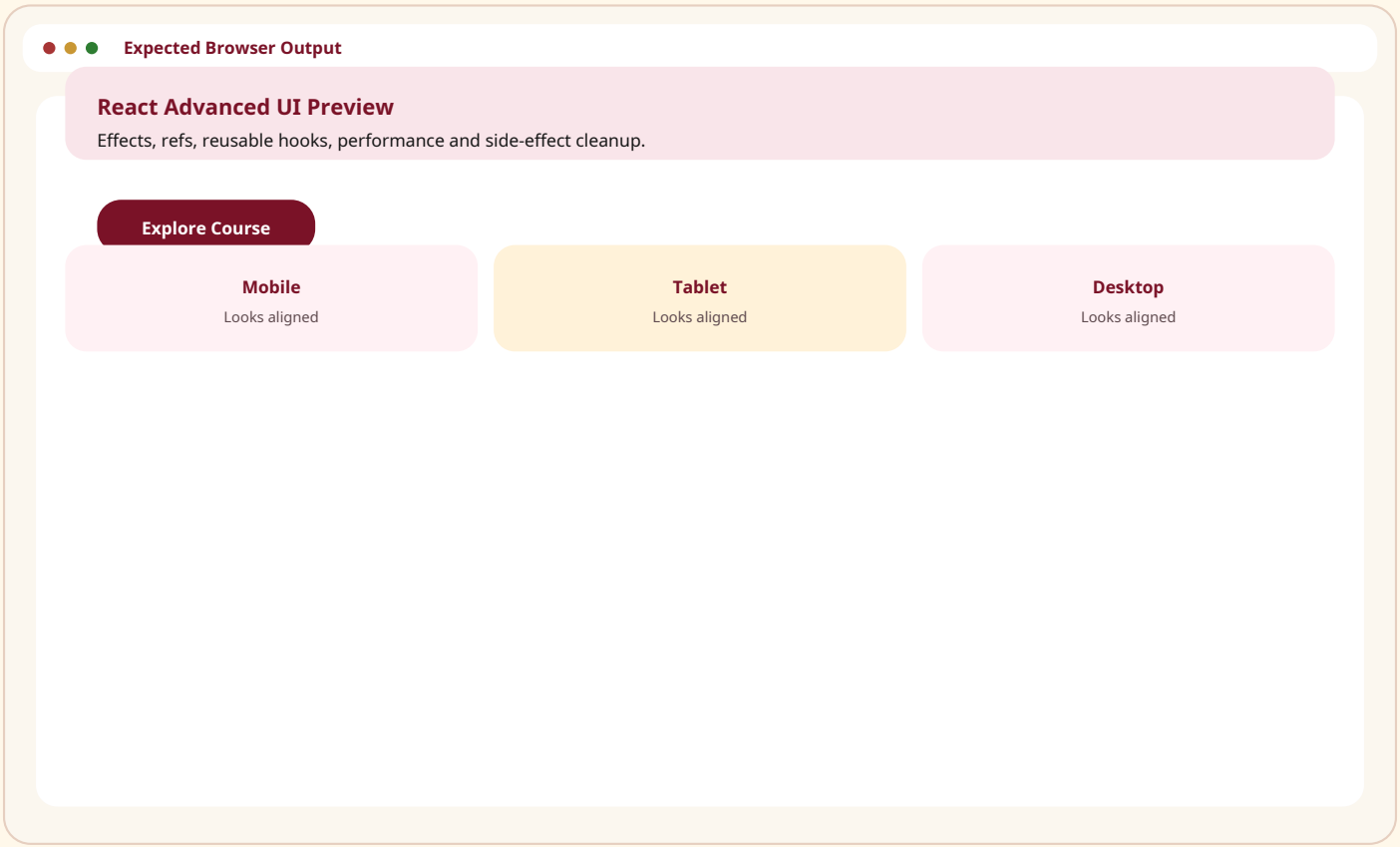
RUN AND UNDERSTAND

- Expected output: Reusable typed course card shows badge, title, progress and continue link.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: React Hooks and Effects

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: React Hooks and Effects

HANDS-ON TASK

- Build one screen section that demonstrates effects, refs, reusable hooks, performance and side-effect cleanup.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-31-react-hooks-and-effects
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: React Hooks and Effects

WRITTEN QUESTIONS

- Q1. What problem does React Hooks and Effects solve in real UI engineering?
- Q2. Which file/language is most commonly involved with useEffect useRef useMemo useCallback custom hooks?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where react hooks and effects is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: React Hooks and Effects

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: React Hooks and Effects

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using useEffect useRef useMemo useCallback custom hooks.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-31.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 32: React Forms and Component Libraries

React UI | controlled inputs validation s | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn forms, reusable inputs, component variants and production UI libraries. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap forms, reusable inputs, component variants and production UI libraries ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

React Forms and Component Libraries: Detailed Notes 1

English: These notes explain the practical engineering decisions behind controlled inputs validation shadcn MUI. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: React Forms and Component Libraries means applying controlled inputs validation shadcn MUI to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where react forms and component libraries matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-32: add react ui improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

React Forms and Component Libraries: Detailed Notes 2

English: These notes explain the practical engineering decisions behind controlled inputs validation shadcn MUI. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where react forms and component libraries matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-32: add react ui improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: React Forms and Component Libraries

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE components/CourseCard.tsx

```
type CourseCardProps = {
title: string;
level: 'Beginner' | 'Intermediate' | 'Advanced';
progress: number;
};

export function CourseCard({ title, level, progress }: CourseCardProps) {
return (
<article className="course-card" aria-label={`${title} course`}>
<span className="badge">{level}</span>
<h2>{title}</h2>
<progress value={progress} max={100} aria-label="Course progress" />
<a href="/student/courses/frontend-ui">Continue learning</a>
</article>
);
}
```

```
$ npm run dev
```

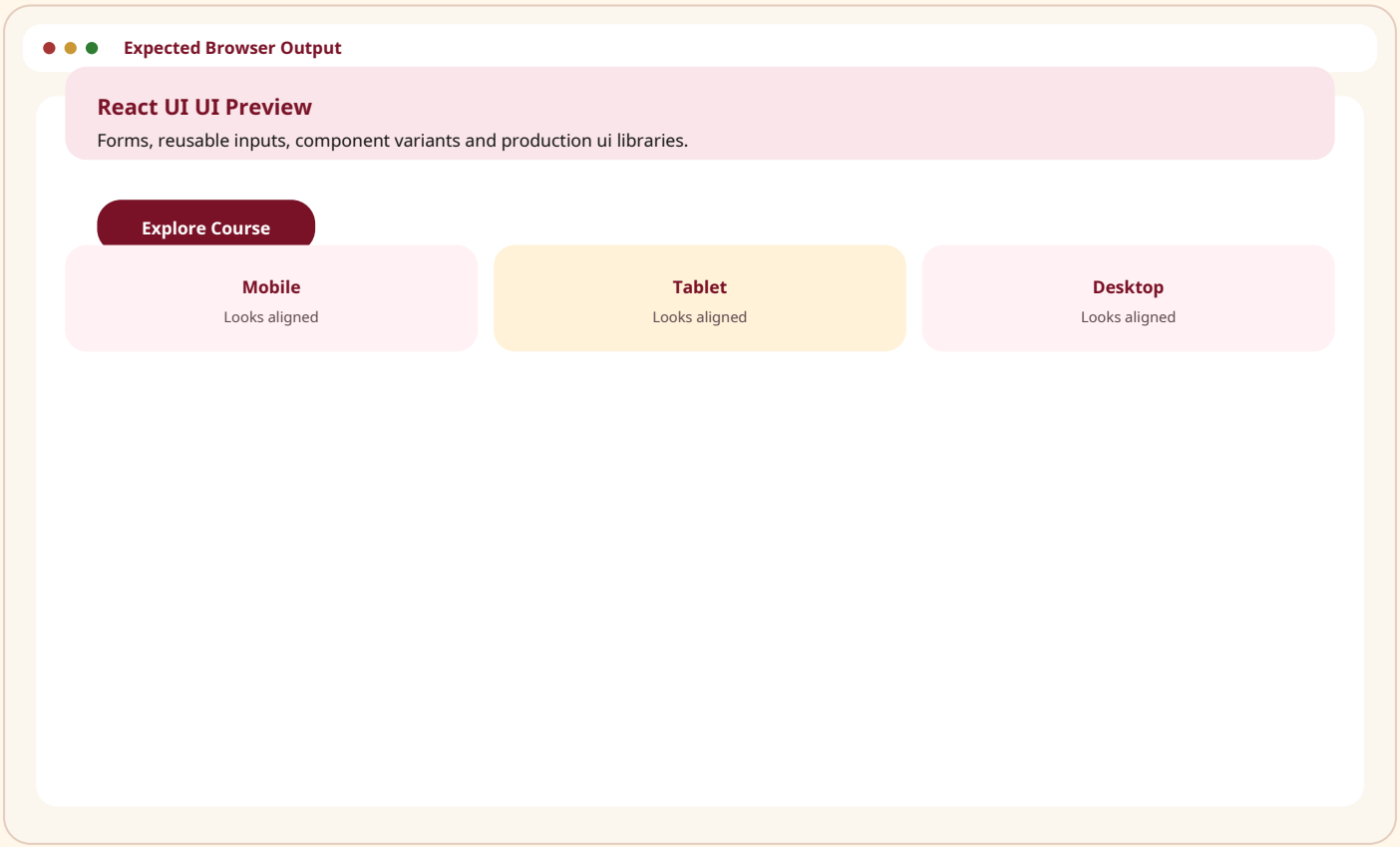
RUN AND UNDERSTAND

- Expected output: Reusable typed course card shows badge, title, progress and continue link.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: React Forms and Component Libraries

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: React Forms and Component Libraries

HANDS-ON TASK

- Build one screen section that demonstrates forms, reusable inputs, component variants and production UI libraries.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-32-react-forms-and-component-librarie
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: React Forms and Component Libraries

WRITTEN QUESTIONS

- Q1. What problem does React Forms and Component Libraries solve in real UI engineering?
- Q2. Which file/language is most commonly involved with controlled inputs validation shadcn MUI?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where react forms and component libraries is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: React Forms and Component Libraries

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: React Forms and Component Libraries

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using controlled inputs validation shadcn MUI.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-32.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

React UI improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 33: React Accessibility and Performance

React Quality | memo lazy Suspense aria keyboa | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn fast accessible React screens, reducing rerenders and code splitting. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap fast accessible React screens, reducing rerenders and code splitting ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

React Accessibility and Performance: Detailed Notes 1

English: These notes explain the practical engineering decisions behind memo lazy Suspense aria keyboard. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: React Accessibility and Performance means applying memo lazy Suspense aria keyboard to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where react accessibility and performance matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-33: add react quality improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

React Accessibility and Performance: Detailed Notes 2

English: These notes explain the practical engineering decisions behind memo lazy Suspense aria keyboard. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where react accessibility and performance matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-33: add react quality improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: React Accessibility and Performance

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE components/CourseCard.tsx

```
type CourseCardProps = {
title: string;
level: 'Beginner' | 'Intermediate' | 'Advanced';
progress: number;
};

export function CourseCard({ title, level, progress }: CourseCardProps) {
return (
<article className="course-card" aria-label={`${title} course`}>
<span className="badge">{level}</span>
<h2>{title}</h2>
<progress value={progress} max={100} aria-label="Course progress" />
<a href="/student/courses/frontend-ui">Continue learning</a>
</article>
);
}
```

```
$ npm run dev
```

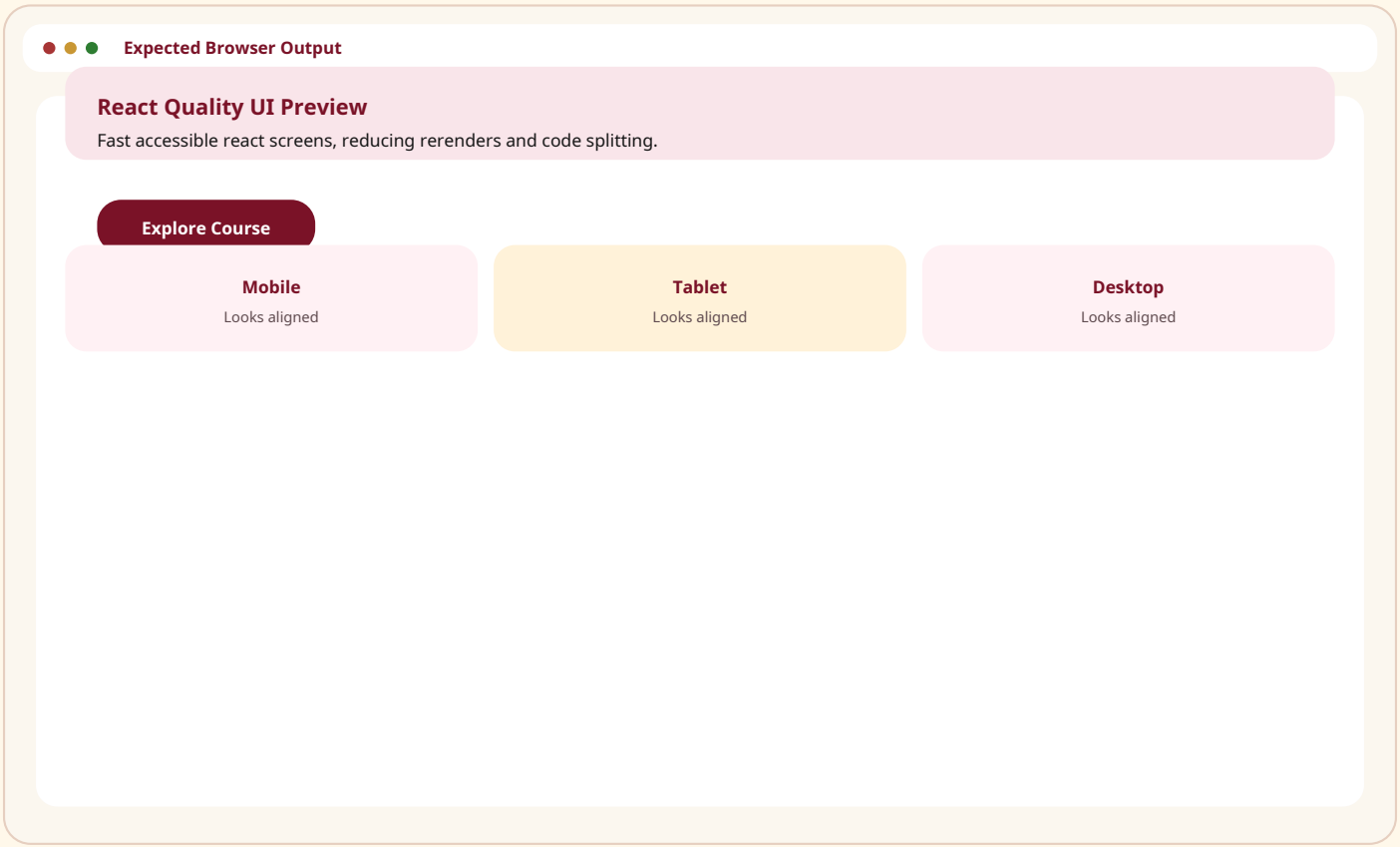
RUN AND UNDERSTAND

- Expected output: Reusable typed course card shows badge, title, progress and continue link.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: React Accessibility and Performance

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: React Accessibility and Performance

HANDS-ON TASK

- Build one screen section that demonstrates fast accessible React screens, reducing rerenders and code splitting.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-33-react-accessibility-and-performanc
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: React Accessibility and Performance

WRITTEN QUESTIONS

- Q1. What problem does React Accessibility and Performance solve in real UI engineering?
- Q2. Which file/language is most commonly involved with memo lazy Suspense aria keyboard?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where react accessibility and performance is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: React Accessibility and Performance

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: React Accessibility and Performance

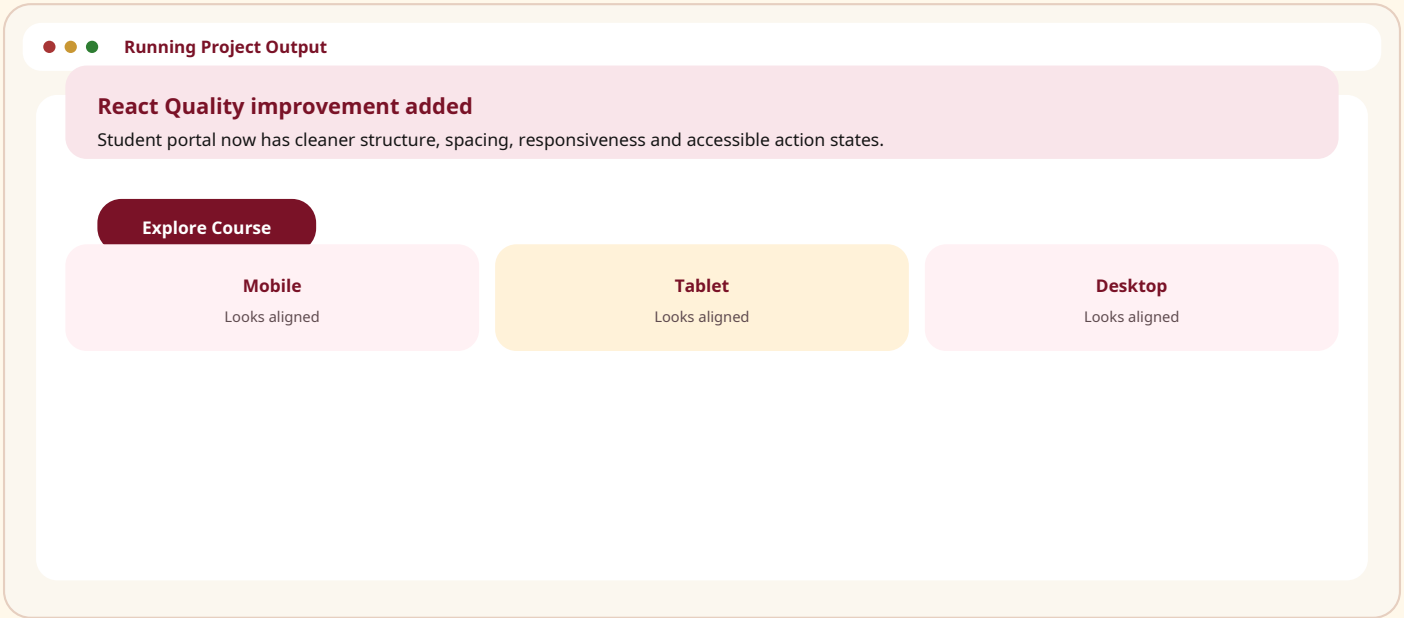
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using memo lazy Suspense aria keyboard.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-33.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 34: Next.js App Router UI Engineering

Next.js | App Router layouts pages Link | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn routing, layouts, server/client component decisions and image optimization. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap routing, layouts, server/client component decisions and image optimization ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Next.js App Router UI Engineering: Detailed Notes 1

English: These notes explain the practical engineering decisions behind App Router layouts pages Link Image metadata. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Next.js App Router UI Engineering means applying App Router layouts pages Link Image metadata to ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where next.js app router ui engineering matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-34: add next.js improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Next.js App Router UI Engineering: Detailed Notes 2

English: These notes explain the practical engineering decisions behind App Router layouts pages Link Image metadata. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where next.js app router ui engineering matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-34: add next.js improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Next.js App Router UI Engineering

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE components/CourseCard.tsx

```
type CourseCardProps = {
title: string;
level: 'Beginner' | 'Intermediate' | 'Advanced';
progress: number;
};

export function CourseCard({ title, level, progress }: CourseCardProps) {
return (
<article className="course-card" aria-label={`${title} course`}>
<span className="badge">{level}</span>
<h2>{title}</h2>
<progress value={progress} max={100} aria-label="Course progress" />
<a href="/student/courses/frontend-ui">Continue learning</a>
</article>
);
}
```

```
$ npm run dev
```

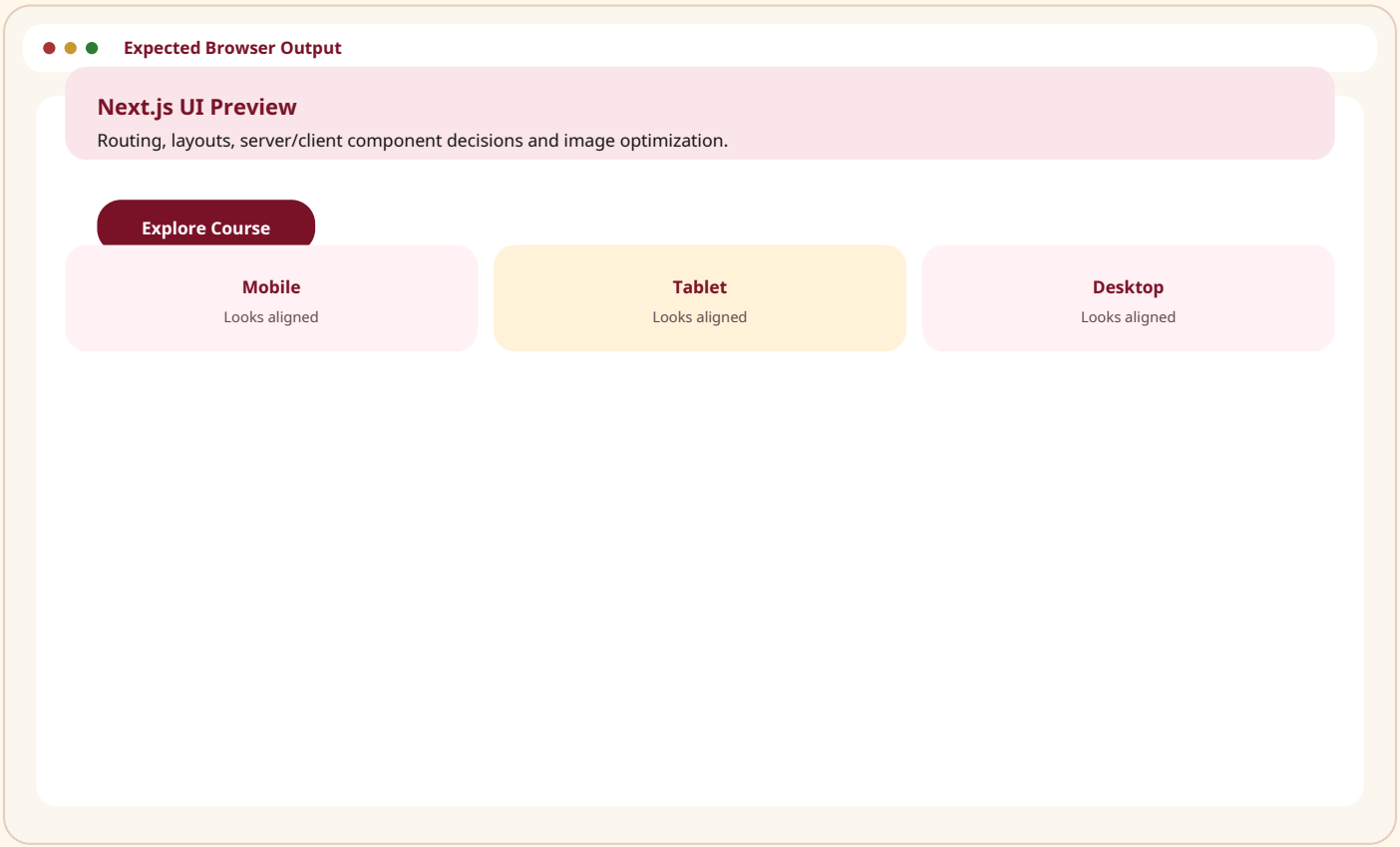
RUN AND UNDERSTAND

- Expected output: Reusable typed course card shows badge, title, progress and continue link.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Next.js App Router UI Engineering

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Next.js App Router UI Engineering

HANDS-ON TASK

- Build one screen section that demonstrates routing, layouts, server/client component decisions and image ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-34-next-js-app-router-ui-engineering
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Next.js App Router UI Engineering

WRITTEN QUESTIONS

- Q1. What problem does Next.js App Router UI Engineering solve in real UI engineering?
- Q2. Which file/language is most commonly involved with App Router layouts pages Link Image metadata?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where next.js app router ui engineering is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Next.js App Router UI Engineering

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Next.js App Router UI Engineering

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using App Router layouts pages Link Image metadata.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-34.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Next.js improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 35: API Integration and Auth UI

Integration | REST GraphQL token cookies ses | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn API helpers, login/register UI, protected screens and secure frontend patterns. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap API helpers, login/register UI, protected screens and secure frontend patterns ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

API Integration and Auth UI: Detailed Notes 1

English: These notes explain the practical engineering decisions behind REST GraphQL token cookies sessions. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: API Integration and Auth UI means applying REST GraphQL token cookies sessions to build reliable user ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where api integration and auth ui matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-35: add integration improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

API Integration and Auth UI: Detailed Notes 2

English: These notes explain the practical engineering decisions behind REST GraphQL token cookies sessions. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where api integration and auth ui matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-35: add integration improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: API Integration and Auth UI

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/api.ts

```
export async function getCourses() {
const res = await fetch('/api/courses');
if (!res.ok) throw new Error('Courses failed to load');
return res.json();
}

export async function renderCourses() {
const courses = await getCourses();
console.log('Loaded courses:', courses.length);
}
```

```
$ npm run dev
```

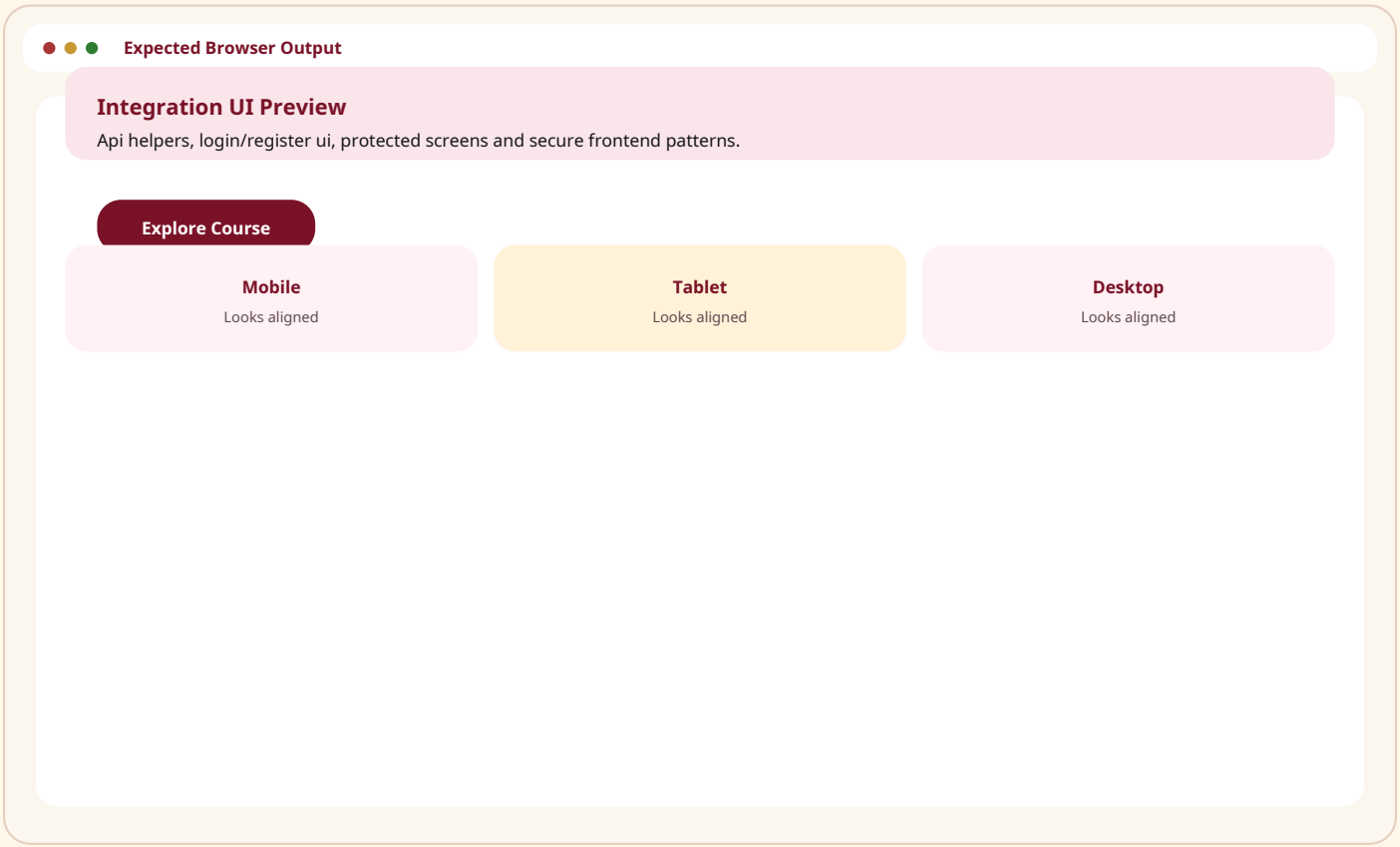
RUN AND UNDERSTAND

- Expected output: Course count logs after successful API call.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: API Integration and Auth UI

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: API Integration and Auth UI

HANDS-ON TASK

- Build one screen section that demonstrates API helpers, login/register UI, protected screens and secure frontend ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-35-api-integration-and-auth-ui
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: API Integration and Auth UI

WRITTEN QUESTIONS

- Q1. What problem does API Integration and Auth UI solve in real UI engineering?
- Q2. Which file/language is most commonly involved with REST GraphQL token cookies sessions?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where api integration and auth ui is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: API Integration and Auth UI

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: API Integration and Auth UI

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using REST GraphQL token cookies sessions.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-35.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Integration improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 36: State Management Patterns

State | Context Zustand Redux Query ca | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn global state, server state, cache invalidation and simple predictable stores. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap global state, server state, cache invalidation and simple predictable stores ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

State Management Patterns: Detailed Notes 1

English: These notes explain the practical engineering decisions behind Context Zustand Redux Query cache. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: State Management Patterns means applying Context Zustand Redux Query cache to build reliable user ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where state management patterns matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-36: add state improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

State Management Patterns: Detailed Notes 2

English: These notes explain the practical engineering decisions behind Context Zustand Redux Query cache. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where state management patterns matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-36: add state improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: State Management Patterns

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE index.html

```
<section class="feature-card">
<p class="eyebrow">UI Engineer Skill</p>
<h2>Build clean interfaces</h2>
<p>Use structure, spacing, contrast and interaction states.</p>
<a href="/learn" class="btn">Learn More</a>
</section>
```

```
$ Open index.html
```

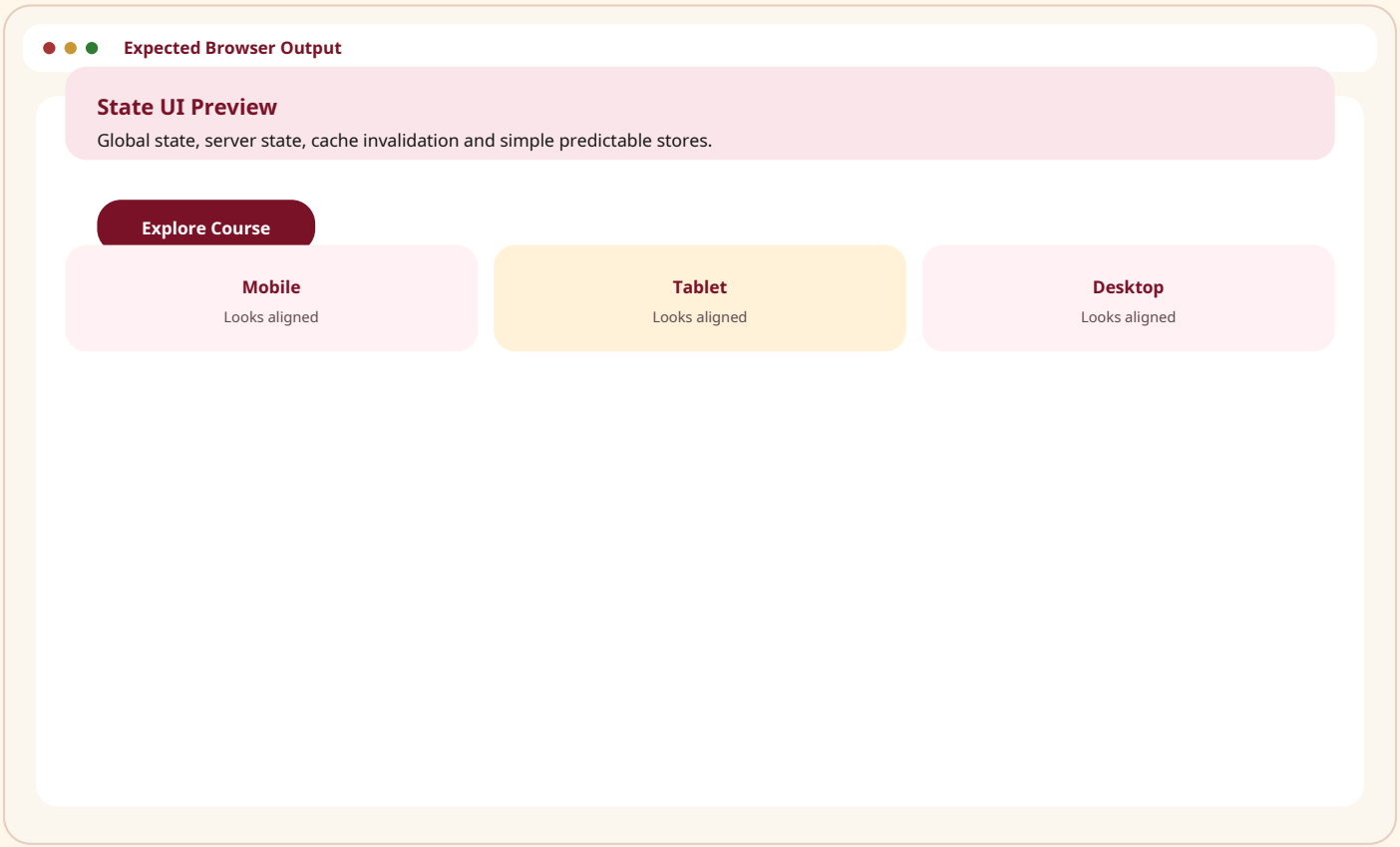
RUN AND UNDERSTAND

- Expected output: Feature card appears with heading, paragraph and link button.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: State Management Patterns

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: State Management Patterns

HANDS-ON TASK

- Build one screen section that demonstrates global state, server state, cache invalidation and simple predictable ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-36-state-management-patterns
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: State Management Patterns

WRITTEN QUESTIONS

- Q1. What problem does State Management Patterns solve in real UI engineering?
- Q2. Which file/language is most commonly involved with Context Zustand Redux Query cache?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where state management patterns is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: State Management Patterns

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: State Management Patterns

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using Context Zustand Redux Query cache.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-36.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

State improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 37: Routing, Navigation and Protected UI

Navigation | React Router Next Link guards | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn routes, active links, breadcrumbs, protected pages and loading navigation. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap routes, active links, breadcrumbs, protected pages and loading navigation ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Routing, Navigation and Protected UI: Detailed Notes 1

English: These notes explain the practical engineering decisions behind React Router Next Link guards breadcrumbs. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Routing, Navigation and Protected UI means applying React Router Next Link guards breadcrumbs to ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where routing, navigation and protected ui matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-37: add navigation improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Routing, Navigation and Protected UI: Detailed Notes 2

English: These notes explain the practical engineering decisions behind React Router Next Link guards breadcrumbs. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where routing, navigation and protected ui matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-37: add navigation improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Routing, Navigation and Protected UI

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE style.css

```
:root {
--brand: #7a1227;
--cream: #fff8ea;
--text: #151515;
}
.card {
background: var(--cream);
color: var(--text);
border: 1px solid #e6cfc0;
border-radius: 24px;
padding: clamp(1rem, 2vw, 2rem);
}
```

```
$ Open page and inspect styles
```

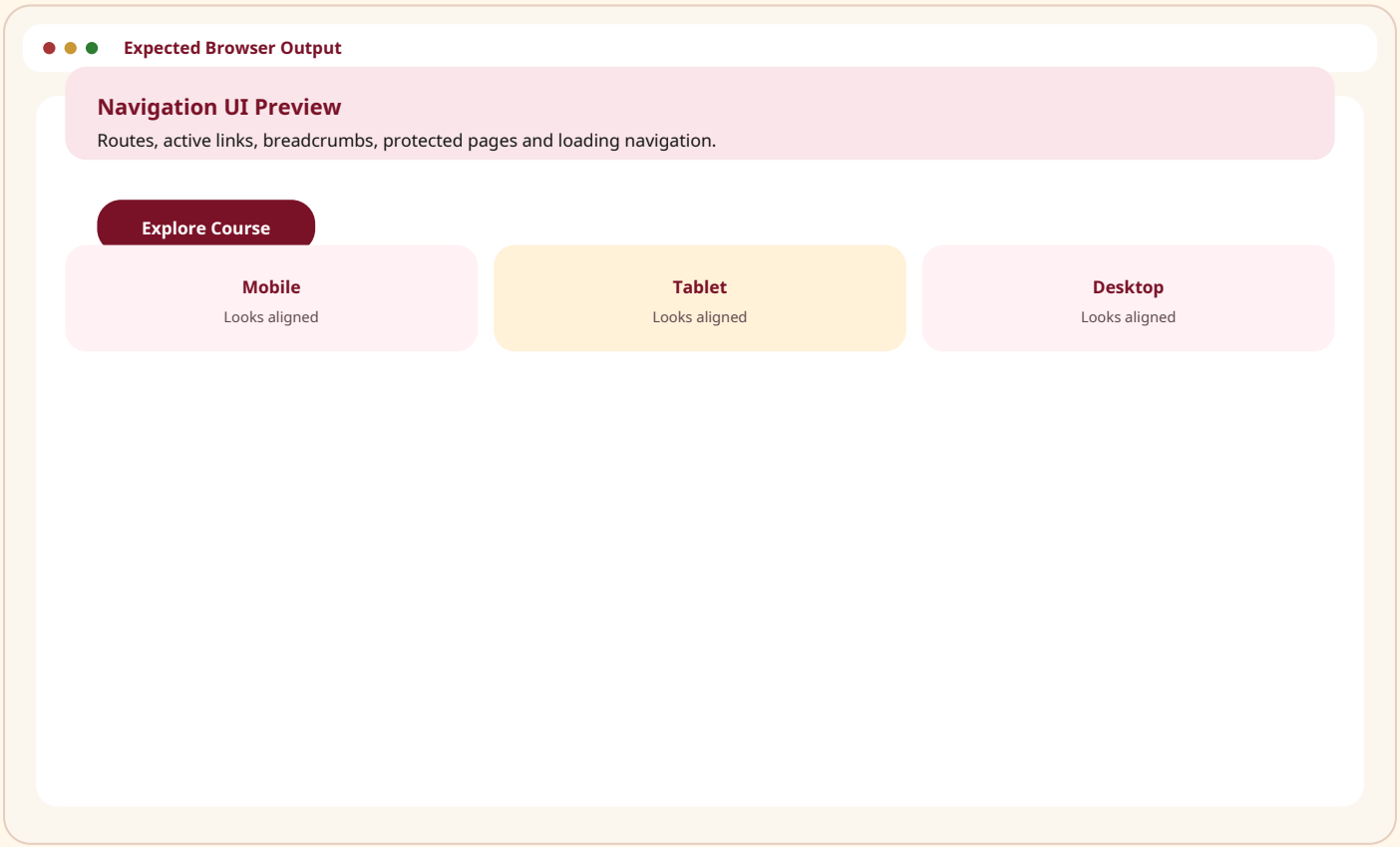
RUN AND UNDERSTAND

- Expected output: Card uses reusable tokens and scales spacing across devices.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Routing, Navigation and Protected UI

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Routing, Navigation and Protected UI

HANDS-ON TASK

- Build one screen section that demonstrates routes, active links, breadcrumbs, protected pages and loading ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-37-routing-navigation-and-protected-u
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Routing, Navigation and Protected UI

WRITTEN QUESTIONS

- Q1. What problem does Routing, Navigation and Protected UI solve in real UI engineering?
- Q2. Which file/language is most commonly involved with React Router Next Link guards breadcrumbs?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where routing, navigation and protected ui is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Routing, Navigation and Protected UI

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Routing, Navigation and Protected UI

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using React Router Next Link guards breadcrumbs.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-37.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 38: Animation and Micro-Interactions

Motion UI | CSS transitions keyframes Fram | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn smooth interactions, reduced motion, hover states and professional feedback. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap smooth interactions, reduced motion, hover states and professional feedback ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Animation and Micro-Interactions: Detailed Notes 1

English: These notes explain the practical engineering decisions behind CSS transitions keyframes Framer Motion. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Animation and Micro-Interactions means applying CSS transitions keyframes Framer Motion to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where animation and micro-interactions matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-38: add motion ui improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Animation and Micro-Interactions: Detailed Notes 2

English: These notes explain the practical engineering decisions behind CSS transitions keyframes Framer Motion. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where animation and micro-interactions matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-38: add motion ui improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Animation and Micro-Interactions

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/main.js

```
const tabs = document.querySelectorAll('[data-tab]');
const panels = document.querySelectorAll('[data-panel]');

tabs.forEach((tab) => {
tab.addEventListener('click', () => {
panels.forEach((panel) => panel.hidden = panel.id !== tab.dataset.tab);
});
});
```

```
$ Open tabs.html and click tabs
```

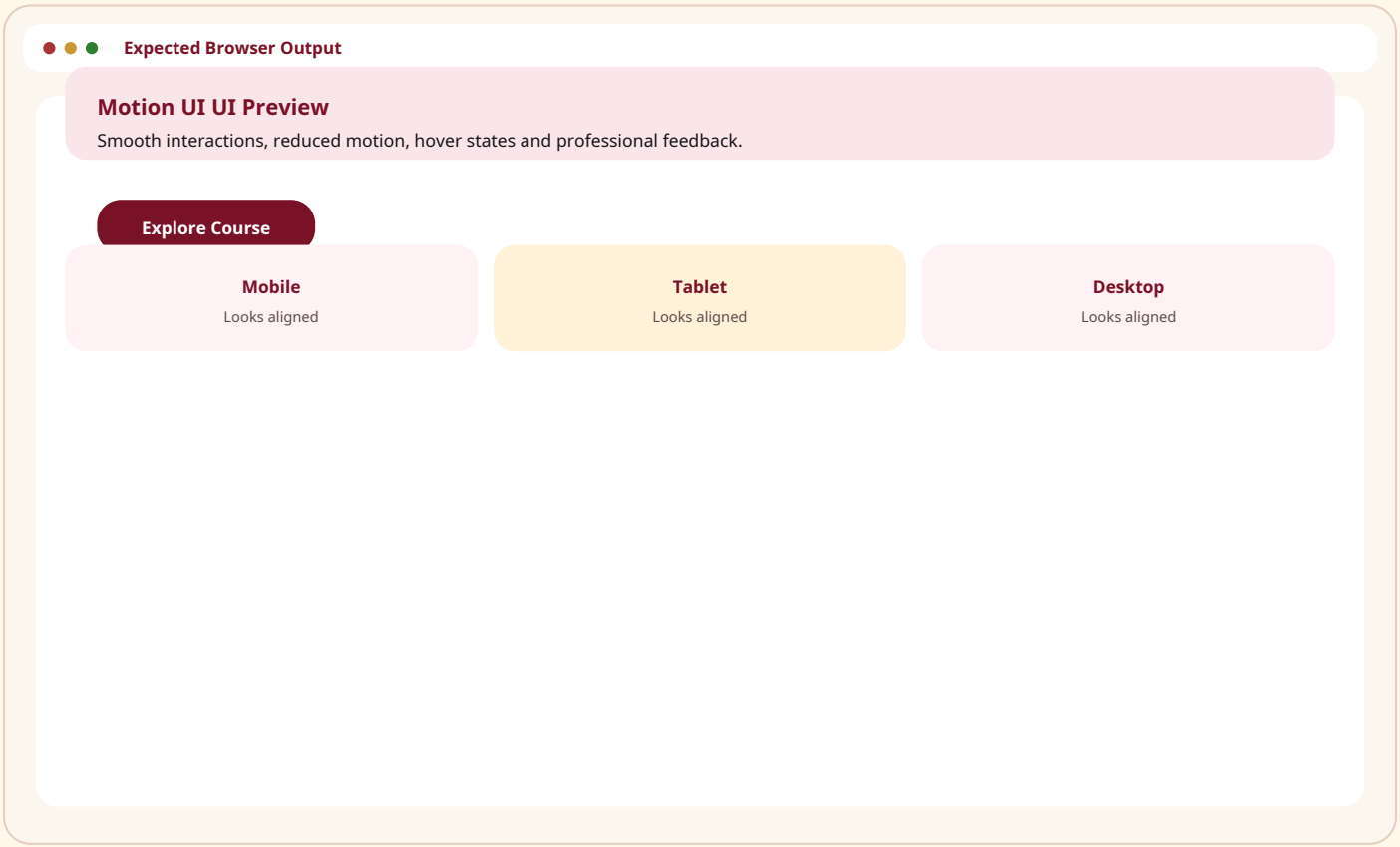
RUN AND UNDERSTAND

- Expected output: Only the selected tab panel remains visible.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Animation and Micro-Interactions

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Animation and Micro-Interactions

HANDS-ON TASK

- Build one screen section that demonstrates smooth interactions, reduced motion, hover states and professional ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-38-animation-and-micro-interactions
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Animation and Micro-Interactions

WRITTEN QUESTIONS

- Q1. What problem does Animation and Micro-Interactions solve in real UI engineering?
- Q2. Which file/language is most commonly involved with CSS transitions keyframes Framer Motion?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where animation and micro-interactions is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Animation and Micro-Interactions

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Animation and Micro-Interactions

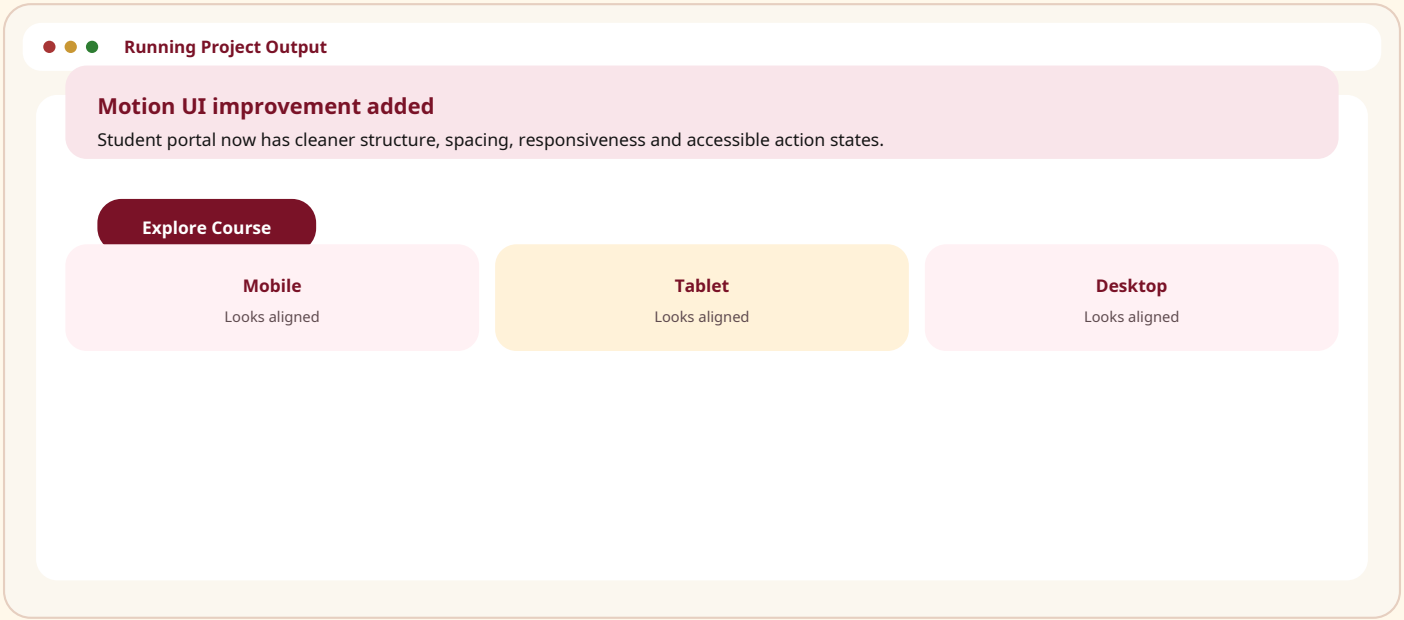
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using CSS transitions keyframes Framer Motion.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-38.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 39: SVG, Icons, Canvas and Visual Assets

Visuals | SVG icon sprites canvas lottie | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn crisp icons, scalable illustrations, asset optimization and visual polish. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap crisp icons, scalable illustrations, asset optimization and visual polish ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

SVG, Icons, Canvas and Visual Assets: Detailed Notes 1

English: These notes explain the practical engineering decisions behind SVG icon sprites canvas lottie images. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: SVG, Icons, Canvas and Visual Assets means applying SVG icon sprites canvas lottie images to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where svg, icons, canvas and visual assets matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-39: add visuals improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

SVG, Icons, Canvas and Visual Assets: Detailed Notes 2

English: These notes explain the practical engineering decisions behind SVG icon sprites canvas lottie images. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where svg, icons, canvas and visual assets matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-39: add visuals improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: SVG, Icons, Canvas and Visual Assets

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/api.ts

```
export async function getCourses() {
const res = await fetch('/api/courses');
if (!res.ok) throw new Error('Courses failed to load');
return res.json();
}

export async function renderCourses() {
const courses = await getCourses();
console.log('Loaded courses:', courses.length);
}
```

```
$ npm run dev
```

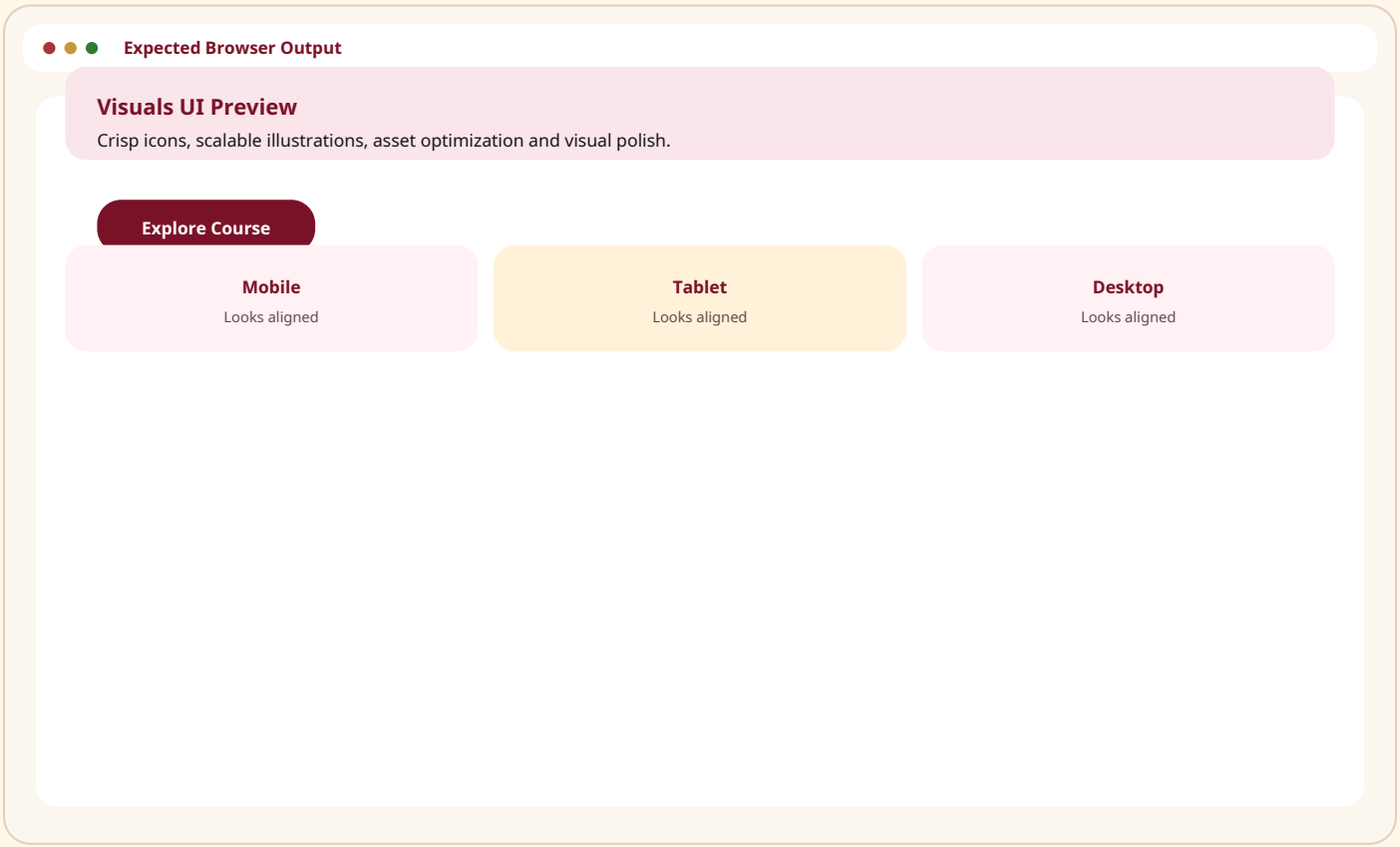
RUN AND UNDERSTAND

- Expected output: Course count logs after successful API call.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: SVG, Icons, Canvas and Visual Assets

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: SVG, Icons, Canvas and Visual Assets

HANDS-ON TASK

- Build one screen section that demonstrates crisp icons, scalable illustrations, asset optimization and visual polish.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-39-svg-icons-canvas-and-visual-assets
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: SVG, Icons, Canvas and Visual Assets

WRITTEN QUESTIONS

- Q1. What problem does SVG, Icons, Canvas and Visual Assets solve in real UI engineering?
- Q2. Which file/language is most commonly involved with SVG icon sprites canvas lottie images?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where svg, icons, canvas and visual assets is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: SVG, Icons, Canvas and Visual Assets

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: SVG, Icons, Canvas and Visual Assets

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using SVG icon sprites canvas lottie images.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-39.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Visuals improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 40: Charts and Data Visualization UI

Data Viz | SVG Recharts Chart.js tables d | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn dashboard charts, legends, labels, empty states and accessible data display. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap dashboard charts, legends, labels, empty states and accessible data display ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Charts and Data Visualization UI: Detailed Notes 1

English: These notes explain the practical engineering decisions behind SVG Recharts Chart.js tables dashboards. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Charts and Data Visualization UI means applying SVG Recharts Chart.js tables dashboards to build reliable ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where charts and data visualization ui matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-40: add data viz improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Charts and Data Visualization UI: Detailed Notes 2

English: These notes explain the practical engineering decisions behind SVG Recharts Chart.js tables dashboards. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where charts and data visualization ui matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-40: add data viz improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Charts and Data Visualization UI

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE index.html

```
<section class="feature-card">
<p class="eyebrow">UI Engineer Skill</p>
<h2>Build clean interfaces</h2>
<p>Use structure, spacing, contrast and interaction states.</p>
<a href="/learn" class="btn">Learn More</a>
</section>
```

```
$ Open index.html
```

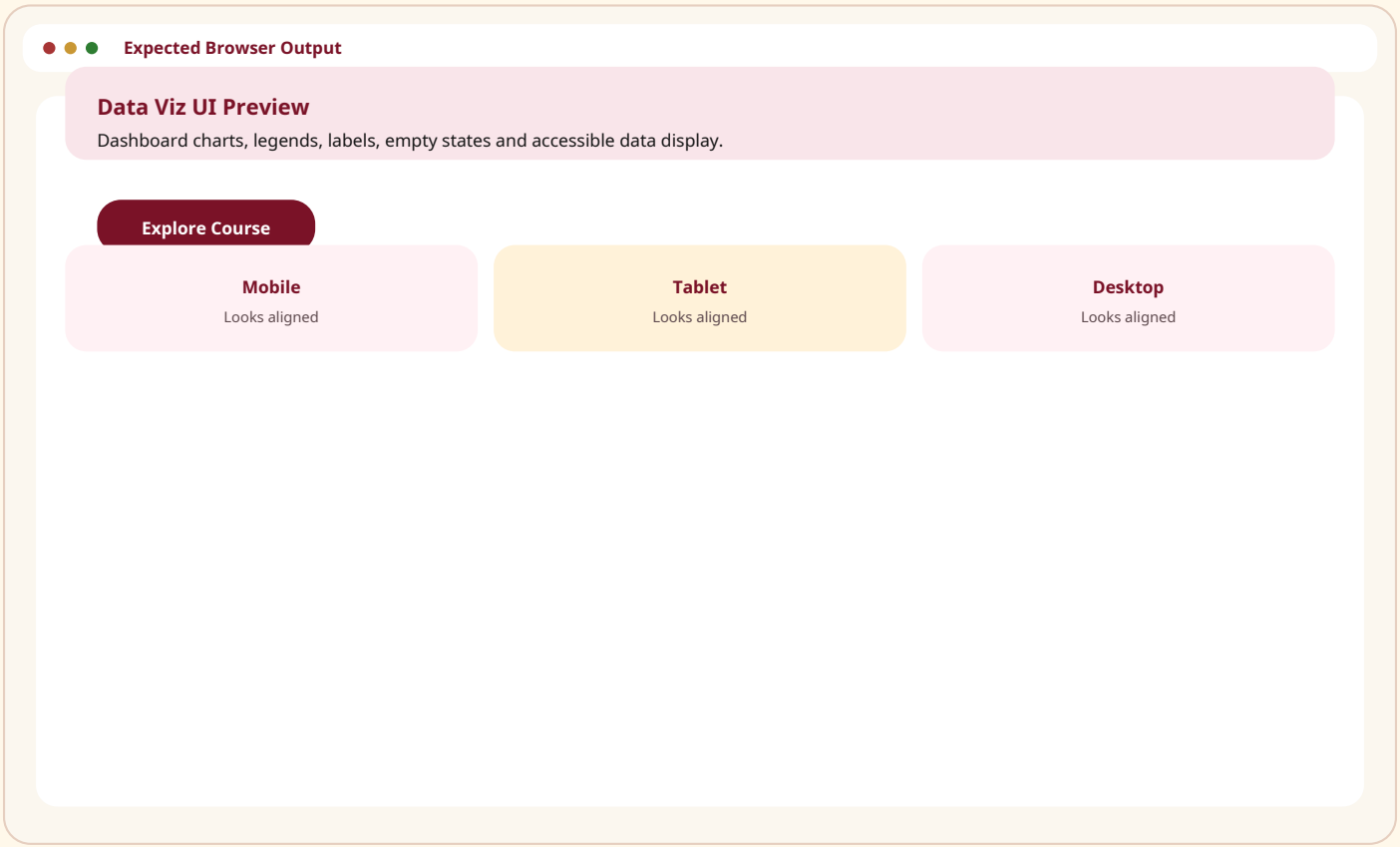
RUN AND UNDERSTAND

- Expected output: Feature card appears with heading, paragraph and link button.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Charts and Data Visualization UI

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Charts and Data Visualization UI

HANDS-ON TASK

- Build one screen section that demonstrates dashboard charts, legends, labels, empty states and accessible data ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-40-charts-and-data-visualization-ui
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Charts and Data Visualization UI

WRITTEN QUESTIONS

- Q1. What problem does Charts and Data Visualization UI solve in real UI engineering?
- Q2. Which file/language is most commonly involved with SVG Recharts Chart.js tables dashboards?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where charts and data visualization ui is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Charts and Data Visualization UI

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Charts and Data Visualization UI

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using SVG Recharts Chart.js tables dashboards.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-40.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Data Viz improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 41: Unit and Component Testing

Testing | Vitest Jest Testing Library | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn testing components, user events, assertions, mocks and safe refactoring. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap testing components, user events, assertions, mocks and safe refactoring ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Unit and Component Testing: Detailed Notes 1

English: These notes explain the practical engineering decisions behind Vitest Jest Testing Library. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Unit and Component Testing means applying Vitest Jest Testing Library to build reliable user interfaces.
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where unit and component testing matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-41: add testing improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Unit and Component Testing: Detailed Notes 2

English: These notes explain the practical engineering decisions behind Vitest Jest Testing Library. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where unit and component testing matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-41: add testing improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Unit and Component Testing

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE tests/course-card.spec.ts

```
import { test, expect } from '@playwright/test';

test('student can open frontend UI course', async ({ page }) => {
await page.goto('/student/courses');
await page.getByRole('link', { name: /frontend ui/i }).click();
await expect(page.getByRole('heading', { name: /Frontend UI Engineer/i })).toBeVisible();
await expect(page.getByRole('button', { name: /Start Learning/i })).toBeEnabled();
});
```

```
$ npx playwright test
```

RUN AND UNDERSTAND

- Expected output: The browser opens course page and verifies heading plus start button.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Unit and Component Testing

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.

Expected Browser Output

```
$ npm run dev
> frontend-ui@1.0.0 dev
Local: http://localhost:5173/
Status: compiled successfully
Module: Testing
No console errors found
Screenshot saved: output/module.png
```

VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Unit and Component Testing

HANDS-ON TASK

- Build one screen section that demonstrates testing components, user events, assertions, mocks and safe ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-41-unit-and-component-testing
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Unit and Component Testing

WRITTEN QUESTIONS

- Q1. What problem does Unit and Component Testing solve in real UI engineering?
- Q2. Which file/language is most commonly involved with Vitest Jest Testing Library?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where unit and component testing is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Unit and Component Testing

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Unit and Component Testing

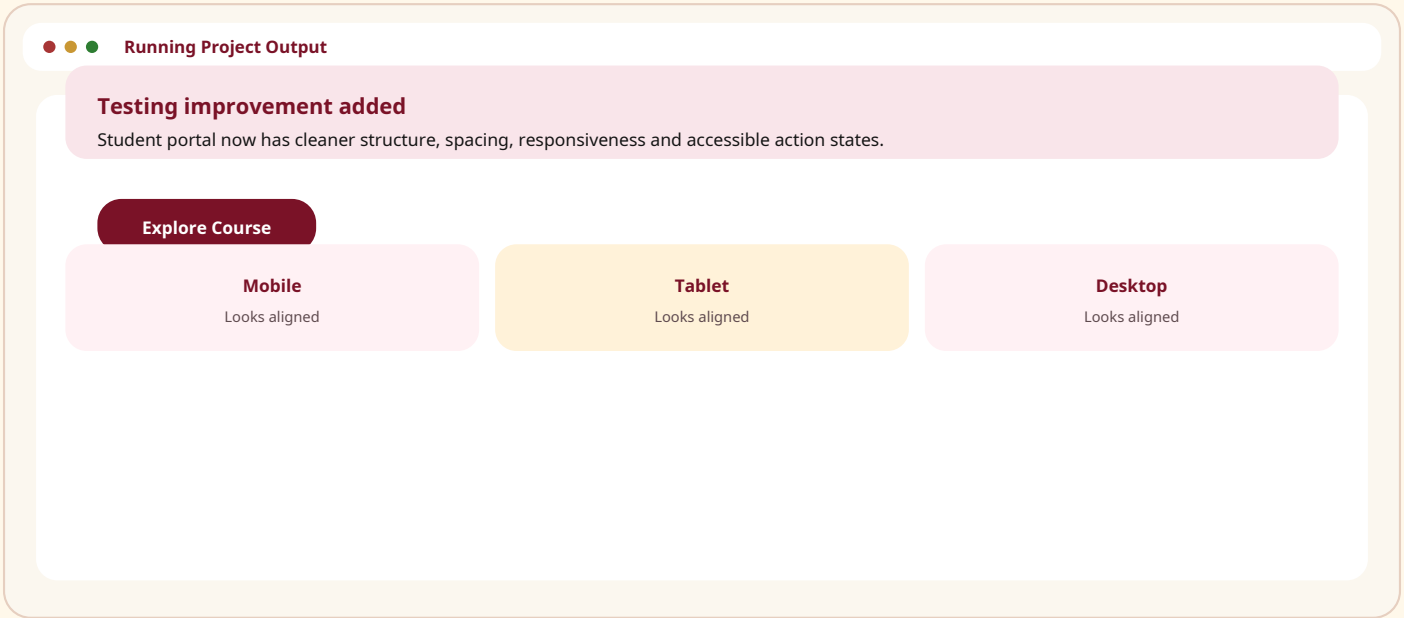
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using Vitest Jest Testing Library.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-41.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 42: End-to-End Testing with Playwright

Testing E2E | Playwright Cypress locators tr | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn browser flows, login tests, form tests, visual stability and CI checks. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap browser flows, login tests, form tests, visual stability and CI checks ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

End-to-End Testing with Playwright: Detailed Notes 1

English: These notes explain the practical engineering decisions behind Playwright Cypress locators traces. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: End-to-End Testing with Playwright means applying Playwright Cypress locators traces to build reliable ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where end-to-end testing with playwright matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-42: add testing e2e improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

End-to-End Testing with Playwright: Detailed Notes 2

English: These notes explain the practical engineering decisions behind Playwright Cypress locators traces. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where end-to-end testing with playwright matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-42: add testing e2e improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: End-to-End Testing with Playwright

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE tests/course-card.spec.ts

```
import { test, expect } from '@playwright/test';

test('student can open frontend UI course', async ({ page }) => {
await page.goto('/student/courses');
await page.getByRole('link', { name: /frontend ui/i }).click();
await expect(page.getByRole('heading', { name: /Frontend UI Engineer/i })).toBeVisible();
await expect(page.getByRole('button', { name: /Start Learning/i })).toBeEnabled();
});
```

```
$ npx playwright test
```

RUN AND UNDERSTAND

- Expected output: The browser opens course page and verifies heading plus start button.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: End-to-End Testing with Playwright

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.

Expected Browser Output

```
$ npm run dev
> frontend-ui@1.0.0 dev
Local: http://localhost:5173/
Status: compiled successfully
Module: Testing E2E
No console errors found
Screenshot saved: output/module.png
```

VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: End-to-End Testing with Playwright

HANDS-ON TASK

- Build one screen section that demonstrates browser flows, login tests, form tests, visual stability and CI checks.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-42-end-to-end-testing-with-playwright
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: End-to-End Testing with Playwright

WRITTEN QUESTIONS

- Q1. What problem does End-to-End Testing with Playwright solve in real UI engineering?
- Q2. Which file/language is most commonly involved with Playwright Cypress locators traces?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where end-to-end testing with playwright is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: End-to-End Testing with Playwright

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: End-to-End Testing with Playwright

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using Playwright Cypress locators traces.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-42.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Testing E2E improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 43: Storybook and UI Documentation

Docs | Storybook stories controls MDX | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn component documentation, states, props, design review and team collaboration. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap component documentation, states, props, design review and team collaboration ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Storybook and UI Documentation: Detailed Notes 1

English: These notes explain the practical engineering decisions behind Storybook stories controls MDX Chromatic. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Storybook and UI Documentation means applying Storybook stories controls MDX Chromatic to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where storybook and ui documentation matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-43: add docs improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Storybook and UI Documentation: Detailed Notes 2

English: These notes explain the practical engineering decisions behind Storybook stories controls MDX Chromatic. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where storybook and ui documentation matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-43: add docs improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Storybook and UI Documentation

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/api.ts

```
export async function getCourses() {
const res = await fetch('/api/courses');
if (!res.ok) throw new Error('Courses failed to load');
return res.json();
}

export async function renderCourses() {
const courses = await getCourses();
console.log('Loaded courses:', courses.length);
}
```

```
$ npm run dev
```

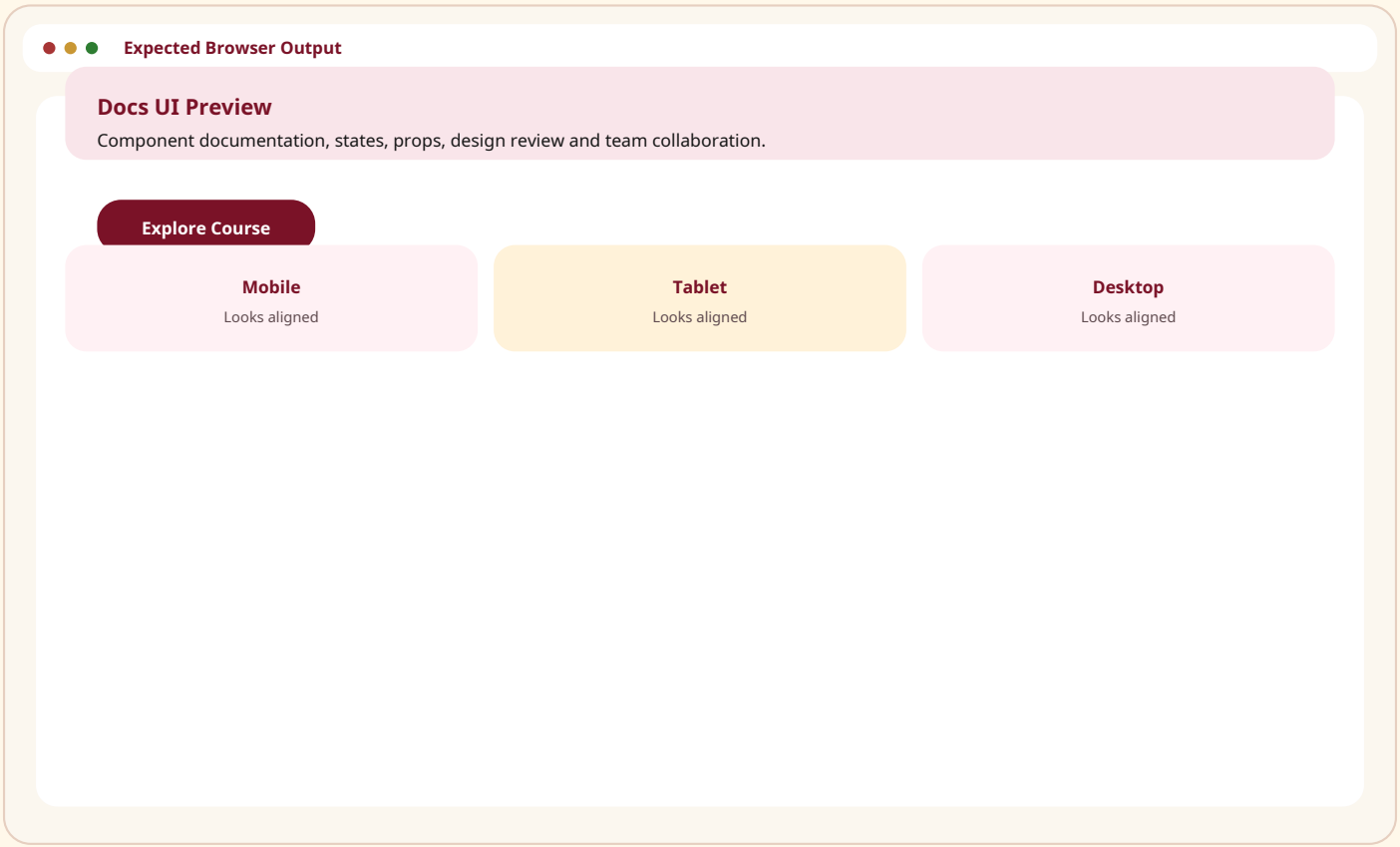
RUN AND UNDERSTAND

- Expected output: Course count logs after successful API call.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Storybook and UI Documentation

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Storybook and UI Documentation

HANDS-ON TASK

- Build one screen section that demonstrates component documentation, states, props, design review and team ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-43-storybook-and-ui-documentation
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Storybook and UI Documentation

WRITTEN QUESTIONS

- Q1. What problem does Storybook and UI Documentation solve in real UI engineering?
- Q2. Which file/language is most commonly involved with Storybook stories controls MDX Chromatic?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where storybook and ui documentation is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Storybook and UI Documentation

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Storybook and UI Documentation

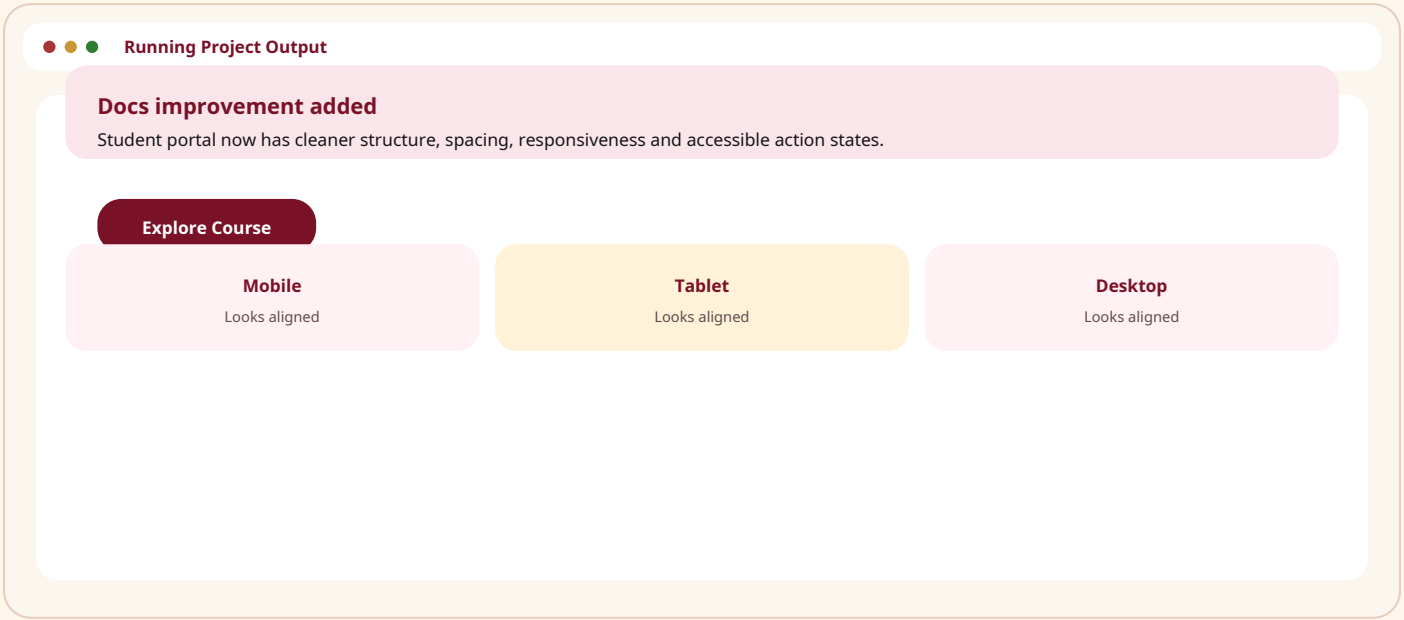
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using Storybook stories controls MDX Chromatic.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-43.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 44: Web Performance and Core Web Vitals

Performance | Lighthouse lazy loading code s | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn fast loading, low layout shift, optimized assets and performance budgets. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap fast loading, low layout shift, optimized assets and performance budgets ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Web Performance and Core Web Vitals: Detailed Notes 1

English: These notes explain the practical engineering decisions behind Lighthouse lazy loading code split cache. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Web Performance and Core Web Vitals means applying Lighthouse lazy loading code split cache to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where web performance and core web vitals matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-44: add performance improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Web Performance and Core Web Vitals: Detailed Notes 2

English: These notes explain the practical engineering decisions behind Lighthouse lazy loading code split cache. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where web performance and core web vitals matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-44: add performance improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Web Performance and Core Web Vitals

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE components/OptimizedHero.tsx

```
import Image from 'next/image';

export function OptimizedHero() {
return (
<section className="hero">
<Image
src="/html/frontend-ui.png"
alt="Student learning frontend UI engineering"
width={920}
height={520}
priority
/>
<h1>Fast, accessible UI</h1>
</section>
);
}
```

```
$ npm run build && npm run start
```

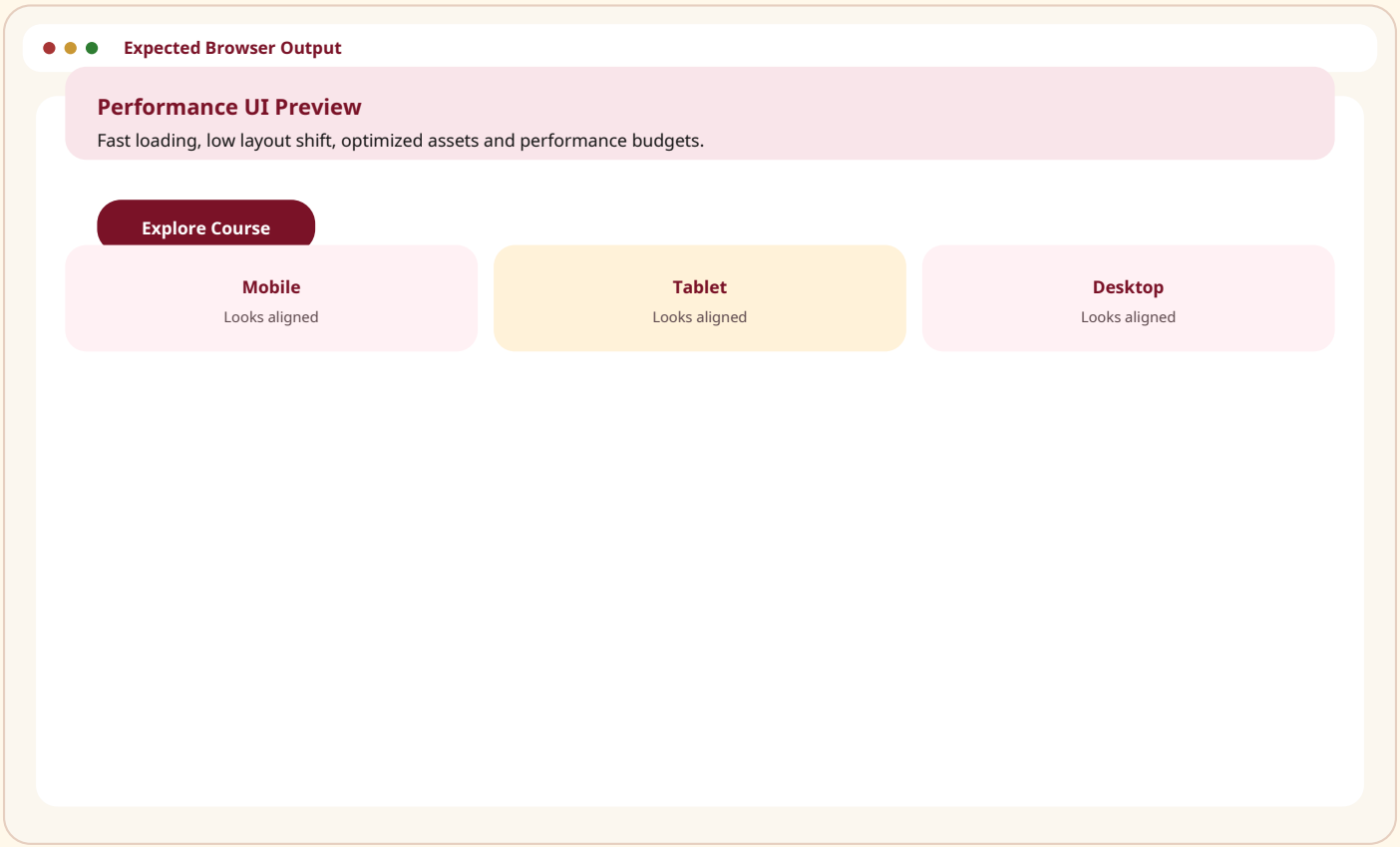
RUN AND UNDERSTAND

- Expected output: Optimized image loads with stable dimensions and no layout jump.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Web Performance and Core Web Vitals

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Web Performance and Core Web Vitals

HANDS-ON TASK

- Build one screen section that demonstrates fast loading, low layout shift, optimized assets and performance ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-44-web-performance-and-core-web-vital
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Web Performance and Core Web Vitals

WRITTEN QUESTIONS

- Q1. What problem does Web Performance and Core Web Vitals solve in real UI engineering?
- Q2. Which file/language is most commonly involved with Lighthouse lazy loading code split cache?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where web performance and core web vitals is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Web Performance and Core Web Vitals

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Web Performance and Core Web Vitals

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using Lighthouse lazy loading code split cache.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-44.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Performance improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 45: SEO, Metadata and Social Sharing

SEO | metadata sitemap robots OG JSO | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn search visibility, share cards, clean URLs and content hierarchy. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap search visibility, share cards, clean URLs and content hierarchy ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

SEO, Metadata and Social Sharing: Detailed Notes 1

English: These notes explain the practical engineering decisions behind metadata sitemap robots OG JSON-LD. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: SEO, Metadata and Social Sharing means applying metadata sitemap robots OG JSON-LD to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where seo, metadata and social sharing matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-45: add seo improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

SEO, Metadata and Social Sharing: Detailed Notes 2

English: These notes explain the practical engineering decisions behind metadata sitemap robots OG JSON-LD. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where seo, metadata and social sharing matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-45: add seo improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: SEO, Metadata and Social Sharing

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE style.css

```
:root {
--brand: #7a1227;
--cream: #fff8ea;
--text: #151515;
}
.card {
background: var(--cream);
color: var(--text);
border: 1px solid #e6cfc0;
border-radius: 24px;
padding: clamp(1rem, 2vw, 2rem);
}
```

```
$ Open page and inspect styles
```

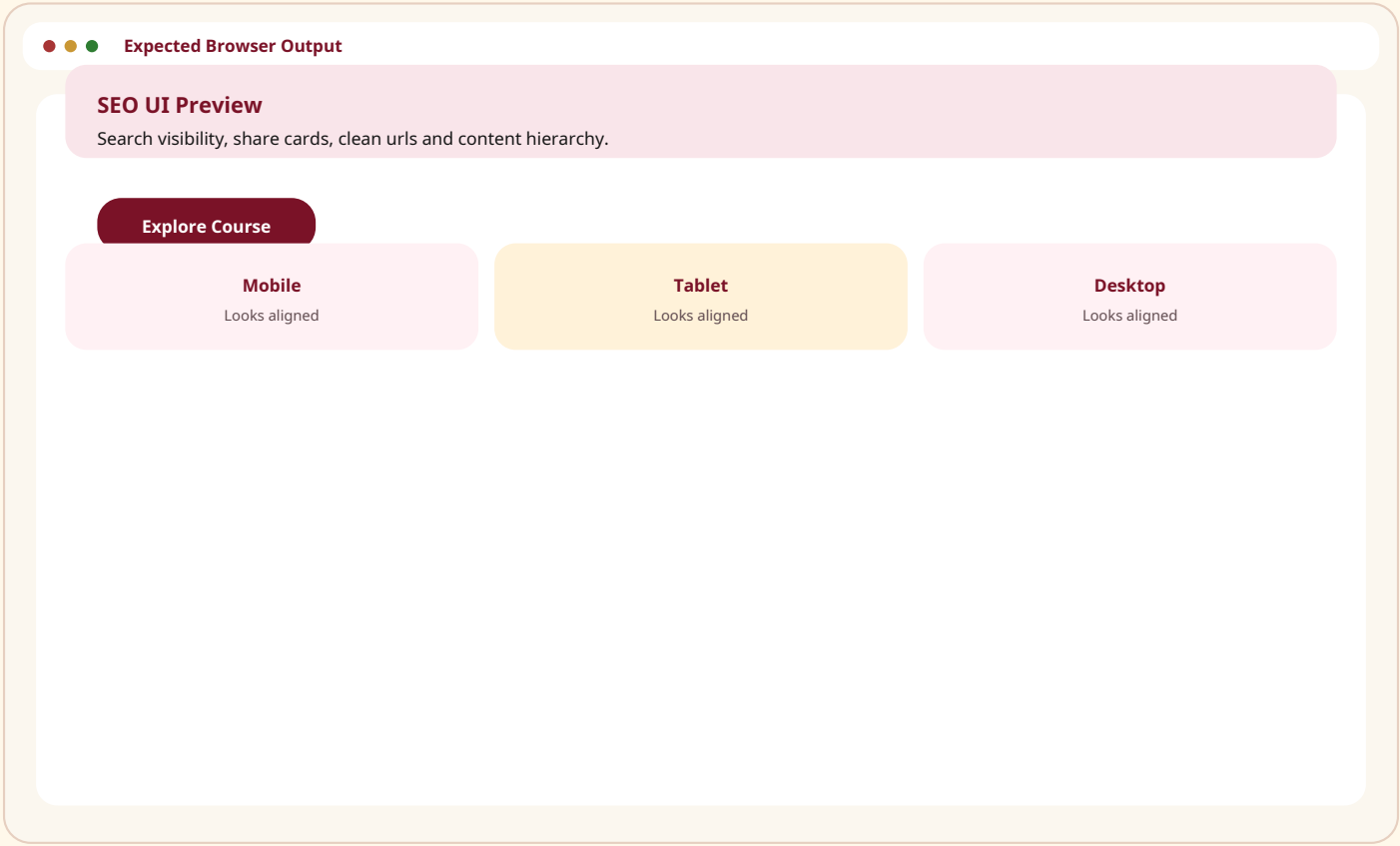
RUN AND UNDERSTAND

- Expected output: Card uses reusable tokens and scales spacing across devices.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: SEO, Metadata and Social Sharing

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: SEO, Metadata and Social Sharing

HANDS-ON TASK

- Build one screen section that demonstrates search visibility, share cards, clean URLs and content hierarchy.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-45-seo-metadata-and-social-sharing
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: SEO, Metadata and Social Sharing

WRITTEN QUESTIONS

- Q1. What problem does SEO, Metadata and Social Sharing solve in real UI engineering?
- Q2. Which file/language is most commonly involved with metadata sitemap robots OG JSON-LD?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where seo, metadata and social sharing is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: SEO, Metadata and Social Sharing

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: SEO, Metadata and Social Sharing

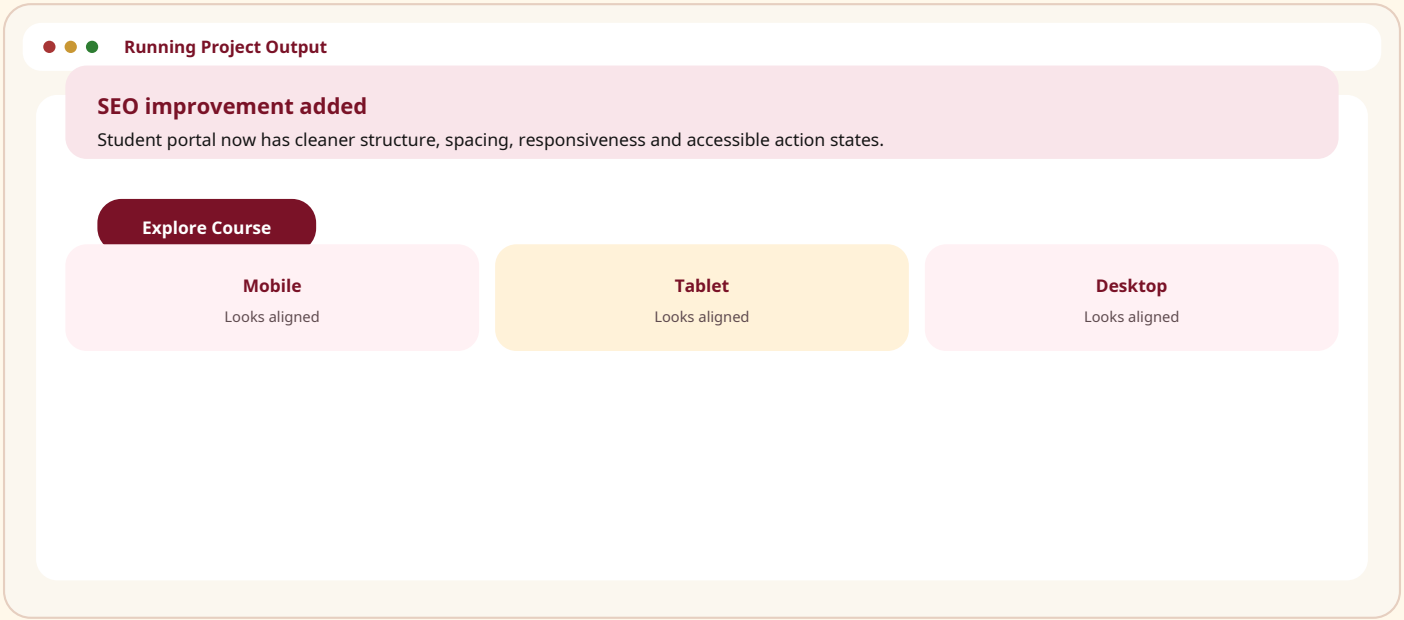
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using metadata sitemap robots OG JSON-LD.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-45.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 46: Frontend Security Essentials

Security | XSS CSRF CSP dependency audit | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn secure rendering, safe storage, token handling and dependency hygiene. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap secure rendering, safe storage, token handling and dependency hygiene ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Frontend Security Essentials: Detailed Notes 1

English: These notes explain the practical engineering decisions behind XSS CSRF CSP dependency audit secrets. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Frontend Security Essentials means applying XSS CSRF CSP dependency audit secrets to build reliable user ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where frontend security essentials matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-46: add security improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Frontend Security Essentials: Detailed Notes 2

English: These notes explain the practical engineering decisions behind XSS CSRF CSP dependency audit secrets. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where frontend security essentials matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-46: add security improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Frontend Security Essentials

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE utils/safeRender.ts

```
export function safeText(value: unknown): string {
return String(value ?? '')
.replaceAll('<', '&lt;')
.replaceAll('>', '&gt;')
.trim();
}

const userComment = '<img src=x onerror=alert(1) />';
document.querySelector('#comment')!.textContent = safeText(userComment);
```

```
$ npm run dev
```

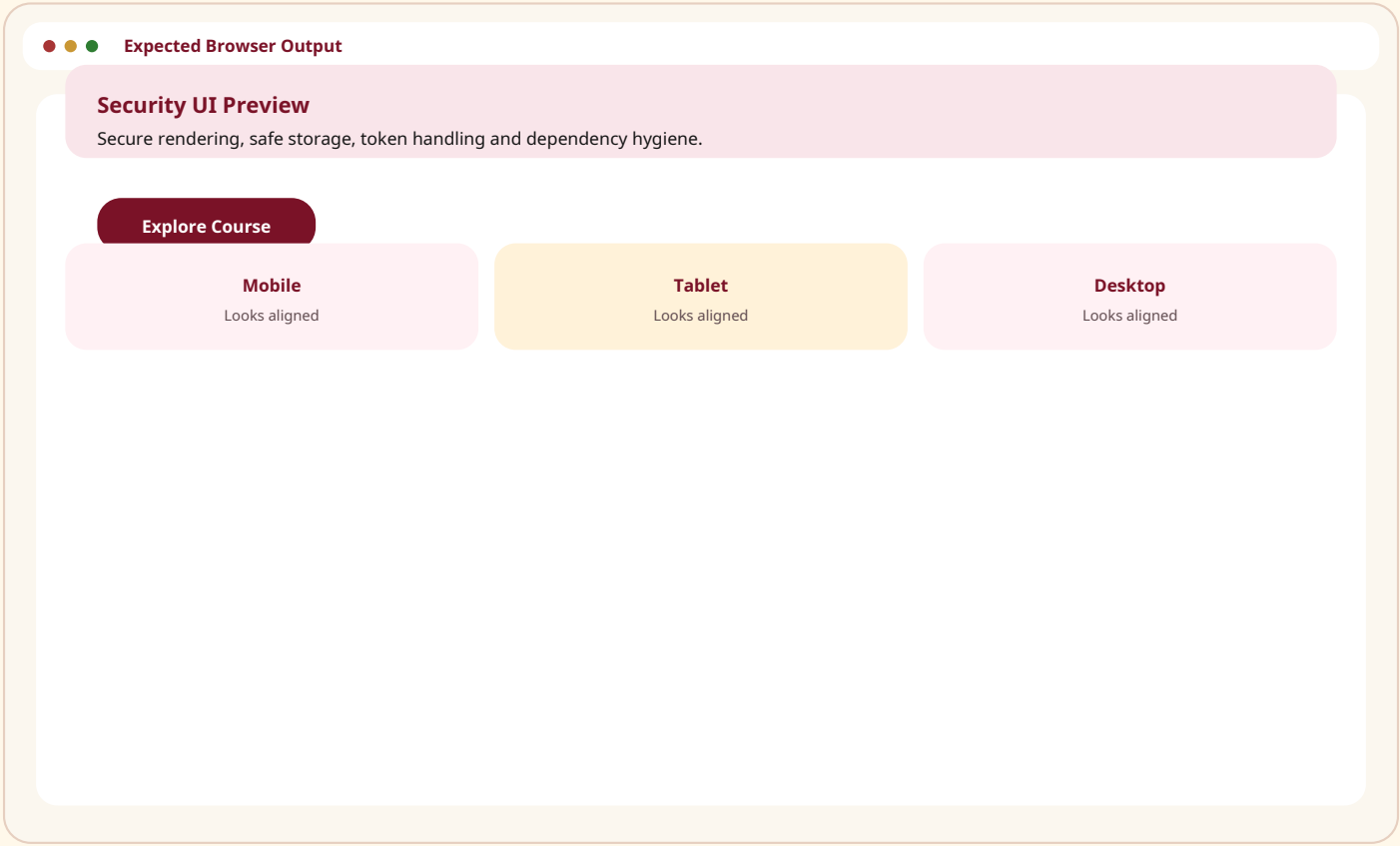
RUN AND UNDERSTAND

- Expected output: User text is displayed safely as text, not executed as HTML.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Frontend Security Essentials

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Frontend Security Essentials

HANDS-ON TASK

- Build one screen section that demonstrates secure rendering, safe storage, token handling and dependency ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-46-frontend-security-essentials
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Frontend Security Essentials

WRITTEN QUESTIONS

- Q1. What problem does Frontend Security Essentials solve in real UI engineering?
- Q2. Which file/language is most commonly involved with XSS CSRF CSP dependency audit secrets?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where frontend security essentials is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Frontend Security Essentials

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Frontend Security Essentials

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using XSS CSRF CSP dependency audit secrets.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-46.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Security improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 47: PWA, Offline UI and Browser Storage

PWA | service worker cache localStor | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn installable apps, offline states, storage strategy and sync thinking. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap installable apps, offline states, storage strategy and sync thinking ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

PWA, Offline UI and Browser Storage: Detailed Notes 1

English: These notes explain the practical engineering decisions behind service worker cache localStorage IndexedDB manifest. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: PWA, Offline UI and Browser Storage means applying service worker cache localStorage IndexedDB ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where pwa, offline ui and browser storage matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-47: add pwa improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

PWA, Offline UI and Browser Storage: Detailed Notes 2

English: These notes explain the practical engineering decisions behind service worker cache localStorage IndexedDB manifest. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where pwa, offline ui and browser storage matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-47: add pwa improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: PWA, Offline UI and Browser Storage

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/api.ts

```
export async function getCourses() {
const res = await fetch('/api/courses');
if (!res.ok) throw new Error('Courses failed to load');
return res.json();
}

export async function renderCourses() {
const courses = await getCourses();
console.log('Loaded courses:', courses.length);
}
```

```
$ npm run dev
```

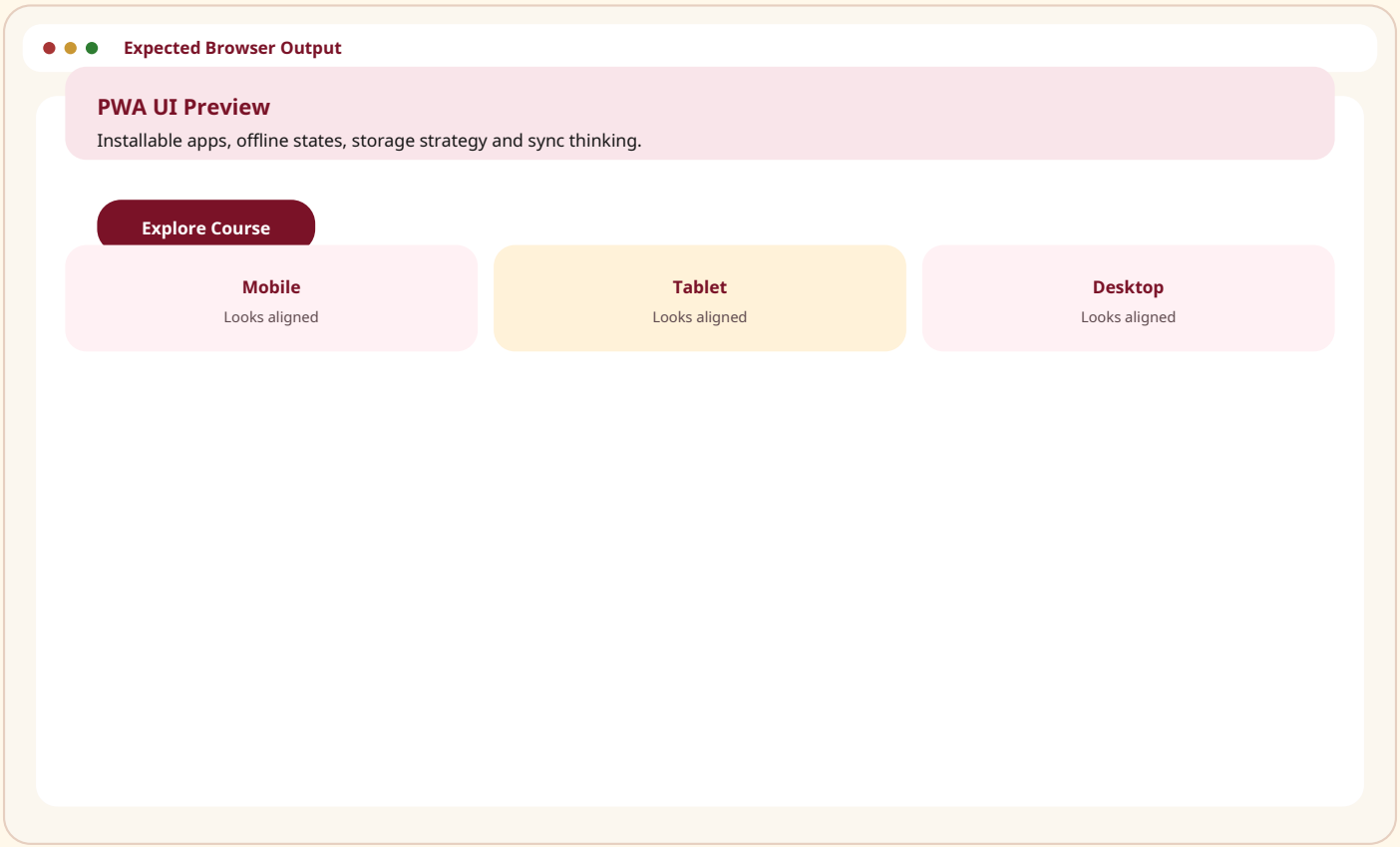
RUN AND UNDERSTAND

- Expected output: Course count logs after successful API call.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: PWA, Offline UI and Browser Storage

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: PWA, Offline UI and Browser Storage

HANDS-ON TASK

- Build one screen section that demonstrates installable apps, offline states, storage strategy and sync thinking.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-47-pwa-offline-ui-and-browser-storage
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: PWA, Offline UI and Browser Storage

WRITTEN QUESTIONS

- Q1. What problem does PWA, Offline UI and Browser Storage solve in real UI engineering?
- Q2. Which file/language is most commonly involved with service worker cache localStorage IndexedDB manifest?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where pwa, offline ui and browser storage is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: PWA, Offline UI and Browser Storage

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: PWA, Offline UI and Browser Storage

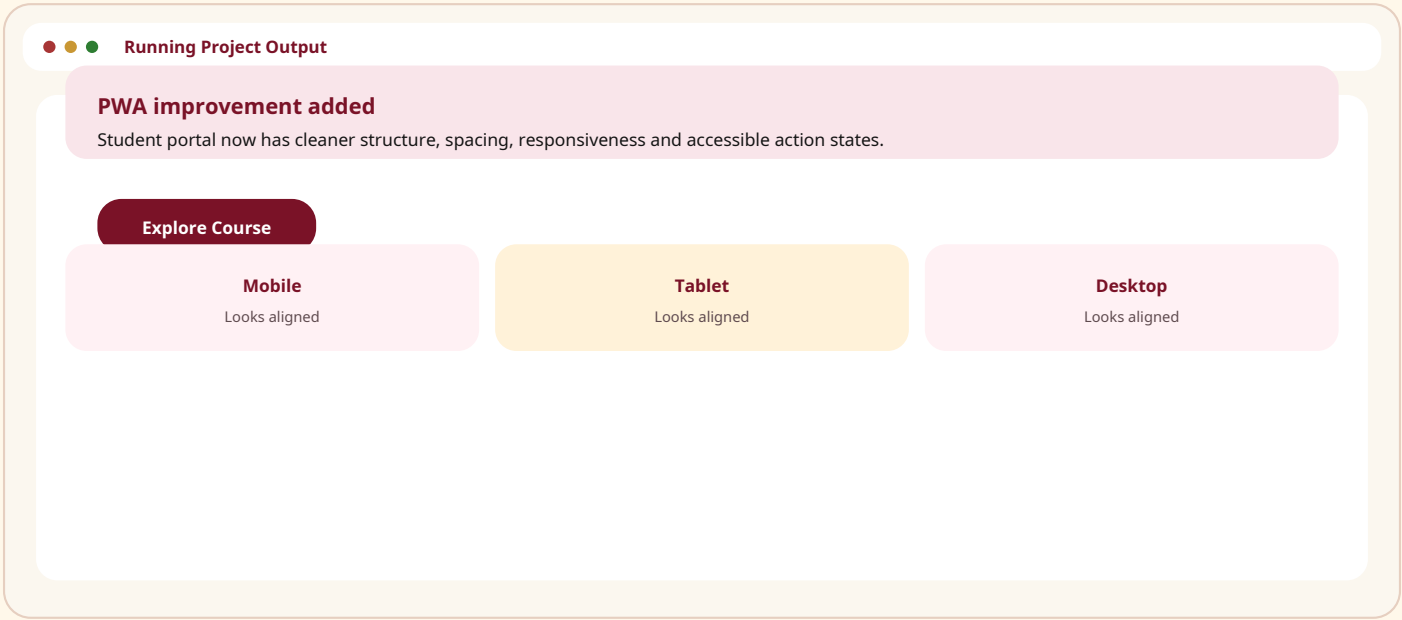
Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using service worker cache localStorage IndexedDB manifest.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-47.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 48: Deployment, CI/CD and Production Checks

DevOps for UI | Vercel Netlify Docker Nginx Gi | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn build, preview deploys, env vars, rollbacks and release checklists. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap build, preview deploys, env vars, rollbacks and release checklists ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Deployment, CI/CD and Production Checks: Detailed Notes 1

English: These notes explain the practical engineering decisions behind Vercel Netlify Docker Nginx GitHub Actions. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Deployment, CI/CD and Production Checks means applying Vercel Netlify Docker Nginx GitHub Actions ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where deployment, ci/cd and production checks matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-48: add devops for ui improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Deployment, CI/CD and Production Checks: Detailed Notes 2

English: These notes explain the practical engineering decisions behind Vercel Netlify Docker Nginx GitHub Actions. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where deployment, ci/cd and production checks matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-48: add devops for ui improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Deployment, CI/CD and Production Checks

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE index.html

```
<section class="feature-card">
<p class="eyebrow">UI Engineer Skill</p>
<h2>Build clean interfaces</h2>
<p>Use structure, spacing, contrast and interaction states.</p>
<a href="/learn" class="btn">Learn More</a>
</section>
```

```
$ Open index.html
```

RUN AND UNDERSTAND

- Expected output: Feature card appears with heading, paragraph and link button.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Deployment, CI/CD and Production Checks

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.

Expected Browser Output

```
$ npm run dev
> frontend-ui@1.0.0 dev
Local: http://localhost:5173/
Status: compiled successfully
Module: DevOps for UI
No console errors found
Screenshot saved: output/module.png
```

VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Deployment, CI/CD and Production Checks

HANDS-ON TASK

- Build one screen section that demonstrates build, preview deploys, env vars, rollbacks and release checklists.
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-48-deployment-ci-cd-and-production-ch
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Deployment, CI/CD and Production Checks

WRITTEN QUESTIONS

- Q1. What problem does Deployment, CI/CD and Production Checks solve in real UI engineering?
- Q2. Which file/language is most commonly involved with Vercel Netlify Docker Nginx GitHub Actions?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where deployment, ci/cd and production checks is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Deployment, CI/CD and Production Checks

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Deployment, CI/CD and Production Checks

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using Vercel Netlify Docker Nginx GitHub Actions.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-48.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

DevOps for UI improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 49: Other Frontend Frameworks and Languages

Ecosystem | Vue Angular Svelte Astro Web C | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn how different frameworks solve UI problems and when each language appears. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap how different frameworks solve UI problems and when each language appears ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Other Frontend Frameworks and Languages: Detailed Notes 1

English: These notes explain the practical engineering decisions behind Vue Angular Svelte Astro Web Components WASM. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Other Frontend Frameworks and Languages means applying Vue Angular Svelte Astro Web Components ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where other frontend frameworks and languages matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-49: add ecosystem improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Other Frontend Frameworks and Languages: Detailed Notes 2

English: These notes explain the practical engineering decisions behind Vue Angular Svelte Astro Web Components WASM. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where other frontend frameworks and languages matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-49: add ecosystem improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Other Frontend Frameworks and Languages

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE style.css

```
:root {
--brand: #7a1227;
--cream: #fff8ea;
--text: #151515;
}
.card {
background: var(--cream);
color: var(--text);
border: 1px solid #e6cfc0;
border-radius: 24px;
padding: clamp(1rem, 2vw, 2rem);
}
```

```
$ Open page and inspect styles
```

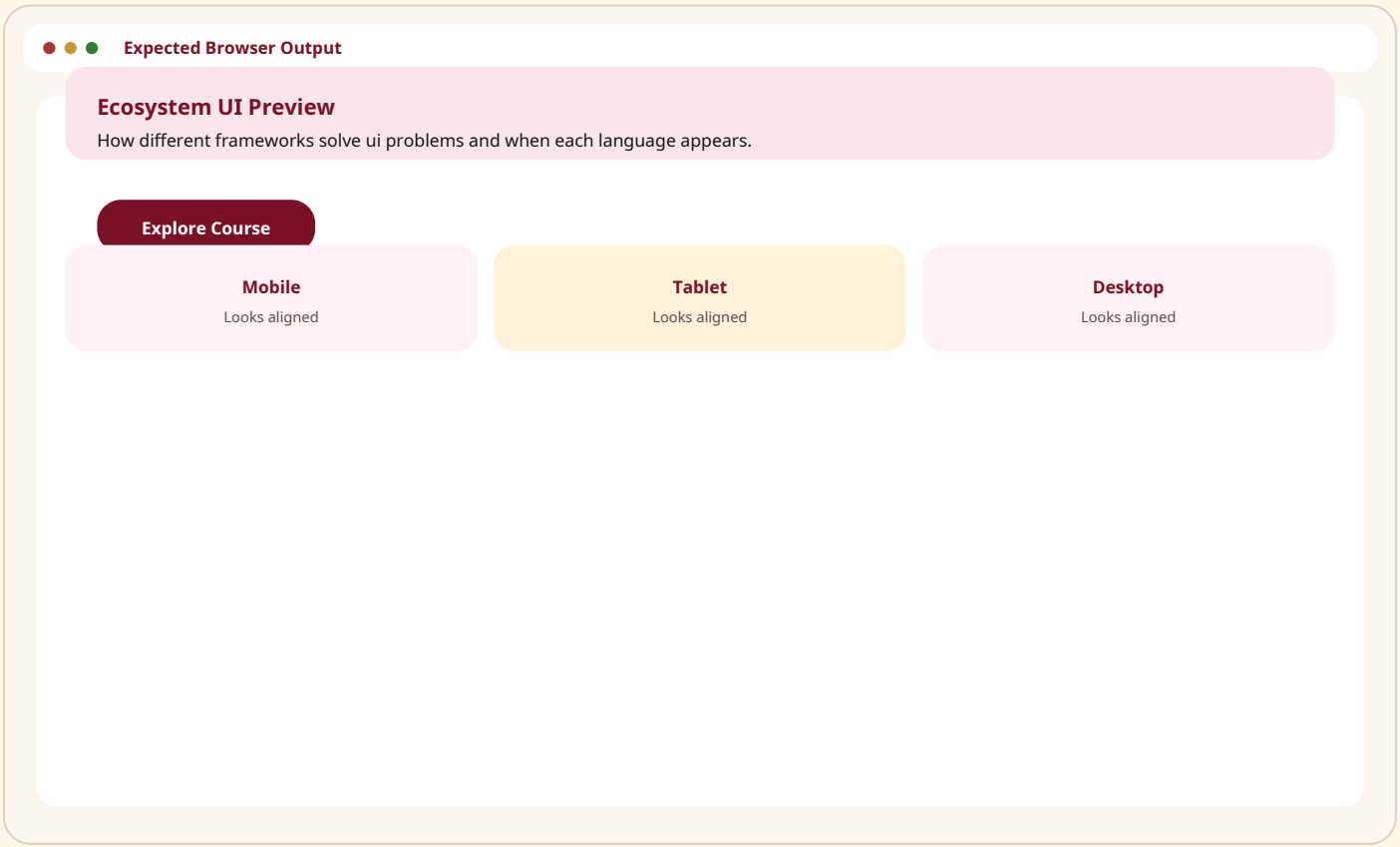
RUN AND UNDERSTAND

- Expected output: Card uses reusable tokens and scales spacing across devices.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Other Frontend Frameworks and Languages

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Other Frontend Frameworks and Languages

HANDS-ON TASK

- Build one screen section that demonstrates how different frameworks solve UI problems and when each language ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-49-other-frontend-frameworks-and-lang
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Other Frontend Frameworks and Languages

WRITTEN QUESTIONS

- Q1. What problem does Other Frontend Frameworks and Languages solve in real UI engineering?
- Q2. Which file/language is most commonly involved with Vue Angular Svelte Astro Web Components WASM?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where other frontend frameworks and languages is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Other Frontend Frameworks and Languages

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Other Frontend Frameworks and Languages

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using Vue Angular Svelte Astro Web Components WASM.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-49.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Ecosystem improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Module 50: Career, Interview and Capstone Delivery

Career | portfolio resume interviews sy | Concept | Beginner Friendly

WHAT YOU WILL LEARN

English: In this lesson you learn building professional portfolio, interview prep, freelance readiness and capstone submission. A frontend UI engineer must understand both visual quality and implementation quality, so the goal is not only to make the screen work but also to make it readable, responsive, accessible and maintainable.

Hinglish: Is lesson me aap building professional portfolio, interview prep, freelance readiness and capstone submission ko simple tarike se samjhoge. Sirf design banana enough nahi hai; UI clean, fast, accessible, responsive aur future me change karne layak hona chahiye.

PRODUCTION CHECKLIST

- Use meaningful names for files, classes, variables and components.
- Make the UI readable on mobile, tablet and desktop screens.
- Confirm keyboard support, focus state and accessible labels.
- Keep styles reusable; avoid random values and duplicate code.
- Test empty, loading, error and success states before submission.
- Write clear commit messages and document important decisions.

COMMON MISTAKES

- Building only for one screen size and ignoring mobile layout.
- Using divs for everything instead of semantic HTML where possible.
- Hiding focus outlines without replacing them with visible states.
- Writing code that works once but cannot be reused in another page.
- Not checking performance, image size or layout shift.
- Submitting without README, screenshots or run instructions.

MINI WORKFLOW FOR THIS TOPIC

RUN

Run the smallest code example first and keep the browser open beside your editor.

VERIFY

Verify alignment, spacing, colors, keyboard focus, mobile width and console errors.

SUBMIT

Submit code, screenshot, README notes and one Hinglish explanation of what changed.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Career, Interview and Capstone Delivery: Detailed Notes 1

English: These notes explain the practical engineering decisions behind portfolio resume interviews system design. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Definition: Career, Interview and Capstone Delivery means applying portfolio resume interviews system design to build ...
- Real use case: course card, dashboard widget, landing page section, form flow, modal, navbar or product page.
- Engineering rule: separate structure, style, behavior, state and data flow whenever possible.
- UI rule: spacing, alignment, contrast, hierarchy and states must be consistent across screens.
- Quality rule: write code that another developer can understand and safely change.
- Hinglish: Har screen ko clean system ke tarah socho - title, content, action, feedback aur error state.

TEACHER NOTES

- Before coding: read requirement, identify user action, list states and sketch layout quickly.
- During coding: build smallest component first, test in browser, then compose into full section.
- After coding: check responsiveness, accessibility, console errors, network errors and final polish.
- Code review: remove unused code, fix naming, avoid hardcoded magic numbers and add README notes.
- Teacher tip: every topic should end with one visible UI improvement in the running project.
- Hinglish: Kaam complete tab maana jayega jab screen real user ke liye easy aur stable ho.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where career, interview and capstone delivery matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-50: add career improvement".

PROOF BEFORE MOVING FORWARD

RUN

Run the page locally and save one screenshot of the correct output.

VERIFY

Verify there is no header overlap, footer overlap, broken link or horizontal scroll.

SUBMIT

Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Career, Interview and Capstone Delivery: Detailed Notes 2

English: These notes explain the practical engineering decisions behind portfolio resume interviews system design. Hinglish: yaha theory ko real project decision se connect kiya gaya hai, taki student sirf yaad na kare balki apply bhi kare.

CORE EXPLANATION

- Beginner level: understand syntax, copy small example, change text/color/spacing and observe output.
- Intermediate level: convert example into reusable component or reusable CSS class.
- Advanced level: handle states, edge cases, accessibility, testing and design-system consistency.
- Professional level: document the component and make it safe for team usage.
- Debug habit: inspect element, console, network, performance and accessibility tree.
- Hinglish: Pehle chhota banao, fir reusable banao, fir production quality tak polish karo.

TEACHER NOTES

- Output expectation: visually balanced screen, no text overlap, no accidental horizontal scroll.
- Responsive expectation: layout remains usable from 360px mobile to wide desktop.
- Accessibility expectation: keyboard users can reach and understand every control.
- Performance expectation: images and code should not slow down the first screen unnecessarily.
- Security expectation: never trust user-generated content and never expose secrets in frontend code.
- Hinglish: User ko confusion nahi hona chahiye ki next click kaha karna hai ya error kaise fix karna hai.

HOW TO APPLY THIS IN THE RUNNING PROJECT

- Find the student portal area where career, interview and capstone delivery matters most.
- Create or update one small file first; avoid touching unrelated design until the output is stable.
- Run locally, compare output with preview, inspect layout and fix responsive issues.
- Write one mini test or manual checklist item before marking the lesson complete.
- Commit with message: "module-50: add career improvement".

PROOF BEFORE MOVING FORWARD

RUN	VERIFY	SUBMIT
Run the page locally and save one screenshot of the correct output.	Verify there is no header overlap, footer overlap, broken link or horizontal scroll.	Submit a short note: what file changed, what output improved and what test was done.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Code Lab: Career, Interview and Capstone Delivery

Copy the code, run the command and compare your output. English first, then Hinglish support is written in the checklist below.

CODE RUN EXAMPLE src/main.js

```
const tabs = document.querySelectorAll('[data-tab]');
const panels = document.querySelectorAll('[data-panel]');

tabs.forEach((tab) => {
tab.addEventListener('click', () => {
panels.forEach((panel) => panel.hidden = panel.id !== tab.dataset.tab);
});
});
```

```
$ Open tabs.html and click tabs
```

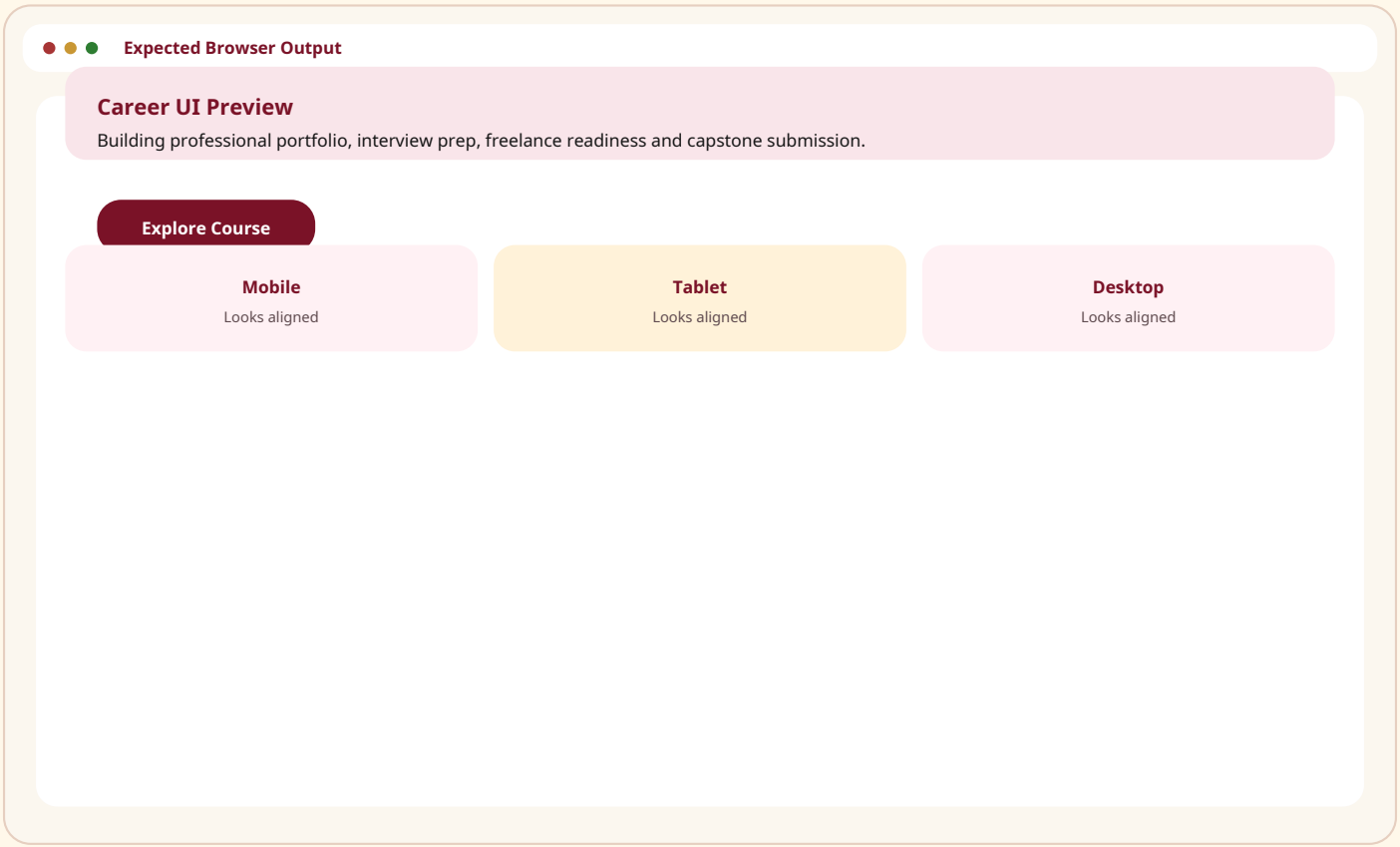
RUN AND UNDERSTAND

- Expected output: Only the selected tab panel remains visible.
- English: Do not paste code blindly. Change one value at a time and observe the browser result.
- Hinglish: Code paste karne ke baad text, color, spacing ya data change karke samjho ki kaunsa part kya kar raha hai.
- Check browser console. There should be no red error unless the lesson intentionally creates one for debugging.
- Add screenshot/output proof in your assignment folder before pushing to GitHub.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Output Preview Screen: Career, Interview and Capstone Delivery

This page shows how your running screen should look or behave. Use it like a visual checkpoint before submitting the task.



VISUAL QUALITY CHECK

- No text touches the border or overlaps another component.
- Every card has enough padding and consistent border radius.
- Important action button is visible without searching.
- Text color has strong contrast on the background.
- Mobile layout does not create sideways scrolling.

HINGLISH REVIEW

- Screen dekhte hi user ko samajh aana chahiye ki kya karna hai.
- Button click karne layak, readable aur properly aligned hona chahiye.
- Image/text overlap nahi hona chahiye, spacing clean honi chahiye.
- Aadha page blank nahi chhodna; content balanced cards me arrange karo.
- Output screenshot ko assignment ke saath attach karo.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Practice Task: Career, Interview and Capstone Delivery

HANDS-ON TASK

- Build one screen section that demonstrates building professional portfolio, interview prep, freelance readiness ...
- Add one primary action and one secondary action with correct hover/focus/disabled states.
- Make the layout responsive at 360px, 768px, 1024px and desktop width.
- Add one empty/loading/error state if the topic involves data or user input.
- Write a short README: what you built, how to run, screenshot path and what you learned.
- Commit and push to GitHub with a descriptive message.

EXAM-STYLE QUESTIONS

- Define the topic in one sentence.
- Write one real-world use case from a course platform or dashboard.
- Explain one accessibility issue that could appear in this topic.
- Explain one performance or maintainability issue.
- Show before/after screenshot and list the improvement.

SUBMISSION FORMAT

- Folder name: module-50-career-interview-and-capstone-deli
- Required files: source code, README.md, screenshot.png, optional notes.md.
- README must include: command to run, browser URL, what was tested and known limitations.
- Hinglish: Assignment tabhi complete hoga jab code run ho, output screenshot ho aur explanation simple ho.
- Teacher grading: 40% correctness, 20% UI quality, 20% responsiveness, 10% accessibility, 10% documentation.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Quick Exam: Career, Interview and Capstone Delivery

WRITTEN QUESTIONS

- Q1. What problem does Career, Interview and Capstone Delivery solve in real UI engineering?
- Q2. Which file/language is most commonly involved with portfolio resume interviews system design?
- Q3. What would break if the screen is tested only on desktop?
- Q4. Write one accessibility check for this module.
- Q5. Write one debugging step using browser DevTools.
- Q6. Explain the topic in Hinglish to a beginner student.
- Q7. Write one small improvement you can add to the running project.
- Q8. What screenshot/output proof should be submitted?

MARKING RUBRIC

- Excellent: answer connects concept to real project and mentions user impact.
- Good: answer explains syntax and at least one UI example.
- Needs improvement: answer only copies theory without explaining usage.
- Fail: answer cannot run code or cannot show output screenshot.
- Bonus: student writes reusable component, test or Storybook example.

MINI ORAL VIVA GUIDE

- Teacher asks: Show where career, interview and capstone delivery is used in your code.
- Student should open browser, inspect element and explain HTML/CSS/JS or component relationship.
- Student should show one responsive breakpoint and one focus/keyboard interaction.
- Student should explain in English first and then Hinglish: "simple words me iska kaam yeh hai..."
- Student should mention one mistake they made and how they fixed it.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Debugging and Professional Fixes: Career, Interview and Capstone Delivery

CODE RUN EXAMPLE debug/checklist.js

```
// Debug checklist helper
function assertUI(condition, message) {
if (!condition) {
console.warn('UI check failed:', message);
}
}

assertUI(document.querySelector('main'), 'Page must have a main landmark');
assertUI(!document.body.scrollWidth > window.innerWidth, 'Avoid horizontal overflow');
assertUI(document.querySelectorAll('img[alt]').length >= 0, 'Images need useful alt text');
```

```
$ Open DevTools Console
```

COMMON BUG SYMPTOMS

- Text overlaps header, footer or another card.
- Page has too much blank area because content is not distributed well.
- Card width looks fine on desktop but breaks on mobile.
- Button has no hover/focus state or wrong cursor behavior.
- Console shows error after clicking UI.
- Image loads slowly or causes layout jump.

HOW TO FIX PROFESSIONALLY

- Reserve fixed header/footer height and keep body content inside safe area.
- Use grid/flex with gaps instead of manual random margins.
- Set max-width and responsive breakpoints for large sections.
- Add visible focus ring and disabled state where needed.
- Use DevTools Elements, Console, Network and Lighthouse tabs.
- Compress images and define width/height or aspect-ratio.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Project Application: Career, Interview and Capstone Delivery

Apply the module to the one running product: AIWorldAcademy student portal. The goal is visible improvement plus clean engineering.

BUILD THIS

- Add or improve a section using portfolio resume interviews system design.
- Keep header, footer and course-card alignment consistent.
- Add real content instead of lorem ipsum wherever possible.
- Write small reusable class/component for repeated UI.
- Include output screenshot in /screenshots/module-50.png.
- Write a 5-line reflection: what was hard, how you fixed it, what you learned.

ACCEPTANCE CRITERIA

- No visible overlap with header/footer.
- No half-empty unbalanced page/card unless intentionally designed.
- Desktop and mobile screenshots are submitted.
- All links/buttons have correct purpose and feedback.
- Code is formatted and pushed to GitHub.
- Teacher can run it using README instructions.

Running Project Output

Career improvement added

Student portal now has cleaner structure, spacing, responsiveness and accessible action states.

Explore Course

Mobile	Tablet	Desktop
Looks aligned	Looks aligned	Looks aligned

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 1: Responsive Portfolio Website

HTML, CSS, JS | Portfolio | Code + Output | Teacher Review

English: Create a personal portfolio with hero, skills, projects, contact form and dark/light theme. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

CODE RUN EXAMPLE projects/project-1.js

```
// Project 1: Responsive Portfolio Website
const project = {
title: 'Responsive Portfolio Website',
stack: 'HTML, CSS, JS',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/1" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 2: AIWorldAcademy Course Catalog UI

React, TypeScript, CSS Grid | Portfolio | Code + Output | Teacher Review

English: Build searchable course cards with filters, empty state and responsive layout. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

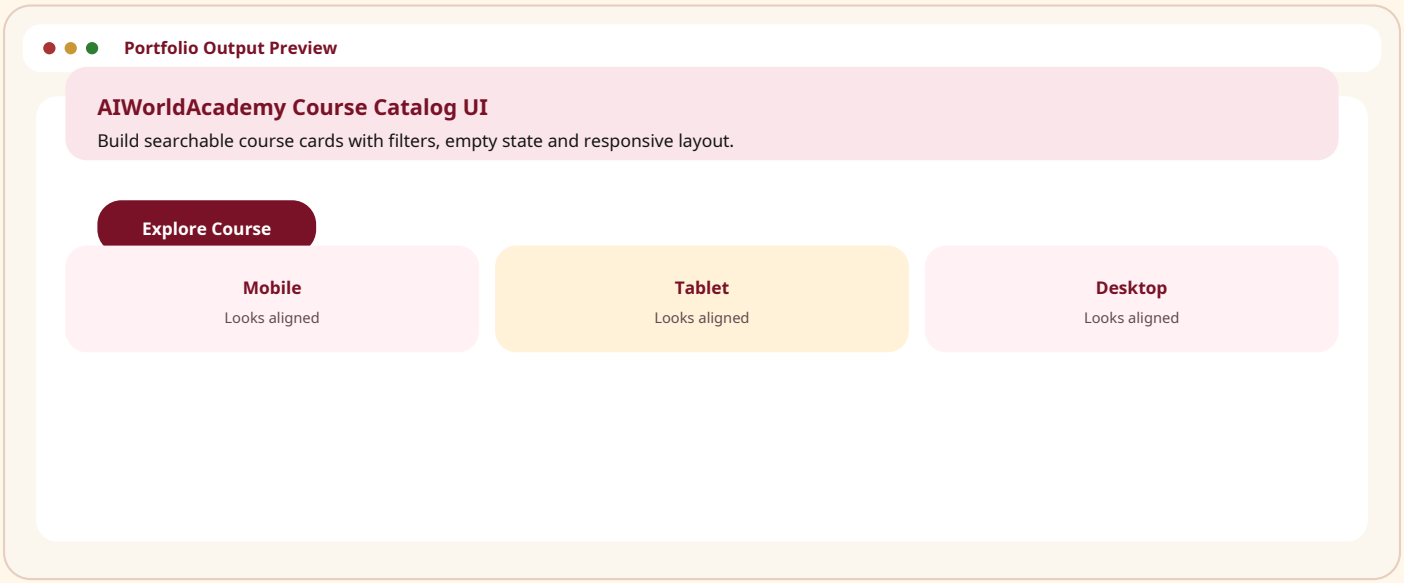
CODE RUN EXAMPLE projects/project-2.js

```
// Project 2: AIWorldAcademy Course Catalog UI
const project = {
title: 'AIWorldAcademy Course Catalog UI',
stack: 'React, TypeScript, CSS Grid',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/2" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 3: Student Dashboard UI

Next.js, Tailwind | Portfolio | Code + Output | Teacher Review

English: Dashboard with progress cards, upcoming lessons, certificates and profile chip. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

CODE RUN EXAMPLE projects/project-3.js

```
// Project 3: Student Dashboard UI
const project = {
title: 'Student Dashboard UI',
stack: 'Next.js, Tailwind',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/3" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 4: Accessible Modal and Menu Library

React, ARIA | Portfolio | Code + Output | Teacher Review

English: Reusable dialog, dropdown and tabs with keyboard support. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

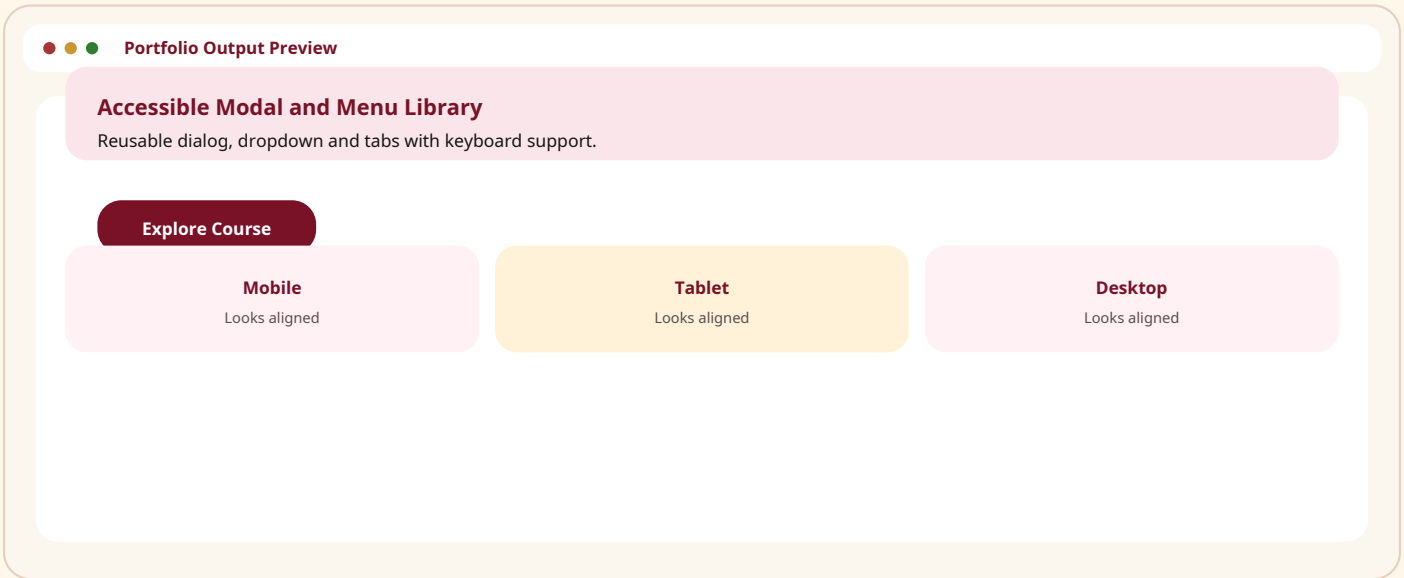
CODE RUN EXAMPLE projects/project-4.js

```
// Project 4: Accessible Modal and Menu Library
const project = {
title: 'Accessible Modal and Menu Library',
stack: 'React, ARIA',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/4" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 5: Design Token System

CSS Variables, JSON | Portfolio | Code + Output | Teacher Review

English: Brand colors, spacing, radius, typography and theme switching. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

CODE RUN EXAMPLE projects/project-5.js

```
// Project 5: Design Token System
const project = {
title: 'Design Token System',
stack: 'CSS Variables, JSON',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/5" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 6: Authentication UI Flow

React, Forms | Portfolio | Code + Output | Teacher Review

English: Login, register, OTP, forgot password and protected route screens. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

CODE RUN EXAMPLE projects/project-6.js

```
// Project 6: Authentication UI Flow
const project = {
title: 'Authentication UI Flow',
stack: 'React, Forms',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/6" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 7: Course Lesson Player UI

HTML5 Video, React | Portfolio | Code + Output | Teacher Review

English: Video player layout with notes, resources, progress and completion state. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

CODE RUN EXAMPLE projects/project-7.js

```
// Project 7: Course Lesson Player UI
const project = {
title: 'Course Lesson Player UI',
stack: 'HTML5 Video, React',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/7" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 8: Resume Builder UI

React, Forms, PDF preview | Portfolio | Code + Output | Teacher Review

English: Multi-step resume form with templates and live preview. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

CODE RUN EXAMPLE projects/project-8.js

```
// Project 8: Resume Builder UI
const project = {
title: 'Resume Builder UI',
stack: 'React, Forms, PDF preview',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/8" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 9: E-commerce Product UI

Next.js, API | Portfolio | Code + Output | Teacher Review

English: Product listing, details page, cart drawer and checkout UI states. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

CODE RUN EXAMPLE projects/project-9.js

```
// Project 9: E-commerce Product UI
const project = {
title: 'E-commerce Product UI',
stack: 'Next.js, API',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/9" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 10: Admin Analytics Dashboard

Charts, Tables | Portfolio | Code + Output | Teacher Review

English: Metric cards, charts, filters, table sorting and export button. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

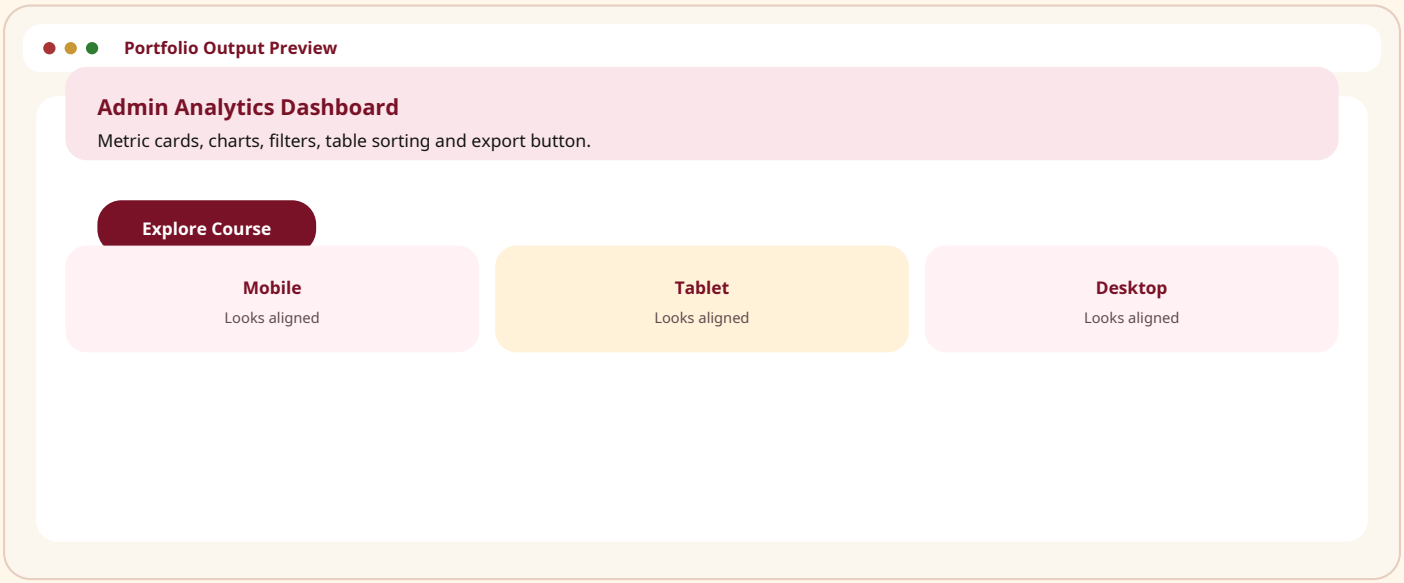
CODE RUN EXAMPLE projects/project-10.js

```
// Project 10: Admin Analytics Dashboard
const project = {
title: 'Admin Analytics Dashboard',
stack: 'Charts, Tables',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/10" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 11: PWA Notes App

Service Worker, IndexedDB | Portfolio | Code + Output | Teacher Review

English: Offline notes with install prompt and sync-ready UI. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

CODE RUN EXAMPLE projects/project-11.js

```
// Project 11: PWA Notes App
const project = {
title: 'PWA Notes App',
stack: 'Service Worker, IndexedDB',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/11" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 12: Storybook Component Library

Storybook, React | Portfolio | Code + Output | Teacher Review

English: Document buttons, inputs, cards, modals and layout components. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

CODE RUN EXAMPLE projects/project-12.js

```
// Project 12: Storybook Component Library
const project = {
title: 'Storybook Component Library',
stack: 'Storybook, React',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/12" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 13: Playwright Tested LMS Flow

Playwright | Portfolio | Code + Output | Teacher Review

English: E2E test for login, open course, complete lesson and view certificate eligibility. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

CODE RUN EXAMPLE projects/project-13.js

```
// Project 13: Playwright Tested LMS Flow
const project = {
title: 'Playwright Tested LMS Flow',
stack: 'Playwright',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/13" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 14: Performance Optimized Landing Page

Next Image, Lighthouse | Portfolio | Code + Output | Teacher Review

English: Hero image optimization, lazy sections and clean metadata. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

CODE RUN EXAMPLE projects/project-14.js

```
// Project 14: Performance Optimized Landing Page
const project = {
title: 'Performance Optimized Landing Page',
stack: 'Next Image, Lighthouse',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/14" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 15: Frontend Security Demo UI

Safe rendering, CSP awareness | Portfolio | Code + Output | Teacher Review

English: Show safe text rendering and explain XSS prevention. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

CODE RUN EXAMPLE projects/project-15.js

```
// Project 15: Frontend Security Demo UI
const project = {
title: 'Frontend Security Demo UI',
stack: 'Safe rendering, CSP awareness',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/15" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 16: Multi-language UI

i18n, RTL basics | Portfolio | Code + Output | Teacher Review

English: English/Hinglish content switcher and RTL-ready layout test. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

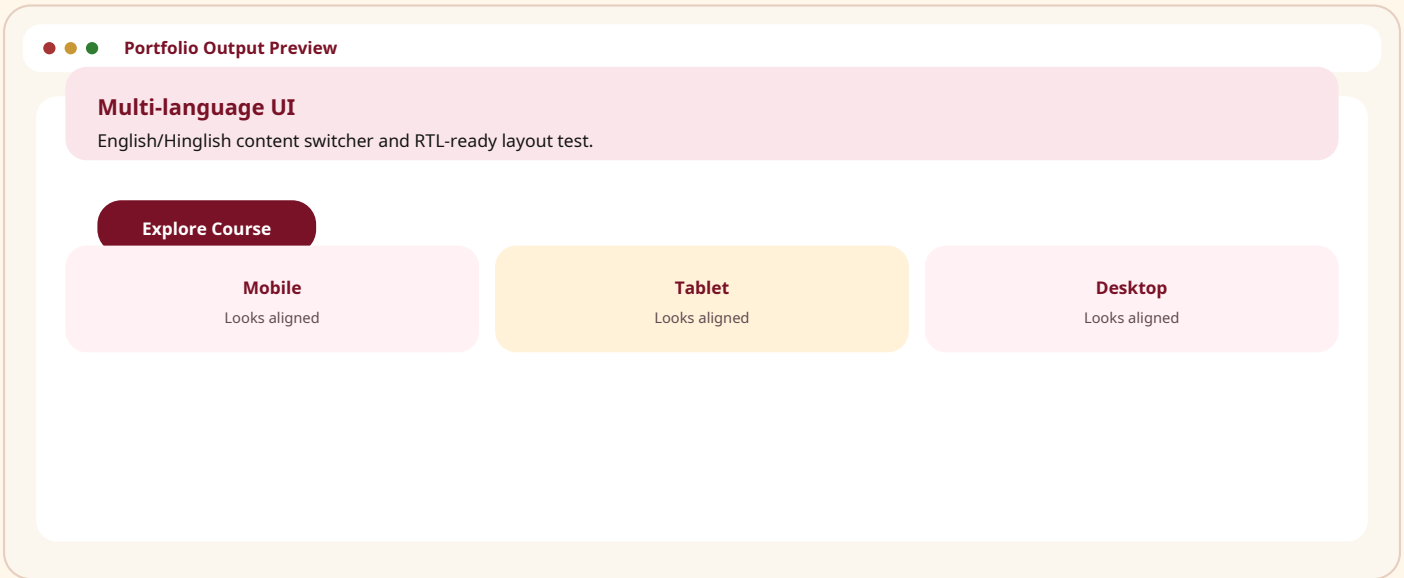
CODE RUN EXAMPLE projects/project-16.js

```
// Project 16: Multi-language UI
const project = {
title: 'Multi-language UI',
stack: 'i18n, RTL basics',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/16" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 17: Figma to Code Challenge

Figma, CSS, React | Portfolio | Code + Output | Teacher Review

English: Convert design screenshot to responsive production component. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

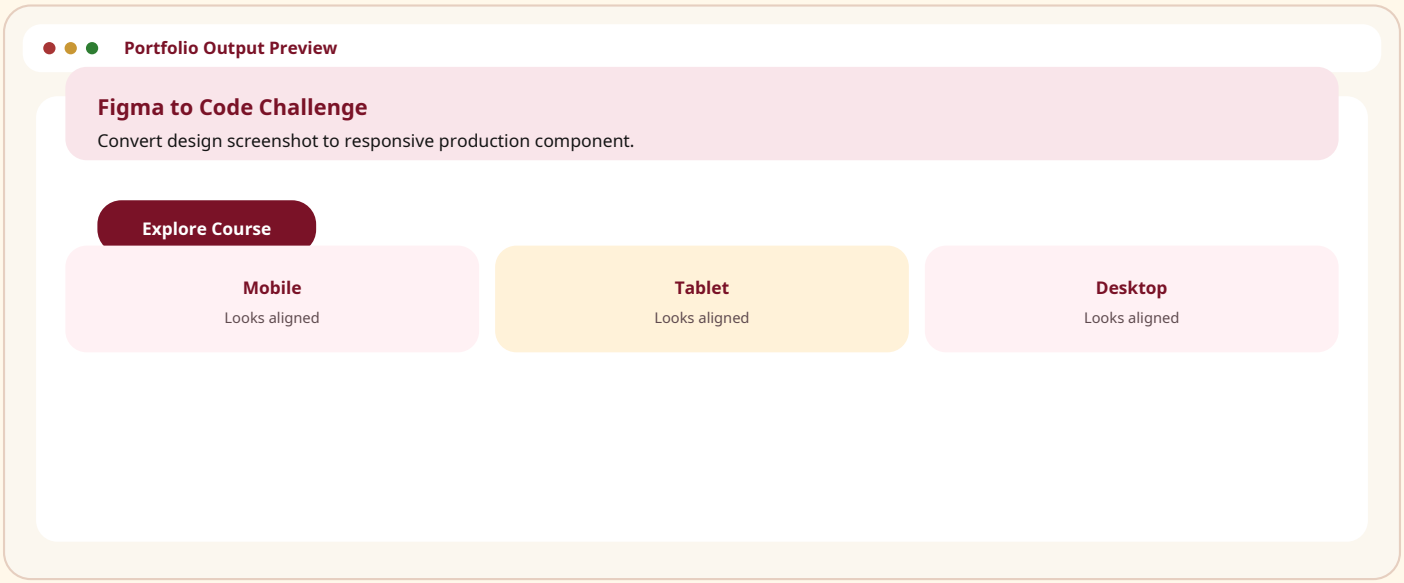
CODE RUN EXAMPLE projects/project-17.js

```
// Project 17: Figma to Code Challenge
const project = {
title: 'Figma to Code Challenge',
stack: 'Figma, CSS, React',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/17" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 18: Animated Onboarding Flow

Framer Motion/CSS | Portfolio | Code + Output | Teacher Review

English: Step-by-step onboarding with reduced-motion support. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

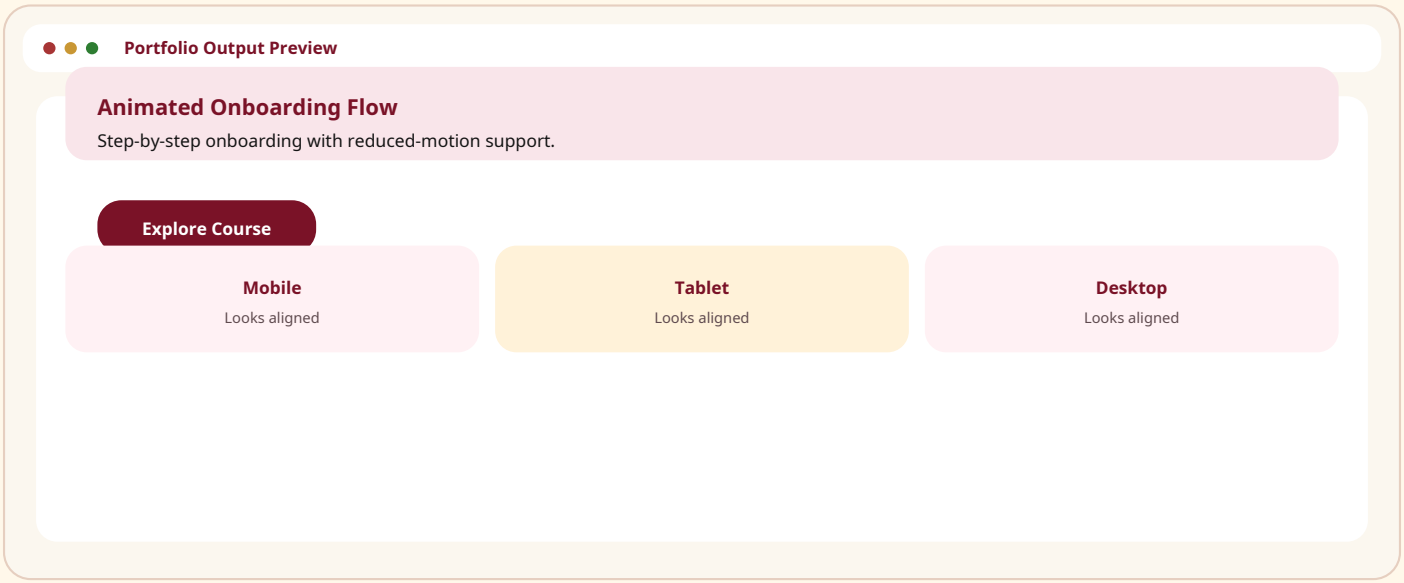
CODE RUN EXAMPLE projects/project-18.js

```
// Project 18: Animated Onboarding Flow
const project = {
title: 'Animated Onboarding Flow',
stack: 'Framer Motion/CSS',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/18" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 19: Data Table Pro UI

React Table style logic | Portfolio | Code + Output | Teacher Review

English: Search, sort, pagination, status badges and responsive table view. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

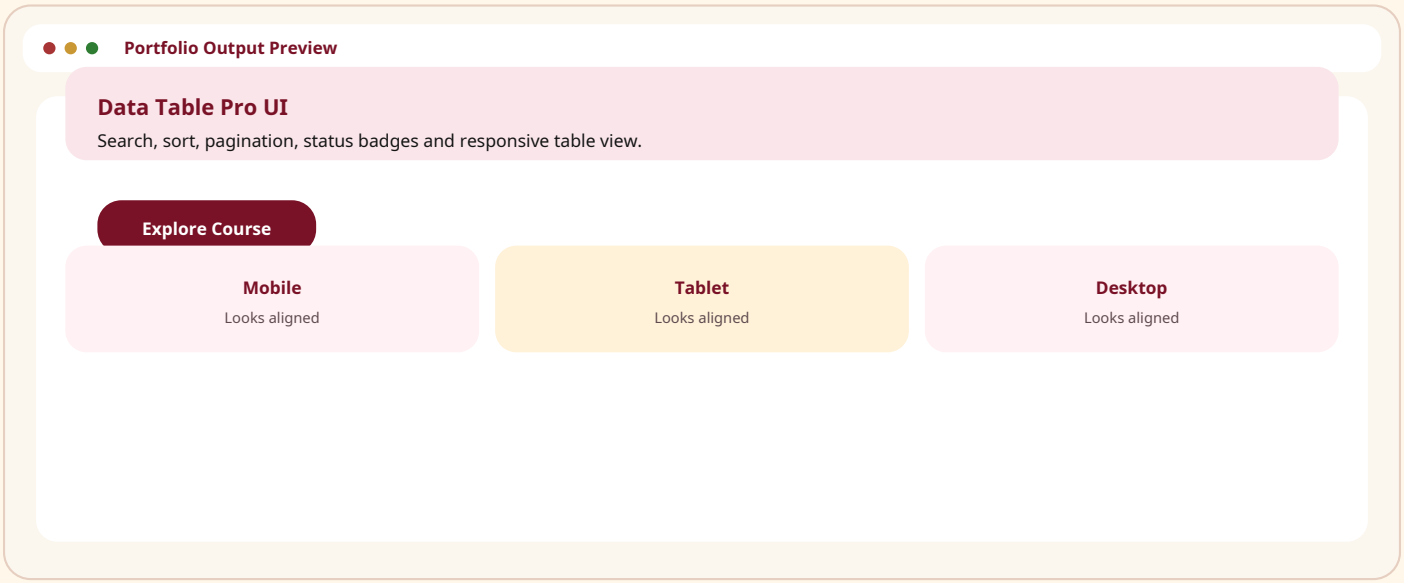
CODE RUN EXAMPLE projects/project-19.js

```
// Project 19: Data Table Pro UI
const project = {
title: 'Data Table Pro UI',
stack: 'React Table style logic',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/19" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio Project 20: Final Capstone Frontend UI Platform

All skills | Portfolio | Code + Output | Teacher Review

English: Complete AIWorldAcademy-like student learning portal with deployed link. Hinglish: Is project ko real portfolio level par banao - clean design, proper README, screenshot, deployed link aur GitHub code ke saath.

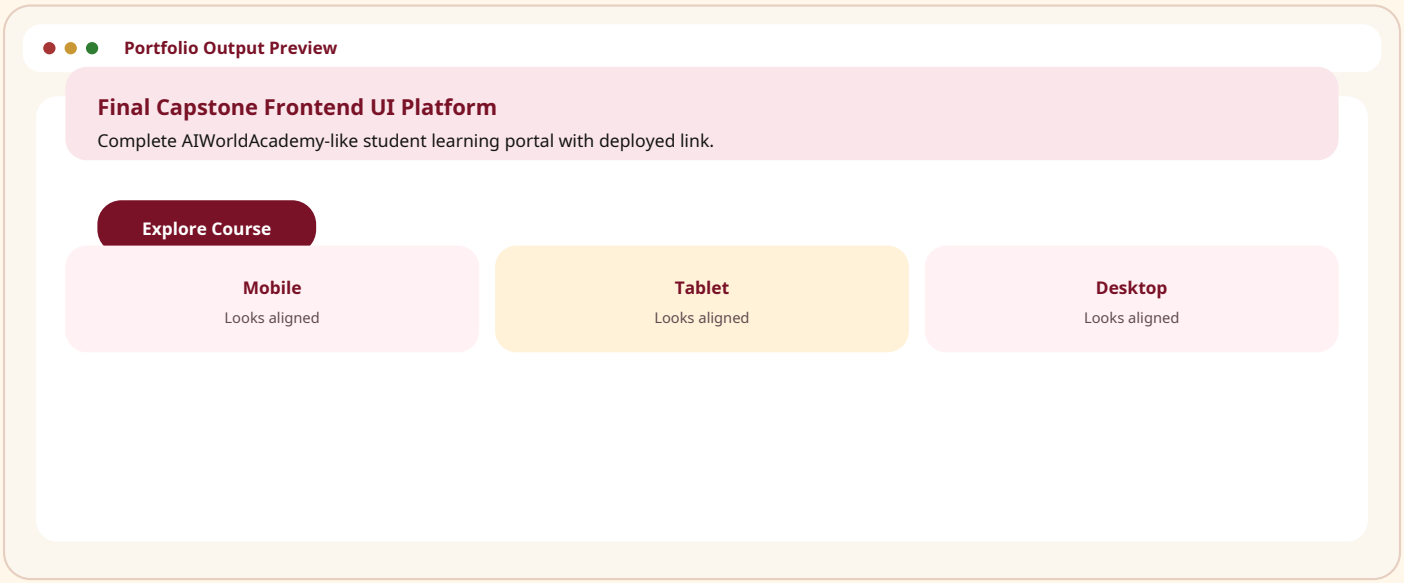
CODE RUN EXAMPLE projects/project-20.js

```
// Project 20: Final Capstone Frontend UI Platform
const project = {
title: 'Final Capstone Frontend UI Platform',
stack: 'All skills',
status: 'in-progress',
output: 'responsive production UI'
};

function renderProjectCard(project) {
return `<article class="project-card">
<h2>${project.title}</h2>
<p>${project.stack}</p>
<a href="/projects/20" class="btn">Open Project</a>
</article>`;
}

document.querySelector('#projects').innerHTML += renderProjectCard(project);
```

```
$ npm run dev
```



Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Capstone Sprint 1: Requirement and UX map

English: This sprint converts all course knowledge into one complete frontend UI engineering product. Hinglish: Ab alag-alag topics ko ek real website/app me combine karna hai.

SPRINT STEPS

- Define target users and top 5 user journeys.
- Draw page list and route map.
- Write content outline for each screen.
- Define success metric for each journey.

DELIVERABLES

- Create clear task checklist and GitHub issue/README section.
- Build the required UI screens with responsive layout and component reuse.
- Add loading, error, empty and success states where data is involved.
- Run manual test, accessibility check and performance check.
- Submit deployed link, GitHub link and final screenshots.

CODE RUN EXAMPLE capstone/sprint-1.ts

```
// Capstone Sprint 1
export const sprint = {
name: 'Requirement and UX map',
status: 'ready-for-review',
checklist: 4,
};

console.log('Capstone sprint ready:', sprint.name);
```

```
$ npm run build
```


Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Capstone Sprint 2: Design tokens and layout system

English: This sprint converts all course knowledge into one complete frontend UI engineering product. Hinglish: Ab alag-alag topics ko ek real website/app me combine karna hai.

SPRINT STEPS

- Create color, spacing, type, radius and shadow tokens.
- Create grid/container rules.
- Create button, input and card variants.
- Document theme usage.

DELIVERABLES

- Create clear task checklist and GitHub issue/README section.
- Build the required UI screens with responsive layout and component reuse.
- Add loading, error, empty and success states where data is involved.
- Run manual test, accessibility check and performance check.
- Submit deployed link, GitHub link and final screenshots.

CODE RUN EXAMPLE capstone/sprint-2.ts

```
// Capstone Sprint 2
export const sprint = {
name: 'Design tokens and layout system',
status: 'ready-for-review',
checklist: 4,
};

console.log('Capstone sprint ready:', sprint.name);
```

```
$ npm run build
```


Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Capstone Sprint 3: Home and course catalog

English: This sprint converts all course knowledge into one complete frontend UI engineering product. Hinglish: Ab alag-alag topics ko ek real website/app me combine karna hai.

SPRINT STEPS

- Build hero, course grid, filters and CTA sections.
- Add mobile navigation.
- Add empty and loading states.
- Add screenshot proofs.

DELIVERABLES

- Create clear task checklist and GitHub issue/README section.
- Build the required UI screens with responsive layout and component reuse.
- Add loading, error, empty and success states where data is involved.
- Run manual test, accessibility check and performance check.
- Submit deployed link, GitHub link and final screenshots.

CODE RUN EXAMPLE capstone/sprint-3.ts

```
// Capstone Sprint 3
export const sprint = {
name: 'Home and course catalog',
status: 'ready-for-review',
checklist: 4,
};

console.log('Capstone sprint ready:', sprint.name);
```

```
$ npm run build
```


Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Capstone Sprint 4: Course detail and lesson UI

English: This sprint converts all course knowledge into one complete frontend UI engineering product. Hinglish: Ab alag-alag topics ko ek real website/app me combine karna hai.

SPRINT STEPS

- Build lesson list, video area, notes section and progress controls.
- Add accessible tabs/accordion.
- Add resource download/open states.
- Test mobile layout.

DELIVERABLES

- Create clear task checklist and GitHub issue/README section.
- Build the required UI screens with responsive layout and component reuse.
- Add loading, error, empty and success states where data is involved.
- Run manual test, accessibility check and performance check.
- Submit deployed link, GitHub link and final screenshots.

CODE RUN EXAMPLE capstone/sprint-4.ts

```
// Capstone Sprint 4
export const sprint = {
name: 'Course detail and lesson UI',
status: 'ready-for-review',
checklist: 4,
};

console.log('Capstone sprint ready:', sprint.name);
```

```
$ npm run build
```


Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Capstone Sprint 5: Auth and profile flow

English: This sprint converts all course knowledge into one complete frontend UI engineering product. Hinglish: Ab alag-alag topics ko ek real website/app me combine karna hai.

SPRINT STEPS

- Build login, register, OTP and profile screens.
- Add validation and inline errors.
- Protect dashboard route.
- Document auth UI assumptions.

DELIVERABLES

- Create clear task checklist and GitHub issue/README section.
- Build the required UI screens with responsive layout and component reuse.
- Add loading, error, empty and success states where data is involved.
- Run manual test, accessibility check and performance check.
- Submit deployed link, GitHub link and final screenshots.

CODE RUN EXAMPLE capstone/sprint-5.ts

```
// Capstone Sprint 5
export const sprint = {
name: 'Auth and profile flow',
status: 'ready-for-review',
checklist: 4,
};

console.log('Capstone sprint ready:', sprint.name);
```

```
$ npm run build
```


Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Capstone Sprint 6: Dashboard and progress

English: This sprint converts all course knowledge into one complete frontend UI engineering product. Hinglish: Ab alag-alag topics ko ek real website/app me combine karna hai.

SPRINT STEPS

- Build progress cards and activity timeline.
- Show certificate eligibility conditions.
- Add skeleton loading and error state.
- Improve data visualization.

DELIVERABLES

- Create clear task checklist and GitHub issue/README section.
- Build the required UI screens with responsive layout and component reuse.
- Add loading, error, empty and success states where data is involved.
- Run manual test, accessibility check and performance check.
- Submit deployed link, GitHub link and final screenshots.

CODE RUN EXAMPLE

capstone/sprint-6.ts

```
// Capstone Sprint 6
export const sprint = {
name: 'Dashboard and progress',
status: 'ready-for-review',
checklist: 4,
};

console.log('Capstone sprint ready:', sprint.name);
```

```
$ npm run build
```


Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Capstone Sprint 7: Accessibility and keyboard pass

English: This sprint converts all course knowledge into one complete frontend UI engineering product. Hinglish: Ab alag-alag topics ko ek real website/app me combine karna hai.

SPRINT STEPS

- Test headings and landmarks.
- Test focus order and visible focus.
- Test modal/dropdown/tabs.
- Write a11y report.

DELIVERABLES

- Create clear task checklist and GitHub issue/README section.
- Build the required UI screens with responsive layout and component reuse.
- Add loading, error, empty and success states where data is involved.
- Run manual test, accessibility check and performance check.
- Submit deployed link, GitHub link and final screenshots.

CODE RUN EXAMPLE capstone/sprint-7.ts

```
// Capstone Sprint 7
export const sprint = {
name: 'Accessibility and keyboard pass',
status: 'ready-for-review',
checklist: 4,
};

console.log('Capstone sprint ready:', sprint.name);
```

```
$ npm run build
```


Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Capstone Sprint 8: Testing and documentation

English: This sprint converts all course knowledge into one complete frontend UI engineering product. Hinglish: Ab alag-alag topics ko ek real website/app me combine karna hai.

SPRINT STEPS

- Add component tests for core UI.
- Add Playwright journey test.
- Add Storybook stories for components.
- Update README and screenshots.

DELIVERABLES

- Create clear task checklist and GitHub issue/README section.
- Build the required UI screens with responsive layout and component reuse.
- Add loading, error, empty and success states where data is involved.
- Run manual test, accessibility check and performance check.
- Submit deployed link, GitHub link and final screenshots.

CODE RUN EXAMPLE capstone/sprint-8.ts

```
// Capstone Sprint 8
export const sprint = {
name: 'Testing and documentation',
status: 'ready-for-review',
checklist: 4,
};

console.log('Capstone sprint ready:', sprint.name);
```

```
$ npm run build
```


Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Capstone Sprint 9: Performance and deployment

English: This sprint converts all course knowledge into one complete frontend UI engineering product. Hinglish: Ab alag-alag topics ko ek real website/app me combine karna hai.

SPRINT STEPS

- Optimize images and lazy load sections.
- Run production build.
- Deploy preview and final URL.
- Record Lighthouse improvement notes.

DELIVERABLES

- Create clear task checklist and GitHub issue/README section.
- Build the required UI screens with responsive layout and component reuse.
- Add loading, error, empty and success states where data is involved.
- Run manual test, accessibility check and performance check.
- Submit deployed link, GitHub link and final screenshots.

CODE RUN EXAMPLE capstone/sprint-9.ts

```
// Capstone Sprint 9
export const sprint = {
name: 'Performance and deployment',
status: 'ready-for-review',
checklist: 4,
};

console.log('Capstone sprint ready:', sprint.name);
```

```
$ npm run build
```


Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Capstone Sprint 10: Final presentation and viva

English: This sprint converts all course knowledge into one complete frontend UI engineering product. Hinglish: Ab alag-alag topics ko ek real website/app me combine karna hai.

SPRINT STEPS

- Prepare 5-minute demo flow.
- Explain architecture in English and Hinglish.
- Show GitHub commits and project screenshots.
- Answer teacher interview questions.

DELIVERABLES

- Create clear task checklist and GitHub issue/README section.
- Build the required UI screens with responsive layout and component reuse.
- Add loading, error, empty and success states where data is involved.
- Run manual test, accessibility check and performance check.
- Submit deployed link, GitHub link and final screenshots.

CODE RUN EXAMPLE

capstone/sprint-10.ts

```
// Capstone Sprint 10
export const sprint = {
name: 'Final presentation and viva',
status: 'ready-for-review',
checklist: 4,
};

console.log('Capstone sprint ready:', sprint.name);
```

```
$ npm run build
```


Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Exam - HTML CSS UI

English: Use this page as a final readiness check. Hinglish: Final submit karne se pehle is checklist ko honestly verify karo.

CHECKLIST A

- Explain semantic HTML and landmark structure.
- Build accessible navigation with active state.
- Create responsive card grid using CSS Grid.
- Explain flexbox alignment with example.
- Fix a text overlap issue.

CHECKLIST B

- Create theme tokens using CSS variables.
- Explain how image alt text works.
- Build a contact form with labels.
- Explain specificity and cascade.
- Show mobile screenshot proof.

FINAL TEACHER GUIDANCE

- Never submit only screenshots. Submit source code, README, run command and deployed URL.
- Never hide poor alignment with excessive decoration. Clean spacing and readable text are more important.
- Every major UI must have mobile, tablet and desktop proof.
- Explain your project in English, then repeat simply in Hinglish during viva.
- Portfolio should show problem, solution, tech stack, screenshots and learning outcome.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Exam - JavaScript TypeScript React

English: Use this page as a final readiness check. Hinglish: Final submit karne se pehle is checklist ko honestly verify karo.

CHECKLIST A

- Build interactive tabs using DOM or React state.
- Fetch data and show loading/error/success states.
- Type a Course object with TypeScript.
- Build reusable CourseCard component.
- Explain useEffect cleanup.

CHECKLIST B

- Validate a form and show inline errors.
- Prevent unnecessary rerenders.
- Use props and children correctly.
- Explain controlled inputs.
- Show code output in browser.

FINAL TEACHER GUIDANCE

- Never submit only screenshots. Submit source code, README, run command and deployed URL.
- Never hide poor alignment with excessive decoration. Clean spacing and readable text are more important.
- Every major UI must have mobile, tablet and desktop proof.
- Explain your project in English, then repeat simply in Hinglish during viva.
- Portfolio should show problem, solution, tech stack, screenshots and learning outcome.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Exam - Next.js Integration

English: Use this page as a final readiness check. Hinglish: Final submit karne se pehle is checklist ko honestly verify karo.

CHECKLIST A

- Explain App Router page and layout files.
- Use Link for navigation.
- Use Image with width/height and alt.
- Create metadata for SEO.
- Build API helper file.

CHECKLIST B

- Handle env variable safely.
- Build protected UI flow.
- Explain server vs client component choice.
- Deploy on Vercel/Netlify.
- Show final URL.

FINAL TEACHER GUIDANCE

- Never submit only screenshots. Submit source code, README, run command and deployed URL.
- Never hide poor alignment with excessive decoration. Clean spacing and readable text are more important.
- Every major UI must have mobile, tablet and desktop proof.
- Explain your project in English, then repeat simply in Hinglish during viva.
- Portfolio should show problem, solution, tech stack, screenshots and learning outcome.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Final Exam - Accessibility Testing Performance

English: Use this page as a final readiness check. Hinglish: Final submit karne se pehle is checklist ko honestly verify karo.

CHECKLIST A

- Show keyboard-only navigation.
- Explain focus trap for modal.
- Run Lighthouse and explain one performance issue.
- Write Playwright test for course flow.
- Write Storybook story for button states.

CHECKLIST B

- Explain empty/loading/error states.
- Fix color contrast issue.
- Explain XSS prevention.
- Optimize one image.
- Submit report.

FINAL TEACHER GUIDANCE

- Never submit only screenshots. Submit source code, README, run command and deployed URL.
- Never hide poor alignment with excessive decoration. Clean spacing and readable text are more important.
- Every major UI must have mobile, tablet and desktop proof.
- Explain your project in English, then repeat simply in Hinglish during viva.
- Portfolio should show problem, solution, tech stack, screenshots and learning outcome.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Interview Question Bank

English: Use this page as a final readiness check. Hinglish: Final submit karne se pehle is checklist ko honestly verify karo.

CHECKLIST A

- What happens when a browser loads a webpage?
- Difference between button and anchor link?
- How do flexbox and grid differ?
- What is semantic HTML?
- What is hydration in React/Next?

CHECKLIST B

- How do you debug layout overflow?
- How do you make a modal accessible?
- How do you improve frontend performance?
- How do you structure a design system?
- How do you explain a project in interview?

FINAL TEACHER GUIDANCE

- Never submit only screenshots. Submit source code, README, run command and deployed URL.
- Never hide poor alignment with excessive decoration. Clean spacing and readable text are more important.
- Every major UI must have mobile, tablet and desktop proof.
- Explain your project in English, then repeat simply in Hinglish during viva.
- Portfolio should show problem, solution, tech stack, screenshots and learning outcome.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Portfolio and Resume Checklist

English: Use this page as a final readiness check. Hinglish: Final submit karne se pehle is checklist ko honestly verify karo.

CHECKLIST A

- Portfolio has 4-6 strong projects.
- Each project has problem, solution, tech stack and screenshots.
- GitHub repos have clean README files.
- Live links work without broken routes.
- Resume lists skills with project proof.

CHECKLIST B

- LinkedIn/GitHub profile is consistent.
- Capstone is pinned on GitHub.
- Screenshots are professional and aligned.
- No copied code without understanding.
- Teacher/career mentor can verify work.

FINAL TEACHER GUIDANCE

- Never submit only screenshots. Submit source code, README, run command and deployed URL.
- Never hide poor alignment with excessive decoration. Clean spacing and readable text are more important.
- Every major UI must have mobile, tablet and desktop proof.
- Explain your project in English, then repeat simply in Hinglish during viva.
- Portfolio should show problem, solution, tech stack, screenshots and learning outcome.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.

Certificate Readiness and Next Steps

English: Use this page as a final readiness check. Hinglish: Final submit karne se pehle is checklist ko honestly verify karo.

CHECKLIST A

- Complete final capstone.
- Submit GitHub and deployed URL.
- Submit all project screenshots.
- Pass final viva.
- Prepare interview answers.

CHECKLIST B

- Practice one UI challenge weekly.
- Keep learning browser APIs and new framework patterns.
- Build real client-style pages.
- Help juniors to strengthen own basics.
- Continue upgrading portfolio every month.

FINAL TEACHER GUIDANCE

- Never submit only screenshots. Submit source code, README, run command and deployed URL.
- Never hide poor alignment with excessive decoration. Clean spacing and readable text are more important.
- Every major UI must have mobile, tablet and desktop proof.
- Explain your project in English, then repeat simply in Hinglish during viva.
- Portfolio should show problem, solution, tech stack, screenshots and learning outcome.

Copyright (c) 2026 AiWorldAcademy. All rights reserved. For student learning and classroom use only.