Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: FDCP • **Updated:** 19 Nov 2026
https://mockattempt.com/training/frontend-developer-career-program

FDCP • FRONTEND REACT, TYPESCRIPT & ANGULAR ENGINEERING **BEGINNER TO JOB-READY**

100% PLACEMENT GUARANTEED* CAREER PROGRAM (6-MO REFUND*)

Frontend Developer – Industry-Ready Career Program

Go from semantic HTML5, CSS3 & modern JavaScript ES6+ to building, testing, optimizing, deploying, and defending production-ready React.js (primary specialization) and Enterprise Angular applications across 24 weekly modules, 6 mandatory projects, and 100 JS DSA + Machine-Coding challenges.

Duration: 24 Weeks (6 Months) • Beginner to Job-Ready (600 Hrs) • **Modules:** 24 • **Live Topics:** 527 • **Subtopics:** 4,216+ • **Labs & Tasks:** 1054 • **Mocks / AI Vivas:** 527 / 527 • **Projects:** 24 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹15,000 (Net Tuition ₹12,712 + Included 18% GST ₹2,288) • **Split Option:** 3 × ₹5,000 (0% Installments) • **Effective ROI:** ~₹25/Guided Hr (~₹28/Live Topic)

Faculty: 100% Active Working IT Professionals • **Weekend-Only Live Batch (7:00 AM – 8:00 PM IST Flexible Slots):** Weekend-Only Career Live Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday Only (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Weekend Slot Based on Working Professional Trainer Availability) • 28 Weeks (600 Guided Live + Lab Hours))

✓ **Trainer LeetCode & HackerRank Mentorship (JavaScript ES6+, TypeScript 5 & React): Live trainer support to crack LeetCode & HackerRank problems, optimize complexity, and handle edge cases.**

✓ **6-Month Placement Guarantee* + Online AI Assessment (OA) Support: Complete 100% assigned repositories & score ≥90% across all Topic Mock Tests & Voice AI Interviews → 100+ verified IT job openings within 6 months with active Senior Trainer coaching to crack employer Online AI Assessments (OA), or 100% net tuition refund*.**

Core Technologies & Engineering Stack: HTML5 & Semantic Accessibility • CSS3, Flexbox & CSS Grid • Responsive Mobile-First UI • Sass / SCSS & Bootstrap 5 • Tailwind CSS • JavaScript ES6+ & Async/DOM • TypeScript 5 (Strict Mode) • React.js (Primary Specialization) • React Router & Axios • React Hook Form & Zod • Redux Toolkit & RTK Query • TanStack Query & Zustand • Next.js App Router Fundamentals • Angular (Enterprise Framework) • RxJS, Signals & Reactive Forms • Angular Material & NgRx • Vitest & React Testing Library • Playwright E2E Fundamentals • Core Web Vitals & Lighthouse • WCAG 2.2 AA & Technical SEO • Frontend Security (XSS/CSRF/CSP) • Git, GitHub PRs & Actions CI/CD • Vite, npm & Figma Handoff • Vercel & Netlify Deployment

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. HTML5, CSS3 & Responsive UI: Build pixel-accurate, semantic, mobile-first layouts (`320px–1440px+`) using Flexbox, CSS Grid, Sass/SCSS, Bootstrap 5, and Tailwind CSS.
2. JavaScript ES6+ Mastery: Command execution contexts, closures, `this`/prototypes, the Event Loop, Promises, `async/await`, `fetch` + `AbortController`, Web Storage, and DOM event delegation.
3. TypeScript 5 Rigor: Model strict interfaces, discriminated unions, type narrowing, generics, and utility types (`Partial`, `Pick`, `Omit`, `Record`) with zero `any` shortcuts.
4. React.js Primary Specialization: Architect feature-sliced React SPAs using Hooks, Custom Hooks, React Router, React Hook Form + Zod, Redux Toolkit, TanStack Query, Compound Components, Code Splitting, and Next.js App Router fundamentals.
5. Enterprise Angular Competency: Build structured Standalone Angular applications using Signals, Hierarchical DI, Router Guards, `HttpClient` Interceptors, Typed Reactive Forms (`FormArray`), RxJS (`switchMap`, `combineLatest`), NgRx, `OnPush`, and Angular Material.
6. Security, Core Web Vitals, A11y & SEO: Prevent XSS/CSRF/open-redirects, tune LCP/INP/CLS (`Lighthouse >= 90`), enforce WCAG 2.2 AA keyboard/ARIA accessibility, and configure OpenGraph/JSON-LD technical SEO.
7. Testing, Git, CI/CD & Machine Coding: Write Vitest, React Testing Library, Angular `TestBed`, and Playwright E2E tests, collaborate via GitHub PRs + Actions CI, deploy live on Vercel/Netlify, and solve 100 JS DSA & UI Machine-Coding interview challenges.
8. Crack LeetCode and HackerRank problems in JavaScript ES6+, TypeScript 5 & React with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
9. Receive active Senior Trainer coaching during your 6-month (180-day) Placement Tenure to crack employer Online AI Assessments (OA), timed coding rounds, and automated screening tests.

10. Automated Testing, Debugging & Quality Gates (HTML5 & Semantic Accessibility, CSS3, Flexbox & CSS Grid, Responsive Mobile-First UI, Sass / SCSS & Bootstrap 5, Tailwind CSS) — Automated HTML5 & Semantic Accessibility, CSS3, Flexbox & CSS Grid, Responsive Mobile-First UI, Sass / SCSS & Bootstrap 5, Tailwind CSS test suites • Linting & static checks • Mentor Git pull-request reviews — ensuring your code never breaks silently when real users hit edge cases.
11. 100% Topic-Wise Timed Mock Tests, Spoken MockAttempt AI Viva & Live Client Production — Verify every single Frontend Developer – Industry-Ready Career Program topic through timed MCQ exams and spoken technical interviews before deploying your Live Client Project.
12. Translated live client functional and non-functional requirements into modular HTML5 & Semantic Accessibility, CSS3, Flexbox & CSS Grid, Responsive Mobile-First UI, Sass / SCSS & Bootstrap 5, Tailwind CSS production architecture.

Eligibility & Prerequisites: No prior HTML, CSS, JavaScript, React, or Angular experience required — Week 1 starts from web fundamentals and semantic HTML5. • A Windows, macOS, or Linux laptop (8 GB+ RAM recommended) with modern Chrome/Edge/Firefox and VS Code. • Consistent 25 hours/week dedication across live sessions, responsive UI builds, GitHub commits, and JS DSA / machine-coding practice. • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend Weekend-Only live classes (Saturday & Sunday, 2 Hours/Day between 7:00 AM – 8:00 PM IST taught by Working IT Professionals), complete weekday real-repository checklists, and practice topic-wise Mock Tests & AI Interviews.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (24 MODULES • 527 LIVE TOPICS • 4,216+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Month 1 – HTML5, CSS3 and Responsive Web Design (Weeks 1–4) (81 Topics / 648 Subtopics) • Modules 5–8: Month 2 – Complete JavaScript and Advanced JavaScript (Weeks 5–8) (95 Topics / 760 Subtopics) • Modules 9–12: Month 3 – TypeScript and React.js Fundamentals (Weeks 9–12) (93 Topics / 744 Subtopics) • Modules 13–16: Month 4 – Advanced React and Frontend Engineering (Weeks 13–16) (86 Topics / 688 Subtopics) • Modules 17–20: Month 5 – Angular and Enterprise Frontend Development (Weeks 17–20) (92 Topics / 736 Subtopics) • Modules 21–24: Month 6 – Industry Practices, Testing, Deployment and Placement Preparation (Weeks 21–24) (80 Topics / 640 Subtopics)

MODULE 1 OF 24 • STAGE 1 • 25 LIVE + LAB HOURS • 20 TOPICS (160 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Web Development Fundamentals and Complete HTML5

Master how browsers, DNS, HTTP/HTTPS, DOM/CSSOM rendering, semantic HTML5 landmarks, accessible tables, native validated forms, multimedia, HTML5 data APIs, and technical SEO metadata work together.

Objectives: ✓ Explain client-server architecture, DNS lookup, HTTP/HTTPS request-response headers/status codes, and the browser rendering pipeline. • ✓ Author valid, semantic HTML5 documents using landmarks (`header`, `nav`, `main`, `section`, `article`, `aside`, `footer`) instead of `div` soup. • ✓ Build accessible data tables (`caption`, `thead`, `th scope`) and rich HTML5 forms with explicit `<label for>` associations and native validation. • ✓ Embed responsive images, audio/video media, and custom `data-*` attributes cleanly. • ✓ Implement foundational technical SEO metadata (`<title>`, `<meta name="description">`, heading hierarchy) across 6 practical HTML pages.

1.1 Introduction to Web Development [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Frontend vs Backend vs Full Stack responsibilities • Client-server request lifecycle • Static sites vs dynamic SPAs/SSR apps • UI, Accessibility & Edge-Case Pitfall Guard (Introduction to Web Development) • Hands-On Frontend Implementation & Verification (Introduction to Web Development) • Introduction to Web Development — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Introduction to Web Development

Purpose: Understanding exact API contract boundaries between a React/Angular frontend and a REST backend.

What Topic Is Next → Next Topic 1.2: Internet and Web Fundamentals

1.2 Internet and Web Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Internet vs WWW • IP addresses • DNS resolution • Domain registrars • UI, Accessibility & Edge-Case Pitfall Guard (Internet and Web Fundamentals) • Hands-On Frontend Implementation & Verification (Internet and Web Fundamentals) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Internet

Purpose: Configuring custom domains, DNS `A`/`CNAME` records, and CDN edge caching when deploying frontend apps.

What Topic Is Next → Next Topic 1.3: HTTP and HTTPS Fundamentals

1.3 HTTP and HTTPS Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: HTTP methods (GET • POST • PUT • PATCH • UI, Accessibility & Edge-Case Pitfall Guard (HTTP and HTTPS Fundamentals) • Hands-On Frontend Implementation & Verification (HTTP and HTTPS Fundamentals) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP

Purpose: Debugging CORS `OPTIONS` preflights, `401` expired tokens, `422` validation errors, and `304 Not Modified` cache hits.

What Topic Is Next → Next Topic 1.4: Browser Rendering Architecture (DOM and CSSOM)

1.4 Browser Rendering Architecture (DOM and CSSOM) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: HTML parser -> DOM tree • CSS parser -> CSSOM tree • Render Tree construction • Layout (Reflow) • UI, Accessibility & Edge-Case Pitfall Guard (Browser Rendering Architecture (DOM and CSSOM)) • Hands-On Frontend Implementation & Verification (Browser Rendering Architecture (DOM and CSSOM)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Browser Rendering Architecture (DOM and CSSOM)

Purpose: Placing `<script type="module" src="...">` (deferred by default) so JavaScript never blocks first contentful paint.

What Topic Is Next → Next Topic 1.5: HTML5 Introduction and Document Declaration

1.5 HTML5 Introduction and Document Declaration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<!DOCTYPE html>` standards mode vs quirks mode • Evolution to living HTML5 standard • W3C markup validation • UI, Accessibility & Edge-Case Pitfall Guard (HTML5 Introduction and Document Declaration) • Hands-On Frontend Implementation & Verification (HTML5 Introduction and Document Declaration) • HTML5 Introduction and Document Declaration — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTML5 Introduction

Purpose: Ensuring `<!DOCTYPE html>` is line 1 of every document so browsers never fall back to broken legacy Quirks Mode box sizing.

What Topic Is Next → Next Topic 1.6: HTML Document Structure and Head Metadata

1.6 HTML Document Structure and Head Metadata [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<html lang="en">` • `<meta charset="UTF-8">` • `<meta name="viewport" content="width=device-width • Initial-scale=1.0">` • UI, Accessibility & Edge-Case Pitfall Guard (HTML Document Structure and Head Metadata) • Hands-On Frontend Implementation & Verification (HTML Document Structure and Head Metadata) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTML Document Structure

Purpose: Setting `lang="en"` for screen-reader pronunciation and `viewport` meta so mobile browsers don’t shrink pages to 980px desktop width!

What Topic Is Next → Next Topic 1.7: HTML Elements (Block, Inline and Void Elements)

1.7 HTML Elements (Block, Inline and Void Elements) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Block-level flow (`p` • `section` • `ul`) vs phrasing inline (`span` • `a` • UI, Accessibility & Edge-Case Pitfall Guard (HTML Elements (Block, Inline and Void Elements)) • Hands-On Frontend Implementation & Verification (HTML Elements (Block, Inline and Void Elements)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTML Elements (Block, Inline and Void Elements)

Purpose: Preventing invalid DOM auto-correction hydration mismatches in React/Next.js (e.g., never nesting `<div>` inside `<p>` or `<button>` inside `<a>`!).

What Topic Is Next → Next Topic 1.8: Text Formatting and Semantic Phrasing

1.8 Text Formatting and Semantic Phrasing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Heading hierarchy `<h1>`–`<h6>` (one logical `<h1>` per view) • `<p>` • `<strong>` vs `<b>` • `<em>` vs `<i>` • UI, Accessibility & Edge-Case Pitfall Guard (Text Formatting and Semantic Phrasing) • Hands-On Frontend Implementation & Verification (Text Formatting and Semantic Phrasing) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Text Formatting

Purpose: Using `<time datetime="2026-11-17">` and `<h1>`–`<h3>` logical outlines for accessibility and search-engine indexing.

What Topic Is Next → Next Topic 1.9: Links and Navigation Anchors

1.9 Links and Navigation Anchors [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<a href>` • Relative vs absolute URLs • In-page fragment skip links (`#main-content`) • `mailto:` / `tel:` • UI, Accessibility & Edge-Case Pitfall Guard (Links and Navigation Anchors) • Hands-On Frontend Implementation & Verification (Links and Navigation Anchors) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Links

Purpose: Adding a keyboard "Skip to main content" link at the top of every layout and descriptive anchor labels.

What Topic Is Next → Next Topic 1.10: Images and Responsive Image Attributes

1.10 Images and Responsive Image Attributes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<img src alt width height>` • Meaningful `alt` vs decorative `alt=""` • Preventing CLS layout shift with explicit `width`/`height` • `loading="lazy"` • UI, Accessibility & Edge-Case Pitfall Guard (Images and Responsive Image Attributes) • Hands-On Frontend Implementation & Verification (Images and Responsive Image Attributes) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Images

Purpose: Eliminating Cumulative Layout Shift (CLS) by reserving aspect ratio with `width` and `height` and lazy-loading below-the-fold product cards.

What Topic Is Next → Next Topic 1.11: Lists (Ordered, Unordered and Description Lists)

1.11 Lists (Ordered, Unordered and Description Lists) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<ul>` (unordered navigation/features) • `<ol>` (`start` • `reversed` step sequences) • `<dl>` / `<dt>` / `<dd>` key-value specification pairs • UI, Accessibility & Edge-Case Pitfall Guard (Lists (Ordered, Unordered and Description Lists)) • Hands-On Frontend Implementation & Verification (Lists (Ordered, Unordered and Description Lists)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Lists (Ordered, Unordered and Description Lists)

Purpose: Marking up navigation menus (`<nav><ul>`), checkout steps (`<ol>`), and product tech specs (`<dl>`) semantically.

What Topic Is Next → Next Topic 1.12: Accessible HTML Tables

1.12 Accessible HTML Tables [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<table>` • `<caption>` • `<thead>` • `<tbody>` • UI, Accessibility & Edge-Case Pitfall Guard (Accessible HTML Tables) • Hands-On Frontend Implementation & Verification (Accessible HTML Tables) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Accessible HTML Tables

Purpose: Building screen-reader-accessible Employee Information and Invoice Line-Item data tables.

What Topic Is Next → Next Topic 1.13: Semantic HTML5 Landmarks

1.13 Semantic HTML5 Landmarks [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<header>` • `<nav aria-label="...">` • `<main>` (one visible per page) • `<section>` • UI, Accessibility & Edge-Case Pitfall Guard (Semantic HTML5 Landmarks) • Hands-On Frontend Implementation & Verification (Semantic HTML5 Landmarks) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Semantic HTML5 Landmarks

Purpose: Allowing assistive-technology users to jump directly to `main` or `Primary navigation` landmarks.

What Topic Is Next → Next Topic 1.14: HTML5 Forms Architecture

1.14 HTML5 Forms Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<form action method>` • Explicit `<label for="id">` • `<fieldset>` + `<legend>` for radio/checkbox groups • `<textarea>` • UI, Accessibility & Edge-Case Pitfall Guard (HTML5 Forms Architecture) • Hands-On Frontend Implementation & Verification (HTML5 Forms Architecture) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTML5 Forms Architecture

Purpose: Building accessible Login, Registration, and Checkout forms where clicking any label focuses its control and `Enter` submits naturally.

What Topic Is Next → Next Topic 1.15: HTML5 Input Types

1.15 HTML5 Input Types [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `text` • `email` • `password` • `tel` (`inputmode="numeric"`) • UI, Accessibility & Edge-Case Pitfall Guard (HTML5 Input Types) • Hands-On Frontend Implementation & Verification (HTML5 Input Types) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTML5 Input Types

Purpose: Triggering the optimal mobile virtual keyboard (`type="email"`, `type="tel"`, `inputmode="numeric"` for OTPs) automatically on iOS and Android.

What Topic Is Next → Next Topic 1.16: Form Attributes (required, pattern, minlength, maxlength, autocomplete)

1.16 Form Attributes (required, pattern, minlength, maxlength, autocomplete) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `required` • Regex `pattern` + `title` hint • `minlength`/`maxlength` • `readonly` vs `disabled` (`disabled` fields are NOT submitted!) • UI, Accessibility & Edge-Case Pitfall Guard (Form Attributes (required, pattern, minlength, maxlength, autocomplete)) • Hands-On Frontend Implementation & Verification (Form Attributes (required, pattern, minlength, maxlength, autocomplete)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Form Attributes (required, pattern, minlength, maxle...

Purpose: Enabling password managers and browser autofill seamlessly via `autocomplete="current-password"` and `autocomplete="new-password"`.

What Topic Is Next → Next Topic 1.17: Native HTML Constraint Validation

1.17 Native HTML Constraint Validation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Browser constraint validation lifecycle • `:valid` / `:invalid` / `:user-invalid` CSS hooks • `checkValidity()` • `reportValidity()` • UI, Accessibility & Edge-Case Pitfall Guard (Native HTML Constraint Validation) • Hands-On Frontend Implementation & Verification (Native HTML Constraint Validation) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Native HTML Constraint Validation

Purpose: Leveraging native HTML constraints while understanding why client validation improves UX whereas backend validation enforces security.

What Topic Is Next → Next Topic 1.18: Multimedia Elements (Audio, Video and Iframe)

1.18 Multimedia Elements (Audio, Video and Iframe) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<video controls preload="metadata" poster playsinline>` • `<track kind="captions">` accessibility subtitles • `<audio>` • `<iframe sandbox loading="lazy" title="...">` • UI, Accessibility & Edge-Case Pitfall Guard (Multimedia Elements (Audio, Video and Iframe)) • Hands-On Frontend Implementation & Verification (Multimedia Elements (Audio, Video and Iframe)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Multimedia Elements (Audio, Video and Iframe)

Purpose: Embedding product demo videos with captions and sandboxed map/video iframes with explicit `title` attributes for screen readers.

What Topic Is Next → Next Topic 1.19: HTML5 Native APIs Overview (dataset, contenteditable, Drag & Drop, dialog)

1.19 HTML5 Native APIs Overview (dataset, contenteditable, Drag & Drop, dialog) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Custom `data-*` attributes (`el.dataset`) • `<dialog>` native accessible modal • `<details>`/`<summary>` disclosure • `contenteditable` • UI, Accessibility & Edge-Case Pitfall Guard (HTML5 Native APIs Overview (dataset, contenteditable, Drag & Drop, dialog)) • Hands-On Frontend Implementation & Verification (HTML5 Native APIs Overview (dataset, contenteditable, Drag & Drop, dialog)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTML5 Native APIs Overview (dataset, contenteditable...

Purpose: Using native `<details>/<summary>` for zero-JS FAQs and `<dialog>` for accessible top-layer modals.

What Topic Is Next → Next Topic 1.20: Frontend SEO Fundamentals in HTML

1.20 Frontend SEO Fundamentals in HTML [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unique `<title>` (50–60 chars) • `<meta name="description">` (120–158 chars) • `<link rel="canonical">` • Logical `h1`->`h2`->`h3` hierarchy • UI, Accessibility & Edge-Case Pitfall Guard (Frontend SEO Fundamentals in HTML) • Hands-On Frontend Implementation & Verification (Frontend SEO Fundamentals in HTML) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Frontend SEO Fundamentals in HTML

Purpose: Ensuring public marketing, product, and category pages rank cleanly in Google Search and render crisp social previews.

What Topic Is Next → Module 1 Capstone (Week 1 Lab Build: Semantic HTML5 Multi-Page Portfolio & Accessible Form Pack) → Next Topic 2.1: CSS Introduction (External, Internal and Inline)

Module 1 Hands-On Capstone Project (100 Marks): Week 1 Lab Build: Semantic HTML5 Multi-Page Portfolio & Accessible Form Pack — Build 6 semantic, W3C-valid HTML5 pages (Personal Portfolio, Registration Form, Login Form, Employee Data Table, Product Listing Skeleton, and Multi-Section Business Page) with landmarks, ex...

MODULE 2 OF 24 • STAGE 1 • 25 LIVE + LAB HOURS • 23 TOPICS (184 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Complete CSS3 Fundamentals

Master CSS3 styling from cascade, specificity, units, colors, typography, and the Box Model through positioning, stacking contexts (`z-index`), CSS custom properties, smooth GPU transitions, transforms, and `@keyframes` animations.

Objectives: ✓ Calculate selector specificity `(id, class/pseudo-class/attr, element)` accurately without resorting to `!important`. • ✓ Apply relative scalable CSS units (`rem`, `em`, `%`, `ch`, `vh`/`dvh`, `vw`) and accessible HSL/HEX/RGB color palettes. • ✓ Control layout dimensions predictably using `box-sizing: border-box`, display modes, positioning, overflow, and `z-index` stacking contexts. • ✓ Define reusable design tokens (`--color-*`, `--space-*`) using CSS Custom Properties. • ✓ Create smooth 60fps UI micro-interactions animating `transform` and `opacity` rather than layout-thrashing properties.

2.1 CSS Introduction (External, Internal and Inline) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: External `<link rel="stylesheet">` caching benefits • `<style>` critical CSS • Why inline `style=""` hurts maintainability & CSP • UI, Accessibility & Edge-Case Pitfall Guard (CSS Introduction (External, Internal and Inline)) • Hands-On Frontend Implementation & Verification (CSS Introduction (External, Internal and Inline)) • CSS Introduction (External, Internal and Inline) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Introduction (External, Internal and Inline)

Purpose: Serving cacheable external stylesheets shared across multi-page websites.

What Topic Is Next → Next Topic 2.2: CSS Syntax, Rulesets and Declarations

2.2 CSS Syntax, Rulesets and Declarations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Selector • Declaration block `{ property: value • Shorthand vs longhand properties • Browser forgiving parsing of unknown properties • UI, Accessibility & Edge-Case Pitfall Guard (CSS Syntax, Rulesets and Declarations) • Hands-On Frontend Implementation & Verification (CSS Syntax, Rulesets and Declarations) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Syntax

Purpose: Understanding shorthand resets (`margin: 0 auto` sets top/bottom to `0` AND overrides prior `margin-top`!).

What Topic Is Next → Next Topic 2.3: Basic CSS Selectors (Element, Class, ID, Universal)

2.3 Basic CSS Selectors (Element, Class, ID, Universal) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Type `button` • Reusable `.btn-primary` classes • `#id` hooks • Universal `*` • UI, Accessibility & Edge-Case Pitfall Guard (Basic CSS Selectors (Element, Class, ID, Universal)) • Hands-On Frontend Implementation & Verification (Basic CSS Selectors (Element, Class, ID, Universal)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Basic CSS Selectors (Element, Class, ID, Universal)

Purpose: Styling 99% of UI components with low-specificity reusable `.class` selectors.

What Topic Is Next → Next Topic 2.4: CSS Combinators (Descendant, Child, Adjacent & General Sibling)

2.4 CSS Combinators (Descendant, Child, Adjacent & General Sibling) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Descendant (`A B`) • Direct child (`A > B`) • Next-sibling (`A + B`) • Subsequent-sibling (`A ~ B`) • UI, Accessibility & Edge-Case Pitfall Guard (CSS Combinators (Descendant, Child, Adjacent & General Sibling)) • Hands-On Frontend Implementation & Verification (CSS Combinators (Descendant, Child, Adjacent & General Sibling)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Combinators (Descendant, Child, Adjacent & Gener...

Purpose: Applying vertical rhythm spacing between consecutive paragraphs (`article > * + * { margin-top: 1rem; }`).

What Topic Is Next → Next Topic 2.5: Attribute Selectors

2.5 Attribute Selectors [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `[disabled]` • `[type="email"]` • `[href^="https"]` • `[href$=".pdf"]` • UI, Accessibility & Edge-Case Pitfall Guard (Attribute Selectors) • Hands-On Frontend Implementation & Verification (Attribute Selectors) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Attribute Selectors

Purpose: Tying visual open/active state directly to ARIA accessibility attributes (`button[aria-expanded="true"]`) so visual and accessibility states never drift!

What Topic Is Next → Next Topic 2.6: Pseudo-Classes (:hover, :focus-visible, :active, :nth-child, :is, :where, :has)

2.6 Pseudo-Classes (:hover, :focus-visible, :active, :nth-child, :is, :where, :has) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Interactive `:hover` • Keyboard-only `:focus-visible` (superior to `:focus`!) • Structural `:nth-child(even)` • Zero-specificity `:where()` • UI, Accessibility & Edge-Case Pitfall Guard (Pseudo-Classes (:hover, :focus-visible, :active, :nth-child, :is, :where, :has)) • Hands-On Frontend Implementation & Verification (Pseudo-Classes (:hover, :focus-visible, :active, :nth-child, :is, :where, :has)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pseudo-Classes (:hover, :focus-visible, :active, :nt...

Purpose: Highlighting a form `.field` card when its inner input is focused using `.field:has(input:focus-visible)`!

What Topic Is Next → Next Topic 2.7: Pseudo-Elements (::before, ::after, ::placeholder, ::selection)

2.7 Pseudo-Elements (::before, ::after, ::placeholder, ::selection) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Decorative `::before`/`::after` (`content: ""`) • Custom badges/dividers • `::first-letter` • `::placeholder` • UI, Accessibility & Edge-Case Pitfall Guard (Pseudo-Elements (::before, ::after, ::placeholder, ::selection)) • Hands-On Frontend Implementation & Verification (Pseudo-Elements (::before, ::after, ::placeholder, ::selection)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pseudo-Elements (::before, ::after, ::placeholder, :...

Purpose: Adding decorative accent bars, quote marks, and external-link icons without cluttering HTML DOM nodes.

What Topic Is Next → Next Topic 2.8: The Cascade, Inheritance and Specificity

2.8 The Cascade, Inheritance and Specificity [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Origin & layer order • `(Inline • Class/Attr/Pseudo-class • Element/Pseudo-element)` tuple scoring • UI, Accessibility & Edge-Case Pitfall Guard (The Cascade, Inheritance and Specificity) • Hands-On Frontend Implementation & Verification (The Cascade, Inheritance and Specificity) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for The Cascade

Purpose: Predicting which CSS rule wins instantaneously without sprinkling `!important` hacks across files.

What Topic Is Next → Next Topic 2.9: CSS Units (px, %, em, rem, vh, dvh, vw, ch)

2.9 CSS Units (px, %, em, rem, vh, dvh, vw, ch) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `rem` (root-relative — respects user browser font-size accessibility zoom!) • `em` (compound component scaling) • `ch` (`60ch–70ch` optimal reading measure!) • `100dvh` mobile dynamic viewport height • UI, Accessibility & Edge-Case Pitfall Guard (CSS Units (px, %, em, rem, vh, dvh, vw, ch)) • Hands-On Frontend Implementation & Verification (CSS Units (px, %, em, rem, vh, dvh, vw, ch)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Units (px, %, em, rem, vh, dvh, vw, ch)

Purpose: Using `rem` for font sizes/spacing, `px` for `1px` crisp borders, `65ch` for readable paragraphs, and `min-height: 100dvh` on mobile layouts.

What Topic Is Next → Next Topic 2.10: CSS Color Systems (HEX, RGB/RGBA, HSL and Modern Color Syntax)

2.10 CSS Color Systems (HEX, RGB/RGBA, HSL and Modern Color Syntax) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: HEX (`#0f172a`) • Modern space-separated `rgb(15 23 42 / 0.8)` • Intuitive `hsl(222 47% 11%)` hue/saturation/lightness palettes • WCAG 4.5:1 text contrast ratio • UI, Accessibility & Edge-Case Pitfall Guard (CSS Color Systems (HEX, RGB/RGBA, HSL and Modern Color Syntax)) • Hands-On Frontend Implementation & Verification (CSS Color Systems (HEX, RGB/RGBA, HSL and Modern Color Syntax)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Color Systems (HEX, RGB/RGBA, HSL and Modern Col...

Purpose: Generating harmonious hover/tint/shade design-token scales effortlessly by adjusting `HSL` lightness.

What Topic Is Next → Next Topic 2.11: Backgrounds and Linear/Radial Gradients

2.11 Backgrounds and Linear/Radial Gradients [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `background-color` • `background-image` • `linear-gradient` / `radial-gradient` • `background-size: cover | contain` • UI, Accessibility & Edge-Case Pitfall Guard (Backgrounds and Linear/Radial Gradients) • Hands-On Frontend Implementation & Verification (Backgrounds and Linear/Radial Gradients) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Backgrounds

Purpose: Ensuring white hero text remains readable even if a hero background image fails to load by setting a dark fallback `background-color` + gradient overlay!

What Topic Is Next → Next Topic 2.12: Web Typography (Font Stacks, @font-face, Weight & Line-Height)

2.12 Web Typography (Font Stacks, @font-face, Weight & Line-Height) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: System font stacks • Variable Google Fonts with `font-display: swap` • Unitless `line-height` (`1.5` body • `1.15–1.25` headings) • UI, Accessibility & Edge-Case Pitfall Guard (Web Typography (Font Stacks, @font-face, Weight & Line-Height)) • Hands-On Frontend Implementation & Verification (Web Typography (Font Stacks, @font-face, Weight & Line-Height)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Web Typography (Font Stacks, @font-face, Weight & Li...

Purpose: Using unitless `line-height: 1.5` (inherits as multiplier, not fixed pixel height!) and `font-display: swap` so text is immediately visible.

What Topic Is Next → Next Topic 2.13: Text Styling, Alignment and Truncation

2.13 Text Styling, Alignment and Truncation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `text-align` • `text-decoration` • `text-underline-offset` • `text-transform` • UI, Accessibility & Edge-Case Pitfall Guard (Text Styling, Alignment and Truncation) • Hands-On Frontend Implementation & Verification (Text Styling, Alignment and Truncation) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Text Styling

Purpose: Clamping variable-length product titles to 2 lines cleanly inside e-commerce cards.

What Topic Is Next → Next Topic 2.14: The CSS Box Model (Content, Padding, Border, Margin & box-sizing)

2.14 The CSS Box Model (Content, Padding, Border, Margin & box-sizing) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `content-box` (width excludes padding/border!) vs `border-box` (width includes padding & border!) • Vertical margin collapsing • Logical properties (`margin-inline` • `padding-block`) • UI, Accessibility & Edge-Case Pitfall Guard (The CSS Box Model (Content, Padding, Border, Margin & box-sizing)) • Hands-On Frontend Implementation & Verification (The CSS Box Model (Content, Padding, Border, Margin & box-sizing)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for The CSS Box Model (Content, Padding, Border, Margin...

Purpose: Setting `box-sizing: border-box` globally so `width: 50%; padding: 1rem;` never overflows `100%`!

What Topic Is Next → Next Topic 2.15: Display Property (block, inline, inline-block, none vs visibility)

2.15 Display Property (block, inline, inline-block, none vs visibility) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Width/height/vertical-margin behavior on `inline` vs `inline-block` vs `block` • `display: none` (removed from layout • Screen readers) vs `.sr-only` visually-hidden accessibility helper • UI, Accessibility & Edge-Case Pitfall Guard (Display Property (block, inline, inline-block, none vs visibility)) • Hands-On Frontend Implementation & Verification (Display Property (block, inline, inline-block, none vs visibility)) • Display Property (block, inline, inline-block, none vs visibility) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Display Property (block, inline, inline-block, none...

Purpose: Keeping an icon-only button accessible via a visually-hidden `.sr-only` label span rather than `display: none` (which hides text from screen readers too!).

What Topic Is Next → Next Topic 2.16: CSS Positioning (static, relative, absolute, fixed, sticky)

2.16 CSS Positioning (static, relative, absolute, fixed, sticky) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Normal flow `static` • Offset/containing-block anchor `relative` • Out-of-flow `absolute` (positions relative to nearest positioned ancestor!) • Viewport `fixed` • UI, Accessibility & Edge-Case Pitfall Guard (CSS Positioning (static, relative, absolute, fixed, sticky)) • Hands-On Frontend Implementation & Verification (CSS Positioning (static, relative, absolute, fixed, sticky)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Positioning (static, relative, absolute, fixed,...

Purpose: Pinning a sticky top Navigation Bar (`position: sticky; top: 0`) and placing a notification count badge (`position: absolute`) on a `position: relative` bell button.

What Topic Is Next → Next Topic 2.17: Overflow Control (visible, hidden, clip, scroll, auto)

2.17 Overflow Control (visible, hidden, clip, scroll, auto) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `overflow-x: auto` for responsive wide tables • `overflow: hidden` vs `overflow: clip` (doesn't break `position: sticky`!) • Preventing unintended horizontal page scrollbars • UI, Accessibility & Edge-Case Pitfall Guard (Overflow Control (visible, hidden, clip, scroll, auto)) • Hands-On Frontend Implementation & Verification (Overflow Control (visible, hidden, clip, scroll, auto)) • Overflow Control (visible, hidden, clip, scroll, auto) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Overflow Control (visible, hidden, clip, scroll, aut...

Purpose: Wrapping wide data tables in an `overflow-x: auto` container so 360px mobile screens scroll the table horizontally without breaking the page viewport.

What Topic Is Next → Next Topic 2.18: Z-Index and Stacking Contexts

2.18 Z-Index and Stacking Contexts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `z-index: 999999` still loses if trapped inside a parent with a lower stacking context! What creates a stacking context (`position`+`z-index` • `opacity < 1` • `transform` • `isolation: isolate`) • UI, Accessibility & Edge-Case Pitfall Guard (Z-Index and Stacking Contexts) • Hands-On Frontend Implementation & Verification (Z-Index and Stacking Contexts) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Z-Index

Purpose: Using `isolation: isolate` on cards and a clean 5-step `--z-dropdown`, `--z-sticky`, `--z-modal`, `--z-toast` scale.

What Topic Is Next → Next Topic 2.19: Borders, Border-Radius, Box-Shadow and Outlines

2.19 Borders, Border-Radius, Box-Shadow and Outlines [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `border-radius` (nested concentric radius `outerRadius = innerRadius + padding`!) • Layered subtle `box-shadow` elevation tokens • Why `outline` doesn't shift box layout • UI, Accessibility & Edge-Case Pitfall Guard (Borders, Border-Radius, Box-Shadow and Outlines) • Hands-On Frontend Implementation & Verification (Borders, Border-Radius, Box-Shadow and Outlines) • Borders, Border-Radius, Box-Shadow and Outlines — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Borders

Purpose: Designing crisp SaaS elevation cards with concentric inner image radii (`16px` outer card, `8px` padding -> `8px` inner image radius).

What Topic Is Next → Next Topic 2.20: CSS Variables (Custom Properties and Design Tokens)

2.20 CSS Variables (Custom Properties and Design Tokens) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `:root` global tokens vs component-scoped variables • `var(--token • Fallback)` • Runtime cascade updates (dark mode `[data-theme="dark"]` swaps tokens in 10 lines!) • UI, Accessibility & Edge-Case Pitfall Guard (CSS Variables (Custom Properties and Design Tokens)) • Hands-On Frontend Implementation & Verification (CSS Variables (Custom Properties and Design Tokens)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Variables (Custom Properties and Design Tokens)

Purpose: Building a complete semantic design-token layer (`--surface`, `--text-primary`, `--border`, `--accent`) enabling instant Light/Dark themes.

What Topic Is Next → Next Topic 2.21: CSS Transitions and Timing Functions

2.21 CSS Transitions and Timing Functions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `transition: transform 180ms cubic-bezier(0.2 • Background-color 150ms ease` • Why `transition: all` hurts performance & causes resize flashes! • UI, Accessibility & Edge-Case Pitfall Guard (CSS Transitions and Timing Functions) • Hands-On Frontend Implementation & Verification (CSS Transitions and Timing Functions) • CSS Transitions and Timing Functions — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Transitions

Purpose: Adding crisp 150–220ms button, link, and card hover feedback without sluggish animations.

What Topic Is Next → Next Topic 2.22: CSS Transforms (translate, scale, rotate, skew)

2.22 CSS Transforms (translate, scale, rotate, skew) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: GPU-composited `transform: translateY(-4px) scale(1.01)` • Modern standalone `translate`/`scale`/`rotate` properties • `transform-origin` • UI, Accessibility & Edge-Case Pitfall Guard (CSS Transforms (translate, scale, rotate, skew)) • Hands-On Frontend Implementation & Verification (CSS Transforms (translate, scale, rotate, skew)) • CSS Transforms (translate, scale, rotate, skew) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Transforms (translate, scale, rotate, skew)

Purpose: Lifting cards on hover via `transform: translateY(-3px)` at 60fps without pushing neighboring cards down!

What Topic Is Next → Next Topic 2.23: CSS @keyframes Animations & Reduced Motion Accessibility

2.23 CSS @keyframes Animations & Reduced Motion Accessibility [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@keyframes` • `animation` (`duration` • `iteration-count` • `fill-mode`) • UI, Accessibility & Edge-Case Pitfall Guard (CSS @keyframes Animations & Reduced Motion Accessibility) • Hands-On Frontend Implementation & Verification (CSS @keyframes Animations & Reduced Motion Accessibility) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS @keyframes Animations

Purpose: Building accessible skeleton loading shimmers that automatically respect vestibular `@media (prefers-reduced-motion: reduce)` user settings.

What Topic Is Next → Module 2 Capstone (Week 2 Lab Build: CSS3 Flexbox & Grid Responsive UI Component Gallery) → Next Topic 3.1: Flexbox Container and Flex Items

Module 2 Hands-On Capstone Project (100 Marks): Week 2 Lab Build: CSS3 Flexbox & Grid Responsive UI Component Gallery — Style a Pricing Table, Responsive Navbar, Profile Card, 3-Column Feature Section, Login Screen, and CSS Grid Dashboard Shell using custom properties, Flexbox, CSS Grid, and smooth `transform...

MODULE 3 OF 24 • STAGE 1 • 25 LIVE + LAB HOURS • 20 TOPICS (160 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Advanced CSS, Flexbox, Grid and Responsive Design

Architect complex 1D and 2D responsive web layouts using Flexbox, CSS Grid (`auto-fit` / `minmax` / `grid-template-areas`), mobile-first media queries, modern `@container` queries, fluid `clamp()` typography, responsive `srcset` images, BEM/modular CSS architecture, and Sass/SCSS.

Objectives: ✓ Master 1D component alignment, distribution, wrapping, and `flex: 1 1 basis` sizing with Flexbox. • ✓ Master 2D page and catalog grids using `grid-template-columns`, `repeat(auto-fit, minmax(...))`, line spans, and named `grid-template-areas`. • ✓ Build mobile-first responsive layouts (`min-width` breakpoints), `@container` component queries, and fluid `clamp()` scales. • ✓ Optimize responsive images (`<picture>`, `srcset`, `sizes`, `object-fit`, `aspect-ratio`) across 320px phones to 2560px monitors. • ✓ Structure maintainable stylesheets using BEM/modular conventions, Sass/SCSS (`@use`, mixins, nesting), and Chrome DevTools Grid/Flex inspectors.

3.1 Flexbox Container and Flex Items [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `display: flex` vs `inline-flex` • Main axis vs cross axis mental model • `gap` property (replaces messy margin hacks!) • UI, Accessibility & Edge-Case Pitfall Guard (Flexbox Container and Flex Items) • Hands-On Frontend Implementation & Verification (Flexbox Container and Flex Items) • Flexbox Container and Flex Items — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Flexbox Container

Purpose: Laying out navbars, button groups, badges, and card footers with uniform `gap` spacing.

What Topic Is Next → Next Topic 3.2: Flex Direction (row, column and reverse)

3.2 Flex Direction (row, column and reverse) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `flex-direction: row | column | row-reverse | column-reverse` • How switching to `column` swaps main • Cross axes! Accessibility tab-order warning on `*-reverse` • UI, Accessibility & Edge-Case Pitfall Guard (Flex Direction (row, column and reverse)) • Hands-On Frontend Implementation & Verification (Flex Direction (row, column and reverse)) • Flex Direction (row, column and reverse) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Flex Direction (row, column and reverse)

Purpose: Making equal-height cards push their CTA footer to the bottom via `display: flex; flex-direction: column;` + `margin-top: auto` on the footer!

What Topic Is Next → Next Topic 3.3: Flexbox Alignment (justify-content, align-items, align-self)

3.3 Flexbox Alignment (justify-content, align-items, align-self) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Main-axis `justify-content` (`flex-start` • `center` • `space-between`) • Cross-axis `align-items` (`stretch` • UI, Accessibility & Edge-Case Pitfall Guard (Flexbox Alignment (justify-content, align-items, align-self)) • Hands-On Frontend Implementation & Verification (Flexbox Alignment (justify-content, align-items, align-self)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Flexbox Alignment (justify-content, align-items, ali...

Purpose: Aligning currency symbol, large price digits, and `/month` label along typographic `align-items: baseline`.

What Topic Is Next → Next Topic 3.4: Flex Wrapping (flex-wrap and align-content)

3.4 Flex Wrapping (flex-wrap and align-content) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `flex-wrap: nowrap | wrap` • Multi-line row/column `gap` • `align-content` for multi-line flex tracks • UI, Accessibility & Edge-Case Pitfall Guard (Flex Wrapping (flex-wrap and align-content)) • Hands-On Frontend Implementation & Verification (Flex Wrapping (flex-wrap and align-content)) • Flex Wrapping (flex-wrap and align-content) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Flex Wrapping (flex-wrap and align-content)

Purpose: Rendering dynamic skill tags, filter chips, and metadata pills that wrap gracefully onto new lines on narrow screens.

What Topic Is Next → Next Topic 3.5: Flex Sizing (flex-grow, flex-shrink, flex-basis & min-width: 0)

3.5 Flex Sizing (flex-grow, flex-shrink, flex-basis & min-width: 0) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `flex: 1` (`1 1 0%`) • `flex: 0 0 auto` (`shrink-0` icons!) • The critical `min-width: 0` fix when truncated text inside a flex item refuses to shrink! • UI, Accessibility & Edge-Case Pitfall Guard (Flex Sizing (flex-grow, flex-shrink, flex-basis & min-width: 0)) • Hands-On Frontend Implementation & Verification (Flex Sizing (flex-grow, flex-shrink, flex-basis & min-width: 0)) • Flex Sizing (flex-grow, flex-shrink, flex-basis & min-width: 0) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Flex Sizing (flex-grow, flex-shrink, flex-basis & mi...

Purpose: Putting `flex-shrink: 0` on avatars/icons so they never squish into ovals, and `min-width: 0` on text wrappers so `text-overflow: ellipsis` works!

What Topic Is Next → Next Topic 3.6: CSS Grid Container and Grid Items

3.6 CSS Grid Container and Grid Items [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 2D layout control (rows • Columns simultaneously) vs 1D Flexbox • `display: grid` • Explicit vs implicit grid tracks • UI, Accessibility & Edge-Case Pitfall Guard (CSS Grid Container and Grid Items) • Hands-On Frontend Implementation & Verification (CSS Grid Container and Grid Items) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Grid Container

Purpose: Choosing CSS Grid for macro page shells, bento grids, and aligned multi-column product catalogs while using Flexbox inside individual cards.

What Topic Is Next → Next Topic 3.7: Grid Structure (Rows, Columns, fr Unit and Gaps)

3.7 Grid Structure (Rows, Columns, fr Unit and Gaps) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fractional free-space `1fr` unit • `minmax(0 • 1fr)` (prevents wide code/pre blocks from blowing out `1fr` tracks!) • `row-gap` / `column-gap` / `gap` • UI, Accessibility & Edge-Case Pitfall Guard (Grid Structure (Rows, Columns, fr Unit and Gaps)) • Hands-On Frontend Implementation & Verification (Grid Structure (Rows, Columns, fr Unit and Gaps)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Grid Structure (Rows, Columns, fr Unit and Gaps)

Purpose: Creating sidebar + main-content app shells (`grid-template-columns: 16rem minmax(0, 1fr)`).

What Topic Is Next → Next Topic 3.8: Grid Templates (grid-template-columns, rows and grid-auto-rows)

3.8 Grid Templates (grid-template-columns, rows and grid-auto-rows) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mixed fixed + flexible tracks • `grid-auto-rows: minmax(8rem • Auto)` • CSS `subgrid` awareness for aligning inner card headings/prices across sibling cards • UI, Accessibility & Edge-Case Pitfall Guard (Grid Templates (grid-template-columns, rows and grid-auto-rows)) • Hands-On Frontend Implementation & Verification (Grid Templates (grid-template-columns, rows and grid-auto-rows)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Grid Templates (grid-template-columns, rows and grid...

Purpose: Structuring KPI summary rows and multi-row bento feature sections.

What Topic Is Next → Next Topic 3.9: Grid Placement (Line Numbers, Spans and Named grid-template-areas)

3.9 Grid Placement (Line Numbers, Spans and Named grid-template-areas) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `grid-column: 1 / -1` full-bleed span • `span 2` • Visual ASCII-map `grid-template-areas` (`"header header" "sidebar main" "footer footer"`) • UI, Accessibility & Edge-Case Pitfall Guard (Grid Placement (Line Numbers, Spans and Named grid-template-areas)) • Hands-On Frontend Implementation & Verification (Grid Placement (Line Numbers, Spans and Named grid-template-areas)) • Grid Placement (Line Numbers, Spans and Named grid-template-areas) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Grid Placement (Line Numbers, Spans and Named grid-t...

Purpose: Reconfiguring an entire Admin Dashboard layout from single-column mobile to sidebar+topbar desktop purely by redefining `grid-template-areas` at `@media (min-width: 64rem...

What Topic Is Next → Next Topic 3.10: Intrinsic Grid Functions (repeat, minmax, auto-fit vs auto-fill)

3.10 Intrinsic Grid Functions (repeat, minmax, auto-fit vs auto-fill) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Zero-media-query responsive grid `repeat(auto-fit • Minmax(min(100% • 17rem) • 1fr))` • UI, Accessibility & Edge-Case Pitfall Guard (Intrinsic Grid Functions (repeat, minmax, auto-fit vs auto-fill)) • Hands-On Frontend Implementation & Verification (Intrinsic Grid Functions (repeat, minmax, auto-fit vs auto-fill)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Intrinsic Grid Functions (repeat, minmax, auto-fit v...

Purpose: Building self-adapting Product Grids that wrap from 4 -> 3 -> 2 -> 1 columns smoothly on every screen width (`min(100%, 17rem)` prevents overflow below `17rem`!).

What Topic Is Next → Next Topic 3.11: Responsive Design Philosophy (Mobile-First vs Desktop-First)

3.11 Responsive Design Philosophy (Mobile-First vs Desktop-First) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Base styles for narrow single-column mobile + progressive `@media (min-width: ...)` enhancements vs desktop-first `max-width` overrides • Touch-target minimum `44x44px` • UI, Accessibility & Edge-Case Pitfall Guard (Responsive Design Philosophy (Mobile-First vs Desktop-First)) • Hands-On Frontend Implementation & Verification (Responsive Design Philosophy (Mobile-First vs Desktop-First)) • Responsive Design Philosophy (Mobile-First vs Desktop-First) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Responsive Design Philosophy (Mobile-First vs Deskto... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Writing 30% cleaner CSS where mobile devices parse simple stacked base styles and larger viewports opt into multi-column grids.

What Topic Is Next → Next Topic 3.12: Media Queries and Content-Driven Breakpoints

3.12 Media Queries and Content-Driven Breakpoints [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Standard `rem`-based breakpoints (`40rem/640px` • `48rem/768px` • `64rem/1024px` • `80rem/1280px`) • UI, Accessibility & Edge-Case Pitfall Guard (Media Queries and Content-Driven Breakpoints) • Hands-On Frontend Implementation & Verification (Media Queries and Content-Driven Breakpoints) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Media Queries

Purpose: Using `rem` in `@media (min-width: 48rem)` so layouts reflow cleanly even when a user sets browser zoom/font-size to 150%!

What Topic Is Next → Next Topic 3.13: CSS Container Queries (@container)

3.13 CSS Container Queries (@container) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `container-type: inline-size` • `@container (min-width: 28rem)` • Making a reusable `<ProductCard>` adapt to its *parent slot width* (sidebar vs main grid) rather than global viewport width! • UI, Accessibility & Edge-Case Pitfall Guard (CSS Container Queries (@container)) • Hands-On Frontend Implementation & Verification (CSS Container Queries (@container)) • CSS Container Queries (@container) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Container Queries (@container)

Purpose: Rendering the exact same `<ProductCard>` vertically inside a narrow `300px` sidebar and horizontally inside a wide `700px` featured slot!

What Topic Is Next → Next Topic 3.14: Fluid Design Functions (clamp(), min(), max())

3.14 Fluid Design Functions (clamp(), min(), max()) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fluid typography `font-size: clamp(1.75rem • 1.2rem + 2.5vw • 3.25rem)` (always include `rem` in preferred `rem + vw` term so browser zoom still scales text!) • Fluid padding & container `width: min(100% - 2rem • UI, Accessibility & Edge-Case Pitfall Guard (Fluid Design Functions (clamp(), min(), max())) • Hands-On Frontend Implementation & Verification (Fluid Design Functions (clamp(), min(), max())) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Fluid Design Functions (clamp(), min(), max())

Purpose: Scaling headings and section padding smoothly between mobile and desktop without 4 stepped media-query jumps.

What Topic Is Next → Next Topic 3.15: Responsive Images (srcset, sizes, picture and object-fit)

3.15 Responsive Images (srcset, sizes, picture and object-fit) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Resolution switching (`srcset` `400w • 800w • 1200w` + `sizes`) • Art direction & modern AVIF/WebP fallback (`<picture>`) • UI, Accessibility & Edge-Case Pitfall Guard (Responsive Images (srcset, sizes, picture and object-fit)) • Hands-On Frontend Implementation & Verification (Responsive Images (srcset, sizes, picture and object-fit)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Responsive Images (srcset, sizes, picture and object...

Purpose: Saving 70% mobile bandwidth by serving a `400w` WebP thumbnail on phones instead of a `2400px` desktop banner.

What Topic Is Next → Next Topic 3.16: Responsive Navigation Patterns

3.16 Responsive Navigation Patterns [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Desktop horizontal nav -> mobile accessible menu toggle (`<button aria-expanded aria-controls>`) • Off-canvas drawer transitions • Escape-key & backdrop close readiness • UI, Accessibility & Edge-Case Pitfall Guard (Responsive Navigation Patterns) • Hands-On Frontend Implementation & Verification (Responsive Navigation Patterns) • Responsive Navigation Patterns — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Responsive Navigation Patterns

Purpose: Building a clean, touch-friendly Responsive Navigation Menu that works equally well with mouse, touch, and keyboard.

What Topic Is Next → Next Topic 3.17: CSS Architecture and Naming Conventions (BEM)

3.17 CSS Architecture and Naming Conventions (BEM) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Block__Element--Modifier (`.card` • `.card__title` • `.card--featured`) • Flat single-class specificity `(0 • UI, Accessibility & Edge-Case Pitfall Guard (CSS Architecture and Naming Conventions (BEM)) • Hands-On Frontend Implementation & Verification (CSS Architecture and Naming Conventions (BEM)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Architecture

Purpose: Keeping multi-developer stylesheets collision-free and predictable before introducing CSS Modules or Tailwind.

What Topic Is Next → Next Topic 3.18: Modular Component-Based CSS Organization

3.18 Modular Component-Based CSS Organization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Structuring styles into `reset` • `tokens` • `base` • `layout` • UI, Accessibility & Edge-Case Pitfall Guard (Modular Component-Based CSS Organization) • Hands-On Frontend Implementation & Verification (Modular Component-Based CSS Organization) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Modular Component-Based CSS Organization

Purpose: Using modern CSS `@layer` cascade layers so utility overrides cleanly beat base/component rules without high specificity.

What Topic Is Next → Next Topic 3.19: Sass / SCSS (Variables, Nesting, Mixins, Functions and Partials)

3.19 Sass / SCSS (Variables, Nesting, Mixins, Functions and Partials) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Modern Dart Sass `@use` / `@forward` (replaces deprecated `@import`!) • Shallow nesting (`<= 3` levels) • `@mixin` responsive helpers • `@function` • UI, Accessibility & Edge-Case Pitfall Guard (Sass / SCSS (Variables, Nesting, Mixins, Functions and Partials)) • Hands-On Frontend Implementation & Verification (Sass / SCSS (Variables, Nesting, Mixins, Functions and Partials)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Sass / SCSS (Variables, Nesting, Mixins, Functions a...

Purpose: Maintaining enterprise Angular/Bootstrap SCSS design systems with modular `@use "tokens" as *;` partials.

What Topic Is Next → Next Topic 3.20: CSS Debugging with Chrome DevTools

3.20 CSS Debugging with Chrome DevTools [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Elements Styles vs Computed panel • Grid & Flexbox visual overlays • Forcing `:hover`/`:focus-visible` states • Device Mode throttling • UI, Accessibility & Edge-Case Pitfall Guard (CSS Debugging with Chrome DevTools) • Hands-On Frontend Implementation & Verification (CSS Debugging with Chrome DevTools) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Debugging with Chrome DevTools

Purpose: Pinpointing why a rule is crossed out (specificity vs shorthand override) or why a mobile scrollbar appeared in < 30 seconds.

What Topic Is Next → Module 3 Capstone (Week 3 Lab Build: Mobile-First Landing Page & Sass/Tailwind Component Kit) → Next Topic 4.1: Bootstrap 5 Architecture and Installation

Module 3 Hands-On Capstone Project (100 Marks): Week 3 Lab Build: Mobile-First Landing Page & Sass/Tailwind Component Kit — Translate a Figma design spec into a fluid mobile-first landing page (`360px` to `1440px+`) alongside reusable Bootstrap 5 and Tailwind CSS utility components.

MODULE 4 OF 24 • STAGE 1 • 25 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Bootstrap, Tailwind CSS and UI Implementation

Deliver rapid, consistent production UIs using both Bootstrap 5 (enterprise component & SCSS customization) and Tailwind CSS (utility-first Vite setup, responsive/state/dark-mode variants, reusable UI patterns), paired with Figma-to-code handoff — culminating in Month 1 Mini-Project 1.

Objectives: ✓ Build responsive 12-column grids, forms, tables, navbars, and modals using Bootstrap 5 and customize Bootstrap SCSS theme variables. • ✓ Configure Tailwind CSS with Vite and compose clean spacing, typography, color, Flexbox/Grid, responsive (`sm:`, `md:`, `lg:`), state (`hover:`, `focus-visible:`, `group-hover:`), and `dark:` utilities. • ✓ Know when to use Bootstrap (rapid enterprise admin back-offices) vs Tailwind CSS (bespoke custom design systems in React/Next.js). • ✓ Inspect spacing, typography, colors, auto-layout, and SVG exports in Figma and translate designs into clean responsive HTML/CSS. • ✓ Deliver Month 1 Deliverable: Mini-Project 1 — Responsive Multi-Page Business Website (Home, About, Services, Contact).

4.1 Bootstrap 5 Architecture and Installation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Zero-jQuery Bootstrap 5 architecture • Npm/Vite vs CDN setup • CSS bundle vs JS Popper bundle • UI, Accessibility & Edge-Case Pitfall Guard (Bootstrap 5 Architecture and Installation) • Hands-On Frontend Implementation & Verification (Bootstrap 5 Architecture and Installation) • Bootstrap 5 Architecture and Installation — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bootstrap 5 Architecture

Purpose: Maintaining and shipping rapid internal tools and enterprise portals built on Bootstrap 5.

What Topic Is Next → Next Topic 4.2: Bootstrap 12-Column Grid System

4.2 Bootstrap 12-Column Grid System [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.container` / `.container-fluid` -> `.row` (`g-*` gutters) -> `.col-*` hierarchy • Equal-width auto columns vs explicit spans • UI, Accessibility & Edge-Case Pitfall Guard (Bootstrap 12-Column Grid System) • Hands-On Frontend Implementation & Verification (Bootstrap 12-Column Grid System) • Bootstrap 12-Column Grid System — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bootstrap 12-Column Grid System • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Laying out multi-column marketing and form sections quickly.

What Topic Is Next → Next Topic 4.3: Bootstrap Responsive Breakpoints

4.3 Bootstrap Responsive Breakpoints [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mobile-first tiers: default (`<576px`) • `sm` (`>=576px`) • `md` (`>=768px`) • `lg` (`>=992px`) • UI, Accessibility & Edge-Case Pitfall Guard (Bootstrap Responsive Breakpoints) • Hands-On Frontend Implementation & Verification (Bootstrap Responsive Breakpoints) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bootstrap Responsive Breakpoints

Purpose: Adapting `.row-cols-1 row-cols-sm-2 row-cols-lg-4` card grids cleanly across phones, tablets, and desktops.

What Topic Is Next → Next Topic 4.4: Bootstrap Spacing, Flex and Display Utilities

4.4 Bootstrap Spacing, Flex and Display Utilities [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `m-*`/`p-*` (`t • Y` logical start/end!) • `d-none d-md-flex` • `justify-content-between` • UI, Accessibility & Edge-Case Pitfall Guard (Bootstrap Spacing, Flex and Display Utilities) • Hands-On Frontend Implementation & Verification (Bootstrap Spacing, Flex and Display Utilities) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bootstrap Spacing

Purpose: Composing spacing and flex alignment without writing custom one-off wrapper CSS.

What Topic Is Next → Next Topic 4.5: Bootstrap UI Components (Buttons, Cards, Navbar, Modal, Alerts)

4.5 Bootstrap UI Components (Buttons, Cards, Navbar, Modal, Alerts) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.navbar-expand-lg` • `.card` • `.btn` • `.modal` (`data-bs-toggle` • UI, Accessibility & Edge-Case Pitfall Guard (Bootstrap UI Components (Buttons, Cards, Navbar, Modal, Alerts)) • Hands-On Frontend Implementation & Verification (Bootstrap UI Components (Buttons, Cards, Navbar, Modal, Alerts)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bootstrap UI Components (Buttons, Cards, Navbar, Mod...

Purpose: Implementing accessible collapsible navbars, confirmation modals, and status alerts.

What Topic Is Next → Next Topic 4.6: Bootstrap Form Controls and Validation Styling

4.6 Bootstrap Form Controls and Validation Styling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.form-label` • `.form-control` • `.form-select` • `.form-check` • UI, Accessibility & Edge-Case Pitfall Guard (Bootstrap Form Controls and Validation Styling) • Hands-On Frontend Implementation & Verification (Bootstrap Form Controls and Validation Styling) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bootstrap Form Controls

Purpose: Building clean multi-column business inquiry forms with inline field error messages.

What Topic Is Next → Next Topic 4.7: Bootstrap Responsive Tables

4.7 Bootstrap Responsive Tables [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.table` • `.table-striped` • `.table-hover` • `.align-middle` • UI, Accessibility & Edge-Case Pitfall Guard (Bootstrap Responsive Tables) • Hands-On Frontend Implementation & Verification (Bootstrap Responsive Tables) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bootstrap Responsive Tables

Purpose: Displaying SLA comparison and pricing matrix tables cleanly on mobile viewports.

What Topic Is Next → Next Topic 4.8: Customizing Bootstrap via SCSS Variables

4.8 Customizing Bootstrap via SCSS Variables [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Overriding `$primary` • `$font-family-sans-serif` • `$border-radius` BEFORE importing Bootstrap SCSS • Importing only used Bootstrap SCSS modules to cut bundle size • UI, Accessibility & Edge-Case Pitfall Guard (Customizing Bootstrap via SCSS Variables) • Hands-On Frontend Implementation & Verification (Customizing Bootstrap via SCSS Variables) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Customizing Bootstrap via SCSS Variables

Purpose: Giving a Bootstrap app a bespoke brand identity instead of the default `#0d6efd` "generic Bootstrap" look.

What Topic Is Next → Next Topic 4.9: Tailwind CSS Utility-First Architecture

4.9 Tailwind CSS Utility-First Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why co-located atomic utility classes eliminate dead CSS growth (`O(1)` production CSS size!) • Naming bikeshedding in component frameworks • UI, Accessibility & Edge-Case Pitfall Guard (Tailwind CSS Utility-First Architecture) • Hands-On Frontend Implementation & Verification (Tailwind CSS Utility-First Architecture) • Tailwind CSS Utility-First Architecture — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Tailwind CSS Utility-First Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Primary styling foundation for modern React, Next.js, and design-system teams.

What Topic Is Next → Next Topic 4.10: Tailwind CSS Setup with Vite

4.10 Tailwind CSS Setup with Vite [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Installing Tailwind CSS + Vite plugin (`@tailwindcss/vite` / PostCSS) • `@import "tailwindcss"` • `@theme` tokens • Automatic unused-class tree-shaking in production builds • UI, Accessibility & Edge-Case Pitfall Guard (Tailwind CSS Setup with Vite) • Hands-On Frontend Implementation & Verification (Tailwind CSS Setup with Vite) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Tailwind CSS Setup with Vite

Purpose: Shipping a `< 15KB` gzipped production CSS bundle containing only the exact utilities used in your templates.

What Topic Is Next → Next Topic 4.11: Tailwind Spacing, Typography and Color Tokens

4.11 Tailwind Spacing, Typography and Color Tokens [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 4px base spacing scale (`p-4` = `1rem`) • `space-y-*` • Typographic scale (`text-sm` • `font-semibold` • UI, Accessibility & Edge-Case Pitfall Guard (Tailwind Spacing, Typography and Color Tokens) • Hands-On Frontend Implementation & Verification (Tailwind Spacing, Typography and Color Tokens) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Tailwind Spacing

Purpose: Maintaining rhythmic 4px/8px visual consistency across every page without arbitrary magic pixel values.

What Topic Is Next → Next Topic 4.12: Tailwind Flexbox and CSS Grid Utilities

4.12 Tailwind Flexbox and CSS Grid Utilities [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `flex items-center justify-between gap-4` • `shrink-0` • `min-w-0 flex-1` • `grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6` • UI, Accessibility & Edge-Case Pitfall Guard (Tailwind Flexbox and CSS Grid Utilities) • Hands-On Frontend Implementation & Verification (Tailwind Flexbox and CSS Grid Utilities) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Tailwind Flexbox

Purpose: Building complex responsive grids and card headers right in markup with zero context-switching.

What Topic Is Next → Next Topic 4.13: Tailwind Responsive Breakpoint Prefixes

4.13 Tailwind Responsive Breakpoint Prefixes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mobile-first `sm:` (`640px`) • `md:` (`768px`) • `lg:` (`1024px`) • `xl:` (`1280px`) • UI, Accessibility & Edge-Case Pitfall Guard (Tailwind Responsive Breakpoint Prefixes) • Hands-On Frontend Implementation & Verification (Tailwind Responsive Breakpoint Prefixes) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Tailwind Responsive Breakpoint Prefixes

Purpose: Understanding that unprefixed `text-center` applies to ALL screens unless overridden at `lg:text-left` (not "mobile only"!).

What Topic Is Next → Next Topic 4.14: Tailwind Interactive State Variants

4.14 Tailwind Interactive State Variants [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `hover:` • `focus-visible:outline-2 focus-visible:outline-sky-600` • `active:scale-98` • `disabled:opacity-50 disabled:cursor-not-allowed` • UI, Accessibility & Edge-Case Pitfall Guard (Tailwind Interactive State Variants) • Hands-On Frontend Implementation & Verification (Tailwind Interactive State Variants) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Tailwind Interactive State Variants

Purpose: Animating an arrow icon smoothly when hovering anywhere on its parent card via `group` and `group-hover:translate-x-1`.

What Topic Is Next → Next Topic 4.15: Tailwind Dark Mode Implementation

4.15 Tailwind Dark Mode Implementation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Class/selector strategy (`[data-theme="dark"]` / `.dark`) • `dark:bg-slate-950 dark:text-slate-100 dark:border-slate-800` • Preventing white flash (FOUC) via inline `<head>` theme script & `localStorage` • UI, Accessibility & Edge-Case Pitfall Guard (Tailwind Dark Mode Implementation) • Hands-On Frontend Implementation & Verification (Tailwind Dark Mode Implementation) • Tailwind Dark Mode Implementation — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Tailwind Dark Mode Implementation

Purpose: Delivering flicker-free Light/Dark theme switching with `prefers-color-scheme` fallback.

What Topic Is Next → Next Topic 4.16: Reusable UI Component Design Patterns

4.16 Reusable UI Component Design Patterns [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Designing consistent Button variants (`primary` • `secondary` • `ghost` • `danger`) • UI, Accessibility & Edge-Case Pitfall Guard (Reusable UI Component Design Patterns) • Hands-On Frontend Implementation & Verification (Reusable UI Component Design Patterns) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Reusable UI Component Design Patterns

Purpose: Establishing a cohesive visual language across Homepage, About, Services, and Contact pages.

What Topic Is Next → Next Topic 4.17: Figma Handoff Fundamentals for Frontend Developers

4.17 Figma Handoff Fundamentals for Frontend Developers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Navigating Figma frames • Auto Layout -> Flexbox mapping (`Gap` • `Hug` = `w-fit` • `Fill` = `flex-1`) • UI, Accessibility & Edge-Case Pitfall Guard (Figma Handoff Fundamentals for Frontend Developers) • Hands-On Frontend Implementation & Verification (Figma Handoff Fundamentals for Frontend Developers) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Figma Handoff Fundamentals for Frontend Developers

Purpose: Translating a product designer's Figma specification accurately without pixel-guessing.

What Topic Is Next → Next Topic 4.18: Design-to-Code Implementation & Responsive QA

4.18 Design-to-Code Implementation & Responsive QA [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Breaking a full page design into landmark sections & atomic components • Checking `320px` • `375px` • `768px` • UI, Accessibility & Edge-Case Pitfall Guard (Design-to-Code Implementation & Responsive QA) • Hands-On Frontend Implementation & Verification (Design-to-Code Implementation & Responsive QA) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Design-to-Code Implementation

Purpose: Delivering Month 1 Mini-Project 1 (Responsive Multi-Page Business Website) to agency/product quality standards.

What Topic Is Next → Module 4 Capstone (Project 1: Complete Responsive Multi-Page Business Website) → Next Topic 5.1: JavaScript Introduction, Engines and Browser Execution

Module 4 Hands-On Capstone Project (100 Marks): Project 1: Complete Responsive Multi-Page Business Website — Design and build a fast, semantic, W3C-valid, mobile-first multi-page digital agency / SaaS business website from scratch using HTML5, CSS3 (Flexbox & Grid), Sass/SCSS variables, Bootstrap 5...

MODULE 5 OF 24 • STAGE 2 • 25 LIVE + LAB HOURS • 19 TOPICS (152 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: JavaScript Fundamentals

Build rock-solid JavaScript programming logic — V8 execution model, `let`/`const` & Temporal Dead Zone, primitives vs references, type coercion, strict `===` equality, conditionals, loops, functions, default/rest parameters, lexical scope, hoisting, strict mode, `try/catch/finally`, and Chrome DevTools breakpoint debugging.

Objectives: ✓ Declare state safely using `const` by default and `let` when reassignment is required, avoiding legacy `var` bugs. • ✓ Predict JavaScript type coercion rules accurately and enforce strict equality (`===` / `!==`) and explicit `Number()` / `String()` conversions. • ✓ Write clean, single-purpose functions using default/rest parameters and early-return guard clauses. • ✓ Explain global, function, and block lexical scope, Temporal Dead Zone (TDZ), hoisting, and strict mode in technical interviews. • ✓ Handle runtime errors gracefully with `try/catch/finally` and debug state step-by-step using Chrome DevTools breakpoints.

5.1 JavaScript Introduction, Engines and Browser Execution [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: ECMAScript specification (ES6–ES2024+) • V8/SpiderMonkey/JavaScriptCore JIT engines • Single-threaded main thread + Web APIs • UI, Accessibility & Edge-Case Pitfall Guard (JavaScript Introduction, Engines and Browser Execution) • Hands-On Frontend Implementation & Verification (JavaScript Introduction, Engines and Browser Execution) • JavaScript Introduction, Engines and Browser Execution — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JavaScript Introduction

Purpose: Understanding how modern JS engines parse ASTs, compile hot paths, and share the main thread with UI rendering.

What Topic Is Next → Next Topic 5.2: JavaScript Development Setup (Vite, Console & ESLint Basics)

5.2 JavaScript Development Setup (Vite, Console & ESLint Basics) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Scaffolding vanilla JS with Vite (`npm create vite@latest`) • `<script type="module">` • Console inspection (`console.table` • `console.time`) • UI, Accessibility & Edge-Case Pitfall Guard (JavaScript Development Setup (Vite, Console & ESLint Basics)) • Hands-On Frontend Implementation & Verification (JavaScript Development Setup (Vite, Console & ESLint Basics)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JavaScript Development Setup (Vite, Console & ESLint...

Purpose: Running instant HMR local dev servers with ES module support.

What Topic Is Next → Next Topic 5.3: Variables (var, let, const and Temporal Dead Zone)

5.3 Variables (var, let, const and Temporal Dead Zone) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Function-scoped `var` pitfalls vs block-scoped `let` & `const` • Temporal Dead Zone (`ReferenceError` before initialization) • `const` reference vs object mutability • UI, Accessibility & Edge-Case Pitfall Guard (Variables (var, let, const and Temporal Dead Zone)) • Hands-On Frontend Implementation & Verification (Variables (var, let, const and Temporal Dead Zone)) • Variables (var, let, const and Temporal Dead Zone) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Variables (var, let, const and Temporal Dead Zone)

Purpose: Defaulting to `const` for 95% of declarations so accidental reassignments are caught immediately.

What Topic Is Next → Next Topic 5.4: Data Types (7 Primitives vs Reference Objects)

5.4 Data Types (7 Primitives vs Reference Objects) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `string` • `number` (IEEE-754 float + `NaN`/`Infinity`) • `bigint` • `boolean` • UI, Accessibility & Edge-Case Pitfall Guard (Data Types (7 Primitives vs Reference Objects)) • Hands-On Frontend Implementation & Verification (Data Types (7 Primitives vs Reference Objects)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Data Types (7 Primitives vs Reference Objects)

Purpose: Checking arrays with `Array.isArray(x)` and null safely (`x === null`) instead of broken `typeof x === "object"` alone!

What Topic Is Next → Next Topic 5.5: Explicit and Implicit Type Conversion (Coercion)

5.5 Explicit and Implicit Type Conversion (Coercion) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: String `+` concatenation trap (`"5" + 2 === "52"` vs `"5" - 2 === 3`!) • Falsy values (`false • Null • Undefined • UI, Accessibility & Edge-Case Pitfall Guard (Explicit and Implicit Type Conversion (Coercion)) • Hands-On Frontend Implementation & Verification (Explicit and Implicit Type Conversion (Coercion)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production

Runtime Execution & Memory Lifecycle for Explicit

Purpose: Converting HTML `<input>` `.value` strings (`"1200"`) explicitly via `Number(input.value.trim())` before adding totals!

What Topic Is Next → Next Topic 5.6: Operators (Arithmetic, Assignment, Comparison & Short-Circuit Logical)

5.6 Operators (Arithmetic, Assignment, Comparison & Short-Circuit Logical) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `+ - * / % **` exponentiation • Logical `&&` • `||` • `!` • UI, Accessibility & Edge-Case Pitfall Guard (Operators (Arithmetic, Assignment, Comparison & Short-Circuit Logical)) • Hands-On Frontend Implementation & Verification (Operators (Arithmetic, Assignment, Comparison & Short-Circuit Logical)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Operators (Arithmetic, Assignment, Comparison & Shor...

Purpose: Preserving valid `0` quantities or `""` strings by choosing `??` over `||` when applying fallback defaults.

What Topic Is Next → Next Topic 5.7: Equality Comparisons (== vs === and Object.is)

5.7 Equality Comparisons (== vs === and Object.is) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Abstract coercive equality `==` (`0 == ""` is `true`! `[] == ![]` is `true`!) vs strict `===` (no coercion) • `Object.is(NaN • NaN)` (`true`) • UI, Accessibility & Edge-Case Pitfall Guard (Equality Comparisons (== vs === and Object.is)) • Hands-On Frontend Implementation & Verification (Equality Comparisons (== vs === and Object.is)) • Equality Comparisons (== vs === and Object.is) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Equality Comparisons (== vs === and Object.is)

Purpose: Enforcing `===` and `!==` (`eqeqeq` rule) across 100% of JavaScript and TypeScript codebases.

What Topic Is Next → Next Topic 5.8: Conditional Statements (if/else, switch and Ternary)

5.8 Conditional Statements (if/else, switch and Ternary) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Early-return guard clauses • `switch` with strict `===` case matching & `break`/`return` • Concise single-level `cond ? a : b` expressions • UI, Accessibility & Edge-Case Pitfall Guard (Conditional Statements (if/else, switch and Ternary)) • Hands-On Frontend Implementation & Verification (Conditional Statements (if/else, switch and Ternary)) • Conditional Statements (if/else, switch and Ternary) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Conditional Statements (if/else, switch and Ternary)

Purpose: Computing letter grades and tax brackets cleanly with early validation guards.

What Topic Is Next → Next Topic 5.9: Loops (for, while, do-while, for...of and for...in)

5.9 Loops (for, while, do-while, for...of and for...in) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Indexed `for` • Condition `while` • `do-while` • Iterable values `for...of` (arrays/strings/Maps/Sets) vs enumerable object keys `for...in` (avoid on arrays!) • UI, Accessibility & Edge-Case Pitfall Guard (Loops (for, while, do-while, for...of and for...in)) • Hands-On Frontend Implementation & Verification (Loops (for, while, do-while, for...of and for...in)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Loops (for, while, do-while, for...of and for...in)

Purpose: Iterating strings by full Unicode code points (`for (const ch of str)`) and arrays cleanly via `for...of`.

What Topic Is Next → Next Topic 5.10: Loop Control (break and continue)

5.10 Loop Control (break and continue) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Short-circuiting search loops immediately via `break`/`return` • Skipping invalid items via `continue` • `O(sqrt(n))` primality optimization • UI, Accessibility & Edge-Case Pitfall Guard (Loop Control (break and continue)) • Hands-On Frontend Implementation & Verification (Loop Control (break and continue)) • Loop Control (break and continue) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Loop Control (break and continue)

Purpose: Exiting divisor checks at `d * d <= n` in a Prime Number Checker.

What Topic Is Next → Next Topic 5.11: Functions (Declarations vs Function Expressions)

5.11 Functions (Declarations vs Function Expressions) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: First-class functions (assigned to variables • Passed as callbacks • Returned from functions) • Hoisted declarations vs `const fn = function() {}` expressions • UI, Accessibility & Edge-Case Pitfall Guard (Functions (Declarations vs Function Expressions)) • Hands-On Frontend Implementation & Verification (Functions (Declarations vs Function Expressions)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Functions (Declarations vs Function Expressions)

Purpose: Passing validator and formatter callbacks cleanly into reusable UI helpers.

What Topic Is Next → Next Topic 5.12: Function Parameters (Default, Rest ...args and arguments)

5.12 Function Parameters (Default, Rest ...args and arguments) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Default parameters `fn(page = 1 • Size = 20)` (triggered ONLY when `undefined` • NOT `null`!) • True-array rest `...nums` vs legacy array-like `arguments` • UI, Accessibility & Edge-Case Pitfall Guard (Function Parameters (Default, Rest ...args and arguments)) • Hands-On Frontend Implementation & Verification (Function Parameters (Default, Rest ...args and arguments)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Function Parameters (Default, Rest ...args and argum...

Purpose: Accepting variable argument lists cleanly with `function sumAll(...values)` as a real Array.

What Topic Is Next → Next Topic 5.13: Return Values, Pure Functions and Side Effects

5.13 Return Values, Pure Functions and Side Effects [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implicit `undefined` return when omitted • Returning structured objects `{ ok • Data • Error }` • UI, Accessibility & Edge-Case Pitfall Guard (Return Values, Pure Functions and Side Effects) • Hands-On Frontend Implementation & Verification (Return Values, Pure Functions and Side Effects) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Return Values

Purpose: Keeping React reducers, selectors, and pricing helpers 100% pure so unit tests need zero DOM setup.

What Topic Is Next → Next Topic 5.14: Scope (Global, Function and Block Scope)

5.14 Scope (Global, Function and Block Scope) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lexical static scoping (determined where a function is *defined* • Not where it is called!) • Block `{}` isolation of `let`/`const` • Avoiding global namespace pollution via ES modules • UI, Accessibility & Edge-Case Pitfall Guard (Scope (Global, Function and Block Scope)) • Hands-On Frontend Implementation & Verification (Scope (Global, Function and Block Scope)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Scope (Global, Function and Block Scope)

Purpose: Isolating module-private constants and state inside ES modules without leaking onto `window`.

What Topic Is Next → Next Topic 5.15: Hoisting (Function Declarations, var and let/const TDZ)

5.15 Hoisting (Function Declarations, var and let/const TDZ) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Creation phase vs Execution phase • Why `function foo(){}` can be called above its line • Why `var` initializes to `undefined` • Why `let`/`const` stay in TDZ until evaluated • UI, Accessibility & Edge-Case Pitfall Guard (Hoisting (Function Declarations, var and let/const TDZ)) • Hands-On Frontend Implementation & Verification (Hoisting (Function Declarations, var and let/const TDZ)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Hoisting (Function Declarations, var and let/const T...

Purpose: Answering one of the top-3 JavaScript Round-1 output-prediction interview questions accurately.

What Topic Is Next → Next Topic 5.16: JavaScript Strict Mode ("use strict")

5.16 JavaScript Strict Mode ("use strict") [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Automatic strict mode inside ES Modules (`type="module"`) • `class` bodies! Catches undeclared globals • Duplicate params • Mutating frozen props • UI, Accessibility & Edge-Case Pitfall Guard (JavaScript Strict Mode ("use strict")) • Hands-On Frontend Implementation & Verification (JavaScript Strict Mode ("use strict")) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JavaScript Strict Mode ("use strict")

Purpose: Preventing accidental `window` mutation when a plain function references `this`.

What Topic Is Next → Next Topic 5.17: Error Handling (try, catch, finally and Custom Error Classes)

5.17 Error Handling (try, catch, finally and Custom Error Classes) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `new Error(msg • { cause })` • `TypeError` • `RangeError` • UI, Accessibility & Edge-Case Pitfall Guard (Error Handling (try, catch, finally and Custom Error Classes)) • Hands-On Frontend Implementation & Verification (Error Handling (try, catch, finally and Custom Error Classes)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Error Handling (try, catch, finally and Custom Error...

Purpose: Wrapping `JSON.parse(localStorage.getItem(...))` in `try/catch` so corrupted browser storage never white-screens your app!

What Topic Is Next → Next Topic 5.18: Debugging with Console Methods and Chrome DevTools Breakpoints

5.18 Debugging with Console Methods and Chrome DevTools Breakpoints [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Line & conditional breakpoints • Watch expressions • Call Stack inspection • Step Over / Step Into / Step Out • UI, Accessibility & Edge-Case Pitfall Guard (Debugging with Console Methods and Chrome DevTools Breakpoints) • Hands-On Frontend Implementation & Verification (Debugging with Console Methods and Chrome DevTools Breakpoints) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Debugging with Console Methods

Purpose: Setting a conditional breakpoint (`item.id === 42`) inside a 500-item loop instead of spamming 500 `console.log` lines.

What Topic Is Next → Next Topic 5.19: Clean JavaScript Conventions & Week 5 Git Repository Workflow

5.19 Clean JavaScript Conventions & Week 5 Git Repository Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: CamelCase identifiers • PascalCase classes/components • Boolean `is*`/`has*`/`can*` prefixes • JSDoc hints • UI, Accessibility & Edge-Case Pitfall Guard (Clean JavaScript Conventions & Week 5 Git Repository Workflow) • Hands-On Frontend Implementation & Verification (Clean JavaScript Conventions & Week 5 Git Repository Workflow) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Clean JavaScript Conventions

Purpose: Writing readable self-documenting frontend logic ready for team PR reviews.

What Topic Is Next → Module 5 Capstone (Week 5 Lab Build: JavaScript Logic & Interactive Financial/Academic Calculators) → Next Topic 6.1: Strings and String Manipulation Methods

Module 5 Hands-On Capstone Project (100 Marks): Week 5 Lab Build: JavaScript Logic & Interactive Financial/Academic Calculators — Implement a modular ES6+ utility pack (EMI/Compound Interest Calculator, Student Grade Analyzer, Number Pattern Engine, and Unit Converter) with strict `Number` input guards.

MODULE 6 OF 24 • STAGE 2 • 25 LIVE + LAB HOURS • 33 TOPICS (264 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: JavaScript Arrays, Objects, Strings and ES6+

Master modern ES6–ES2024+ data manipulation with full topic-level granularity — Strings, every individual Array operation (`declaration`, `access`, `mutation`, `traversal`, `map`, `filter`, `reduce`, `find`, `some`, `every`, `sort`, `slice`, `splice`, `destructuring`, `spread`), Objects, Optional Chaining, Nullish Coalescing, ES Modules, `Set`, `Map`, `WeakMap`/`WeakSet`, `Symbol`, Iterators, Generators, and `class` OOP with `#private` fields.

Objectives: ✓ Distinguish mutating array methods (`push`, `pop`, `splice`, `sort`) from non-mutating immutable methods (`map`, `filter`, `slice`, `toSorted`, `toSpliced`, `[...arr]`) critical for React/Redux state! • ✓ Chain `map`, `filter`, `reduce`, `find`, `some`, and `every` fluently to transform API payloads. • ✓ Manipulate objects cleanly using destructuring, shorthand literals, `Object.entries`/`fromEntries`, `?.`, and `??`. • ✓ Modularize applications with ES

`import`/`export` and choose `Map`, `Set`, `WeakMap`, Symbols, Iterators, Generators, and ES6 Classes appropriately.

6.1 Strings and String Manipulation Methods [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: UTF-16 immutability • `trim` • `slice` • `includes` • UI, Accessibility & Edge-Case Pitfall Guard (Strings and String Manipulation Methods) • Hands-On Frontend Implementation & Verification (Strings and String Manipulation Methods) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Strings

Purpose: Formatting order IDs (`String(42).padStart(6, "0")` -> `"000042"`), normalizing search queries, and slugifying titles.

What Topic Is Next → Next Topic 6.2: Array Declaration and Creation

6.2 Array Declaration and Creation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Array literals `[]` • `Array.from({ length } • I) => i)` • `Array.of(...)` vs sparse `new Array(5)` trap! • UI, Accessibility & Edge-Case Pitfall Guard (Array Declaration and Creation) • Hands-On Frontend Implementation & Verification (Array Declaration and Creation) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array Declaration

Purpose: Generating pagination page-number arrays `[1, 2, 3, 4, 5]` cleanly via `Array.from({ length: totalPages }, (_, i) => i + 1)`.

What Topic Is Next → Next Topic 6.3: Array Access and Negative Indexing (.at)

6.3 Array Access and Negative Indexing (.at) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Zero-based `arr[0]` • `arr.length` • Out-of-bounds `undefined` • Modern `arr.at(-1)` for reading the last element cleanly • UI, Accessibility & Edge-Case Pitfall Guard (Array Access and Negative Indexing (.at)) • Hands-On Frontend Implementation & Verification (Array Access and Negative Indexing (.at)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array Access

Purpose: Reading the latest breadcrumb or message cleanly via `messages.at(-1)` instead of `messages[messages.length - 1]`.

What Topic Is Next → Next Topic 6.4: Array Mutation (push, pop, shift, unshift)

6.4 Array Mutation (push, pop, shift, unshift) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: End-of-array `O(1)` `push`/`pop` vs start-of-array `O(n)` re-indexing `shift`/`unshift` • Return values (`push` returns new length • NOT the array!) • Why React forbids mutating state arrays directly • UI, Accessibility & Edge-Case Pitfall Guard (Array Mutation (push, pop, shift, unshift)) • Hands-On Frontend Implementation & Verification (Array Mutation (push, pop, shift, unshift)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array Mutation (push, pop, shift, unshift)

Purpose: Building local algorithmic stacks (`push`/`pop`) while using `[...items, newItem]` when updating immutable UI state.

What Topic Is Next → Next Topic 6.5: Array Traversal (for, for...of, entries and forEach)

6.5 Array Traversal (for, for...of, entries and forEach) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Index `for` • Clean `for (const [idx • Item] of arr.entries())` (supports `break`/`continue`/`await`!) • Vs `.forEach()` (cannot `break` or `await` properly!) • UI, Accessibility & Edge-Case Pitfall Guard (Array Traversal (for, for...of, entries and forEach)) • Hands-On Frontend Implementation & Verification (Array Traversal (for, for...of, entries and forEach)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array Traversal (for, for...of, entries and forEach)

Purpose: Choosing `for...of` whenever async `await` calls or early `break` exits are needed.

What Topic Is Next → Next Topic 6.6: Array.prototype.map

6.6 Array.prototype.map [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 1-to-1 immutable projection returning a new array of identical length • Transforming API DTOs into UI view-models • `["1" • "2"].map(Number)` vs `parseInt` radix trap! • UI, Accessibility & Edge-Case Pitfall Guard (Array.prototype.map) • Hands-On Frontend Implementation & Verification (Array.prototype.map) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array.prototype.map

Purpose: Mapping raw API product objects into formatted UI card props and JSX `<ProductCard key={p.id} />` lists.

What Topic Is Next → Next Topic 6.7: Array.prototype.filter

6.7 Array.prototype.filter [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable subset selection based on boolean predicate • Combining search keyword + category + inStock predicates • Chaining `.filter(...).map(...)` • UI, Accessibility & Edge-Case Pitfall Guard (Array.prototype.filter) • Hands-On Frontend Implementation & Verification (Array.prototype.filter) • Array.prototype.filter — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array.prototype.filter

Purpose: Removing a deleted task immutably (`tasks.filter(t => t.id !== deletedId)`) and filtering product catalogs.

What Topic Is Next → Next Topic 6.8: Array.prototype.reduce

6.8 Array.prototype.reduce [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Accumulator folding `(acc • Item • Index) => nextAcc` with MANDATORY initial value (`0` • `{}`) • UI, Accessibility & Edge-Case Pitfall Guard (Array.prototype.reduce) • Hands-On Frontend Implementation & Verification (Array.prototype.reduce) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array.prototype.reduce

Purpose: Computing shopping-cart `{ subtotal, itemCount }` or indexing an array by `id` in a single `O(n)` pass.

What Topic Is Next → Next Topic 6.9: Array.prototype.find and findIndex

6.9 Array.prototype.find and findIndex [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Short-circuiting first matching element (`find` -> element or `undefined`) • Index (`findIndex` -> index or `-1` • ES2023 `findLast`) • UI, Accessibility & Edge-Case Pitfall Guard (Array.prototype.find and findIndex) • Hands-On Frontend Implementation & Verification (Array.prototype.find and findIndex) • Array.prototype.find and findIndex — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array.prototype.find

Purpose: Looking up a selected product by `slug` in `O(n)` short-circuiting as soon as found (unlike `.filter(...)[0]` which scans the entire rest of the array!).

What Topic Is Next → Next Topic 6.10: Array.prototype.some

6.10 Array.prototype.some [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Existential short-circuit check (`true` as soon as AT LEAST ONE item matches • `false` on empty array `[]`) • UI, Accessibility & Edge-Case Pitfall Guard (Array.prototype.some) • Hands-On Frontend Implementation & Verification (Array.prototype.some) • Array.prototype.some — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array.prototype.some • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Checking `cart.some(item => item.stock === 0)` to disable the Checkout button when any item is out of stock.

What Topic Is Next → Next Topic 6.11: Array.prototype.every

6.11 Array.prototype.every [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Universal short-circuit check (`true` iff ALL items satisfy predicate • Note vacuous truth `[].every(...) === true`!) • UI, Accessibility & Edge-Case Pitfall Guard (Array.prototype.every) • Hands-On Frontend Implementation & Verification (Array.prototype.every) • Array.prototype.every — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array.prototype.every • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Enabling a "Submit Quiz" button only when `questions.length > 0 && questions.every(q => q.selectedOption !== null)`.

What Topic Is Next → Next Topic 6.12: Array.prototype.sort and toSorted (Custom Comparators)

6.12 Array.prototype.sort and toSorted (Custom Comparators) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Default UTF-16 lexicographical sort trap • Numeric `(a • B) => a - b` • Locale strings `a.name.localeCompare(b.name)` • UI, Accessibility & Edge-Case Pitfall Guard (Array.prototype.sort and toSorted (Custom Comparators)) • Hands-On Frontend Implementation & Verification (Array.prototype.sort and toSorted (Custom Comparators)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array.prototype.sort

Purpose: Sorting product listings by Price Low-to-High, Rating High-to-Low, and Name A–Z without mutating cached API state!

What Topic Is Next → Next Topic 6.13: Array.prototype.slice (Non-Mutating Subarrays)

6.13 Array.prototype.slice (Non-Mutating Subarrays) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `arr.slice(start • End)` (`end` is exclusive!) • Negative indices `slice(-5)` • Client-side pagination `items.slice((page - 1) * size • UI, Accessibility & Edge-Case Pitfall Guard (Array.prototype.slice (Non-Mutating Subarrays)) • Hands-On Frontend Implementation & Verification (Array.prototype.slice (Non-Mutating Subarrays)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array.prototype.slice (Non-Mutating Subarrays)

Purpose: Extracting paginated table rows and "Recent 5 Orders" widgets without mutating the source array.

What Topic Is Next → Next Topic 6.14: Array.prototype.splice and toSpliced

6.14 Array.prototype.splice and toSpliced [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: In-place delete/insert/replace `arr.splice(idx • DelCount • ...added)` (returns removed items!) vs immutable ES2023 `arr.toSpliced(...)` for drag-• -drop reordering • UI, Accessibility & Edge-Case Pitfall Guard (Array.prototype.splice and toSpliced) • Hands-On Frontend Implementation & Verification (Array.prototype.splice and toSpliced) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array.prototype.splice

Purpose: Reordering Kanban task cards immutably when a user drags item `fromIndex` to `toIndex`.

What Topic Is Next → Next Topic 6.15: Array Destructuring

6.15 Array Destructuring [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Positional unpacking `const [first • Second • ...rest] = items` • Skipping slots `[• UI, Accessibility & Edge-Case Pitfall Guard (Array Destructuring) • Hands-On Frontend Implementation & Verification (Array Destructuring) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array Destructuring

Purpose: Unpacking React `useState` tuples and `Promise.all([fetchUser(), fetchOrders()])` results cleanly.

What Topic Is Next → Next Topic 6.16: JavaScript Objects (Creation, Dynamic Keys and Reference Semantics)

6.16 JavaScript Objects (Creation, Dynamic Keys and Reference Semantics) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Object literals • Dot `obj.key` vs computed bracket `obj[dynamicField]` access • `in` vs `Object.hasOwn(obj • Key)` • UI, Accessibility & Edge-Case Pitfall Guard (JavaScript Objects (Creation, Dynamic Keys and Reference Semantics)) • Hands-On Frontend Implementation & Verification (JavaScript Objects (Creation, Dynamic Keys and Reference Semantics)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JavaScript Objects (Creation, Dynamic Keys and Refer...

Purpose: Handling dynamic form state updates `{ ...form, [event.target.name]: event.target.value }`.

What Topic Is Next → Next Topic 6.17: Object Static Methods (keys, values, entries, fromEntries, assign, freeze, structuredClone)

6.17 Object Static Methods (keys, values, entries, fromEntries, assign, freeze, structuredClone) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Object.keys/values/entries` • `Object.fromEntries` (filtering/mapping object keys!) • Shallow `Object.assign`/`Object.freeze` vs deep `structuredClone(obj)`! • UI, Accessibility & Edge-Case Pitfall Guard (Object Static Methods (keys, values, entries, fromEntries, assign, freeze, structuredClone)) • Hands-On Frontend Implementation & Verification (Object Static Methods (keys, values, entries, fromEntries, assign, freeze, structuredClone)) • Object Static Methods (keys, values, entries, fromEntries, assign, freeze, structuredClone) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Object Static Methods (keys, values, entries, fromEn...

Purpose: Stripping empty query params cleanly via `Object.fromEntries(Object.entries(params).filter(([, v]) => v !== ""))` and deep-cloning nested configs via native `structuredCl...

What Topic Is Next → Next Topic 6.18: Object Destructuring, Renaming and Nested Defaults

6.18 Object Destructuring, Renaming and Nested Defaults [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `const { id • Title: productTitle • Rating = 0 } = product` • Nested unpacking • UI, Accessibility & Edge-Case Pitfall Guard (Object Destructuring, Renaming and Nested Defaults) • Hands-On Frontend Implementation & Verification (Object Destructuring, Renaming and Nested Defaults) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Object Destructuring

Purpose: Unpacking React component props with clean defaults and renaming colliding API properties.

What Topic Is Next → Next Topic 6.19: Spread Operator (...) for Arrays and Objects

6.19 Spread Operator (...) for Arrays and Objects [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Shallow immutable cloning `[...a • ...b]` • `{ ...defaults • ...overrides }` (later keys win!) • UI, Accessibility & Edge-Case Pitfall Guard (Spread Operator (...) for Arrays and Objects) • Hands-On Frontend Implementation & Verification (Spread Operator (...) for Arrays and Objects) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spread Operator (...) for Arrays

Purpose: Updating nested React user address `{ ...user, address: { ...user.address, city: "Pune" } }` immutably.

What Topic Is Next → Next Topic 6.20: Rest Operator (...) in Functions and Destructuring

6.20 Rest Operator (...) in Functions and Destructuring [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Collecting remaining props `const { password • ...safeUser } = rawUser` (omitting sensitive/internal props!) • Variadic `(...args)` wrappers • UI, Accessibility & Edge-Case Pitfall Guard (Rest Operator (...) in Functions and Destructuring) • Hands-On Frontend Implementation & Verification (Rest Operator (...) in Functions and Destructuring) • Rest Operator (...) in Functions and Destructuring — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Rest Operator (...) in Functions

Purpose: Forwarding remaining HTML attributes in reusable React `<Button variant="primary" {...restProps} />` components!

What Topic Is Next → Next Topic 6.21: Template Literals and Multi-Line Interpolation

6.21 Template Literals and Multi-Line Interpolation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Backtick `` `Hello ${name}` `` interpolation • Multi-line HTML/SQL templates • Tagged template literals (`css` / HTML-escaping tags) • UI, Accessibility & Edge-Case Pitfall Guard (Template Literals and Multi-Line Interpolation) • Hands-On Frontend Implementation & Verification (Template Literals and Multi-Line Interpolation) • Template Literals and Multi-Line Interpolation — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Template Literals

Purpose: Building readable API endpoint URLs (`/api/v1/orders/${encodeURIComponent(orderId)}`) and dynamic ARIA status announcements.

What Topic Is Next → Next Topic 6.22: Arrow Functions (Syntax and Lexical this)

6.22 Arrow Functions (Syntax and Lexical this) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Concise `(a • B) => a + b` • Returning object literals `() => ({ ok: true })` (parentheses required!) • Lexical `this` inheritance • UI, Accessibility & Edge-Case Pitfall Guard (Arrow Functions (Syntax and Lexical this)) • Hands-On Frontend Implementation & Verification (Arrow Functions (Syntax and Lexical this)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Arrow Functions (Syntax and Lexical this)

Purpose: Preserving outer instance `this` inside timer/promise/array callbacks without `.bind(this)`.

What Topic Is Next → Next Topic 6.23: Enhanced Object Literals

6.23 Enhanced Object Literals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Property value shorthand `{ id • Name }` • Concise method syntax `{ save() {} }` • Computed property names `{ [`step_${n}`]: true }` • UI, Accessibility & Edge-Case Pitfall Guard (Enhanced Object Literals) • Hands-On Frontend Implementation & Verification (Enhanced Object Literals) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Enhanced Object Literals

Purpose: Returning clean service/hook APIs `return { items, isLoading, reload };` with zero redundant `items: items` repetition.

What Topic Is Next → Next Topic 6.24: Optional Chaining (?.)

6.24 Optional Chaining (?.) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Safe property `user?.company?.name` • Safe bracket `arr?.[0]` • Safe optional callback invocation `onSuccess?.(data)` (short-circuits to `undefined` ONLY on `null` or `undefined`) • UI, Accessibility & Edge-Case Pitfall Guard (Optional Chaining (?.)) • Hands-On Frontend Implementation & Verification (Optional Chaining (?.)) • Optional Chaining (?.) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Optional Chaining (?.)

Purpose: Safely reading deeply nested optional API fields and invoking optional React callback props (`onClose?.()`).

What Topic Is Next → Next Topic 6.25: Nullish Coalescing (?? vs ||)

6.25 Nullish Coalescing (?? vs ||) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `lhs ?? fallback` triggers ONLY when `lhs` is `null` or `undefined` (preserving `0` • `false` • `""`!) • Syntax rule requiring parentheses when mixed with `&&`/`||` • UI, Accessibility & Edge-Case Pitfall Guard (Nullish Coalescing (?? vs ||)) • Hands-On Frontend Implementation & Verification (Nullish Coalescing (?? vs ||)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Nullish Coalescing (?? vs ||)

Purpose: Displaying `product.stockCount ?? "N/A"` so `stockCount = 0` accurately renders `0` instead of `"N/A"`!

What Topic Is Next → Next Topic 6.26: ES Modules (import, export and Dynamic import())

6.26 ES Modules (import, export and Dynamic import()) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Named exports (`export function ...`) vs default export • Aliased `import { x as y }` • Barrel files caution (tree-shaking!) • Dynamic `const mod = await import("./chart.js")` lazy loading • UI, Accessibility & Edge-Case Pitfall Guard (ES Modules (import, export and Dynamic import())) • Hands-On Frontend Implementation & Verification (ES Modules (import, export and Dynamic import())) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ES Modules (import, export and Dynamic import())

Purpose: Splitting large frontend apps into cohesive modules and lazy-loading heavy chart/PDF libraries on demand.

What Topic Is Next → Next Topic 6.27: Set Collection (Unique Values and Fast Membership)

6.27 Set Collection (Unique Values and Fast Membership) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `new Set(iterable)` • `O(1)` `.add()` • `.has()` • `.delete()` • UI, Accessibility & Edge-Case Pitfall Guard (Set Collection (Unique Values and Fast Membership)) • Hands-On Frontend Implementation & Verification (Set Collection (Unique Values and Fast Membership)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Set Collection (Unique Values and Fast Membership)

Purpose: Tracking selected row IDs in a 5,000-row Data Table with `O(1)` `selectedIds.has(row.id)` instead of `O(n)` `array.includes()`!

What Topic Is Next → Next Topic 6.28: Map Collection (Key-Value Dictionary)

6.28 Map Collection (Key-Value Dictionary) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `new Map()` • Arbitrary key types (numbers • Objects • DOM nodes!) • UI, Accessibility & Edge-Case Pitfall Guard (Map Collection (Key-Value Dictionary)) • Hands-On Frontend Implementation & Verification (Map Collection (Key-Value Dictionary)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Map Collection (Key-Value Dictionary)

Purpose: Caching API responses or indexing entities by numeric `id` safely without user strings colliding with `Object.prototype` keys!

What Topic Is Next → Next Topic 6.29: WeakMap and WeakSet (Garbage-Collectable Object Metadata)

6.29 WeakMap and WeakSet (Garbage-Collectable Object Metadata) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Object-only keys • Weak references that do NOT prevent garbage collection when a DOM node or object is removed • Why `WeakMap` is non-enumerable • UI, Accessibility & Edge-Case Pitfall Guard (WeakMap and WeakSet (Garbage-Collectable Object Metadata)) • Hands-On Frontend Implementation & Verification (WeakMap and WeakSet (Garbage-Collectable Object Metadata)) • WeakMap and WeakSet (Garbage-Collectable Object Metadata) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for WeakMap

Purpose: Associating private computed metadata with DOM elements or object instances with zero memory leaks when nodes unmount!

What Topic Is Next → Next Topic 6.30: Symbol Primitives

6.30 Symbol Primitives [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Guaranteed-unique `Symbol("desc")` keys (collision-free metadata properties) • Global `Symbol.for()` • Well-known symbols (`Symbol.iterator`) • UI, Accessibility & Edge-Case Pitfall Guard (Symbol Primitives) • Hands-On Frontend Implementation & Verification (Symbol Primitives) • Symbol Primitives — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Symbol Primitives

Purpose: Understanding how React marks valid JSX elements internally with `Symbol.for("react.transitional.element")` to block JSON-injection XSS!

What Topic Is Next → Next Topic 6.31: Iterators and the Iterable Protocol

6.31 Iterators and the Iterable Protocol [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `[Symbol.iterator]()` returning `{ next() -> { value • Done } }` • Making custom domain classes work seamlessly with `for...of` • `[...spread]` • UI, Accessibility & Edge-Case Pitfall Guard (Iterators and the Iterable Protocol) • Hands-On Frontend Implementation & Verification (Iterators and the Iterable Protocol) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Iterators

Purpose: Making a custom `DateRange` or `PaginatedCollection` directly iterable with `for...of`.

What Topic Is Next → Next Topic 6.32: Generator Functions (function* and yield)

6.32 Generator Functions (function* and yield) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lazy pull-based sequence evaluation `function*` • Pausing/resuming local state across `yield` • Infinite ID sequences without memory bloat • UI, Accessibility & Edge-Case Pitfall Guard (Generator Functions (function* and yield)) • Hands-On Frontend Implementation & Verification (Generator Functions (function* and yield)) • Generator Functions (function* and yield) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Generator Functions (function* and yield)

Purpose: Lazily chunking large arrays into batches or generating monotonic client request IDs.

What Topic Is Next → Next Topic 6.33: ES6+ Classes, Inheritance (extends/super), #Private Fields and Static Members

6.33 ES6+ Classes, Inheritance (extends/super), #Private Fields and Static Members [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `class` • `constructor` • Prototype methods • `extends` & `super(...)` • UI, Accessibility & Edge-Case Pitfall Guard (ES6+ Classes, Inheritance (extends/super), #Private Fields and Static Members) • Hands-On Frontend Implementation & Verification (ES6+ Classes, Inheritance (extends/super), #Private Fields and Static Members) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ES6+ Classes

Purpose: Authoring clean custom `ApiError` subclasses, state stores, and preparing directly for TypeScript & Angular class services!

What Topic Is Next → Module 6 Capstone (Week 6 Lab Build: Immutable Array/Object Analytics & Cart Transformation Engine) → Next Topic 7.1: Execution Contexts (Global and Function)

Module 6 Hands-On Capstone Project (100 Marks): Week 6 Lab Build: Immutable Array/Object Analytics & Cart Transformation Engine — Build a pure functional e-commerce catalog & payroll analytics engine using `map`, `filter`, `reduce`, `find`, `some`, `every`, `toSorted`, `Map`/`Set`, destructuring, and spread operators w...

MODULE 7 OF 24 • STAGE 2 • 25 LIVE + LAB HOURS • 24 TOPICS (192 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Advanced JavaScript and Asynchronous Programming

Master JavaScript's runtime internals and asynchronous architecture — Execution Contexts, Call Stack, Closures, `this` & `call/apply/bind`, Prototypes, the Browser Event Loop (Microtasks vs Macrotasks), Promises (`all`, `allSettled`, `race`, `any`), `async/await`, `Fetch` + `AbortController`, Browser Storage (`LocalStorage`, `SessionStorage`, `Cookies`), and Web Timers.

Objectives: ✓ Trace Global/Function Execution Contexts, Lexical Environments, Closures, `this` binding rules, and the Prototype chain accurately in interviews. • ✓ Predict exact execution order across synchronous Call Stack code, Promise Microtasks, `requestAnimationFrame`, and `setTimeout` Macrotasks. • ✓ Replace callback pyramids with clean Promises, `Promise.all` / `Promise.allSettled` concurrency, and `async/await` error handling. • ✓ Communicate with REST APIs using `fetch()` and cancel stale in-flight requests cleanly via `AbortController` (preventing search race conditions!). • ✓ Manage `localStorage`, `sessionStorage`, `Cookies` (`HttpOnly`/`SameSite`), `JSON`, and `setTimeout`/`setInterval` debouncing safely.

7.1 Execution Contexts (Global and Function) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Memory Creation phase vs Code Execution phase • Variable Environment • Lexical Environment • `this` binding creation • UI, Accessibility & Edge-Case Pitfall Guard (Execution Contexts (Global and Function)) • Hands-On Frontend Implementation & Verification (Execution Contexts (Global and Function)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Execution Contexts (Global and Function)

Purpose: Understanding why inner functions retain references to outer lexical environments even after the outer frame pops.

What Topic Is Next → Next Topic 7.2: The JavaScript Call Stack

7.2 The JavaScript Call Stack [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: LIFO stack frames • Single-threaded execution • `RangeError: Maximum call stack size exceeded` • Reading stack traces bottom-to-top • UI, Accessibility & Edge-Case Pitfall Guard (The JavaScript Call Stack) • Hands-On Frontend Implementation & Verification (The JavaScript Call Stack) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for The JavaScript Call Stack

Purpose: Reading production error stack traces in Sentry/DevTools to locate exact call origins.

What Topic Is Next → Next Topic 7.3: Lexical Environment and the Scope Chain

7.3 Lexical Environment and the Scope Chain [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Environment Record + outer environment reference pointer walking outward to `null` at global root • UI, Accessibility & Edge-Case Pitfall Guard (Lexical Environment and the Scope Chain) • Hands-On Frontend Implementation & Verification (Lexical Environment and the Scope Chain) • Lexical Environment and the Scope Chain — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Lexical Environment • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Understanding how React event handlers capture the render snapshot's variables.

What Topic Is Next → Next Topic 7.4: Closures (Mechanism, Practical Uses and Memory Nuances)

7.4 Closures (Mechanism, Practical Uses and Memory Nuances) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Functions bundling their outer lexical environment • Data encapsulation • `once(fn)` • `memoize(fn)` • UI, Accessibility & Edge-Case Pitfall Guard (Closures (Mechanism, Practical Uses and Memory Nuances)) • Hands-On Frontend Implementation & Verification (Closures (Mechanism, Practical Uses and Memory Nuances)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Closures (Mechanism, Practical Uses and Memory Nuanc...

Purpose: Implementing `debounce(searchApi, 300)` and `memoize` utilities asked in almost every frontend coding interview!

What Topic Is Next → Next Topic 7.5: The this Keyword (4 Binding Rules)

7.5 The this Keyword (4 Binding Rules) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 1) `new` binding -> new instance • 2) Explicit `call/apply/bind` • 3) Implicit `obj.method()` -> `obj` • 4) Default standalone `fn()` -> `undefined` in strict mode (plus Arrow lexical inheritance!) • UI, Accessibility & Edge-Case Pitfall Guard (The this Keyword (4 Binding Rules)) • Hands-On Frontend Implementation & Verification (The this Keyword (4 Binding Rules)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for The this Keyword (4 Binding Rules)

Purpose: Fixing `TypeError: Cannot read properties of undefined` when passing `store.increment` as a detached callback!

What Topic Is Next → Next Topic 7.6: Explicit Context Control (call, apply and bind)

7.6 Explicit Context Control (call, apply and bind) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immediate invocation `fn.call(ctx • B)` & `fn.apply(ctx • B])` vs permanent bound wrapper `const bound = fn.bind(ctx • PresetArg)` (partial application) • UI, Accessibility & Edge-Case Pitfall Guard (Explicit Context Control (call, apply and bind)) • Hands-On Frontend Implementation & Verification (Explicit Context Control (call, apply and bind)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Explicit Context Control (call, apply and bind)

Purpose: Preserving caller `this` inside higher-order `debounce`/`throttle` wrappers via `fn.apply(this, args)`.

What Topic Is Next → Next Topic 7.7: Prototypes and the [[Prototype]] Chain

7.7 Prototypes and the [[Prototype]] Chain [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Object.getPrototypeOf(obj)` • How `[1 • 2].map` resolves to `Array.prototype.map` -> `Object.prototype` -> `null` • Memory sharing across 10 • UI, Accessibility & Edge-Case Pitfall Guard (Prototypes and the [[Prototype]] Chain) • Hands-On Frontend Implementation & Verification (Prototypes and the [[Prototype]] Chain) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Prototypes

Purpose: Understanding that ES6 `class` methods live once on `ClassName.prototype` rather than duplicating closures per instance.

What Topic Is Next → Next Topic 7.8: Prototypal Inheritance (Object.create and Class Sugar)

7.8 Prototypal Inheritance (Object.create and Class Sugar) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Object.create(proto)` • Constructor functions vs modern `class Child extends Parent` • Property lookup vs own-property shadowing • UI, Accessibility & Edge-Case Pitfall Guard (Prototypal Inheritance (Object.create and Class Sugar)) • Hands-On Frontend Implementation & Verification (Prototypal Inheritance (Object.create and Class Sugar)) • Prototypal Inheritance (Object.create and Class Sugar) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Prototypal Inheritance (Object.create and Class Suga...

Purpose: Reading legacy pre-ES6 library code and explaining how `extends` wires prototype chains under the hood.

What Topic Is Next → Next Topic 7.9: Browser Event Loop Architecture

7.9 Browser Event Loop Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Call Stack + Web APIs (timers • Network • DOM) + Microtask Queue + Macrotask (Task) Queue + Render (`requestAnimationFrame` / Layout / Paint) cycle • UI, Accessibility & Edge-Case Pitfall Guard (Browser Event Loop Architecture) • Hands-On Frontend Implementation & Verification (Browser Event Loop Architecture) • Browser Event Loop Architecture — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Browser Event Loop Architecture

Purpose: Keeping UI animations silky smooth at 60fps (`16.6ms` frame budget) by never starving the render tick.

What Topic Is Next → Next Topic 7.10: Microtask Queue (Promises & queueMicrotask)

7.10 Microtask Queue (Promises & queueMicrotask) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.then/.catch/.finally` continuations • `await` resumption • `queueMicrotask()` • Why microtasks drain completely (including newly enqueued microtasks!) before the next macrotask or paint • UI, Accessibility & Edge-Case Pitfall Guard (Microtask Queue (Promises & queueMicrotask)) • Hands-On Frontend Implementation & Verification (Microtask Queue (Promises & queueMicrotask)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Microtask Queue (Promises & queueMicrotask)

Purpose: Batching state flushes before browser paint and predicting async execution order.

What Topic Is Next → Next Topic 7.11: Macrotask Queue (Timers, Events and MessageChannel)

7.11 Macrotask Queue (Timers, Events and MessageChannel) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `setTimeout` • `setInterval` • DOM click/input events • Yielding back to the browser paint cycle between heavy chunks • UI, Accessibility & Edge-Case Pitfall Guard (Macrotask Queue (Timers, Events and MessageChannel)) • Hands-On Frontend Implementation & Verification (Macrotask Queue (Timers, Events and MessageChannel)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Macrotask Queue (Timers, Events and MessageChannel)

Purpose: Chunking a heavy 50,000-row CSV parse across macrotask slices so the progress bar actually repaints on screen!

What Topic Is Next → Next Topic 7.12: Synchronous Blocking Operations & Main-Thread Jank

7.12 Synchronous Blocking Operations & Main-Thread Jank [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Identifying Long Tasks (`> 50ms` in DevTools Performance) that hurt Interaction to Next Paint (INP) • Offloading heavy CPU work to Web Workers awareness • UI, Accessibility & Edge-Case Pitfall Guard (Synchronous Blocking Operations & Main-Thread Jank) • Hands-On Frontend Implementation & Verification (Synchronous Blocking Operations & Main-Thread Jank) • Synchronous Blocking Operations & Main-Thread Jank — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Synchronous Blocking Operations • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Diagnosing sluggish input typing caused by synchronous filtering of 20,000 items on every keystroke.

What Topic Is Next → Next Topic 7.13: Non-Blocking Asynchronous Programming Model

7.13 Non-Blocking Asynchronous Programming Model [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Initiating I/O (network • Timers • File read • IndexedDB) without blocking user clicks/scrolls • UI, Accessibility & Edge-Case Pitfall Guard (Non-Blocking Asynchronous Programming Model) • Hands-On Frontend Implementation & Verification (Non-Blocking Asynchronous Programming Model) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Non-Blocking Asynchronous Programming Model

Purpose: Keeping filter dropdowns and navigation interactive while 3 dashboard analytics widgets load concurrently.

What Topic Is Next → Next Topic 7.14: Callback Functions in Async APIs

7.14 Callback Functions in Async APIs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Timer • Event • `Geolocation` / `FileReader` callbacks • Node/browser error-first callback history • UI, Accessibility & Edge-Case Pitfall Guard (Callback Functions in Async APIs) • Hands-On Frontend Implementation & Verification (Callback Functions in Async APIs) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Callback Functions in Async APIs

Purpose: Wrapping callback-only browser APIs (`FileReader`, `navigator.geolocation`) cleanly into Promises.

What Topic Is Next → Next Topic 7.15: Callback Hell and Inversion of Control

7.15 Callback Hell and Inversion of Control [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pyramid of Doom (`step1(() => step2(() => step3(...)))`) • Scattered duplicate error checks • Why composable Promises solved it • UI, Accessibility & Edge-Case Pitfall Guard (Callback Hell and Inversion of Control) • Hands-On Frontend Implementation & Verification (Callback Hell and Inversion of Control) • Callback Hell and Inversion of Control — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Callback Hell

Purpose: Refactoring deeply nested legacy onboarding steps into flat, readable Promise/`async` pipelines.

What Topic Is Next → Next Topic 7.16: Promises and Lifecycle States (pending, fulfilled, rejected)

7.16 Promises and Lifecycle States (pending, fulfilled, rejected) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `new Promise((resolve • Reject) => ...)` • Immutable settlement (settled once forever!) • `.then` • UI, Accessibility & Edge-Case Pitfall Guard (Promises and Lifecycle States (pending, fulfilled, rejected)) • Hands-On Frontend Implementation & Verification (Promises and Lifecycle States (pending, fulfilled, rejected)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Promises

Purpose: Creating timeout wrappers and clean single-flight async operations.

What Topic Is Next → Next Topic 7.17: Promise Combinators (all, allSettled, race and any)

7.17 Promise Combinators (all, allSettled, race and any) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fail-fast parallel `Promise.all` vs partial-failure-tolerant `Promise.allSettled` (`{ status: "fulfilled"|"rejected" }`) vs first-settled `Promise.race` vs first-fulfilled `Promise.any` • UI, Accessibility & Edge-Case Pitfall Guard (Promise Combinators (all, allSettled, race and any)) • Hands-On Frontend Implementation & Verification (Promise Combinators (all, allSettled, race and any)) • Promise Combinators (all, allSettled, race and any) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Promise Combinators (all, allSettled, race and any) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Using `Promise.allSettled([loadProfile(), loadRecommendations(), loadOrders()])` on a dashboard so a failed Recommendations widget doesn't blank the whole dashboard!

What Topic Is Next → Next Topic 7.18: async / await Functions (Sequential vs Parallel Execution)

7.18 async / await Functions (Sequential vs Parallel Execution) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `async` functions ALWAYS return a Promise • Sequential `await a() • Await b() • ` (`tA + tB`) vs parallel `const [ra • UI, Accessibility & Edge-Case Pitfall Guard (async / await Functions (Sequential vs Parallel Execution)) • Hands-On Frontend Implementation & Verification (async / await Functions (Sequential vs Parallel Execution)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for async / await Functions (Sequential vs Parallel Exec...

Purpose: Cutting dashboard load time in half by firing independent `await Promise.all(...)` calls concurrently instead of waterfall `await`s!

What Topic Is Next → Next Topic 7.19: Asynchronous Error Handling & Unhandled Rejections

7.19 Asynchronous Error Handling & Unhandled Rejections [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `try / catch / finally` around `await` • Checking `!response.ok` on `fetch()` (CRITICAL: `fetch` does NOT reject on HTTP `404` or `500`!) • `unhandledrejection` global listener • UI, Accessibility & Edge-Case Pitfall Guard (Asynchronous Error Handling & Unhandled Rejections) • Hands-On Frontend Implementation & Verification (Asynchronous Error Handling & Unhandled Rejections) • Asynchronous Error Handling & Unhandled Rejections — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Asynchronous Error Handling

Purpose: Checking `if (!res.ok) throw new ApiError(res.status, await res.json())` in every `fetch` wrapper!

What Topic Is Next → Next Topic 7.20: Fetch API (GET, POST, Headers & JSON Payloads)

7.20 Fetch API (GET, POST, Headers & JSON Payloads) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `fetch(url • { method • Headers • Body: JSON.stringify(payload) • UI, Accessibility & Edge-Case Pitfall Guard (Fetch API (GET, POST, Headers & JSON Payloads)) • Hands-On Frontend Implementation & Verification (Fetch API (GET, POST, Headers & JSON Payloads)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Fetch API (GET, POST, Headers & JSON Payloads)

Purpose: Integrating live REST CRUD endpoints cleanly without external libraries.

What Topic Is Next → Next Topic 7.21: AbortController (Cancelling Stale Requests & Race Prevention)

7.21 AbortController (Cancelling Stale Requests & Race Prevention) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `const controller = new AbortController()` • Passing `{ signal: controller.signal }` to `fetch` & `addEventListener` • Aborting previous keystroke request on new input • Ignoring `err.name === "AbortError"` • UI, Accessibility & Edge-Case Pitfall Guard (AbortController (Cancelling Stale Requests & Race Prevention)) • Hands-On Frontend Implementation & Verification (AbortController (Cancelling Stale Requests & Race Prevention)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for AbortController (Cancelling Stale Requests & Race Pr...

Purpose: Preventing an older slow response for `"ja"` from arriving *after* `"java"` and overwriting fresh search results!

What Topic Is Next → Next Topic 7.22: LocalStorage and SessionStorage

7.22 LocalStorage and SessionStorage [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Origin-scoped synchronous string key-value storage (~5MB) • `localStorage` (persists across tabs/restarts) vs `sessionStorage` (per-tab session) • `QuotaExceededError` & private-browsing `try/catch` • `storage` cross-tab sync event! • UI, Accessibility & Edge-Case Pitfall Guard (LocalStorage and SessionStorage) • Hands-On Frontend Implementation & Verification (LocalStorage and SessionStorage) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for LocalStorage

Purpose: Persisting UI theme preferences, draft form inputs, and Project 2 tasks across page reloads (and syncing across open tabs via `window.addEventListener("storage", ...)`!).

What Topic Is Next → Next Topic 7.23: Browser Cookies Fundamentals

7.23 Browser Cookies Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Automatic HTTP header transmission • `Secure` • `HttpOnly` (invisible to `document.cookie` — XSS-safe!) • `SameSite=Lax|Strict` (CSRF defense!) • UI, Accessibility & Edge-Case Pitfall Guard (Browser Cookies Fundamentals) • Hands-On Frontend Implementation & Verification (Browser Cookies Fundamentals) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Browser Cookies Fundamentals

Purpose: Understanding why enterprise backends store refresh/session tokens in `HttpOnly; Secure; SameSite=Lax` cookies while keeping short-lived access tokens in memory.

What Topic Is Next → Next Topic 7.24: JSON Serialization and Browser Web APIs (Timers, URL, Intl, Clipboard)

7.24 JSON Serialization and Browser Web APIs (Timers, URL, Intl, Clipboard) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `JSON.stringify(val • Null • 2)` / `JSON.parse` • `setTimeout`/`clearTimeout` • UI, Accessibility & Edge-Case Pitfall Guard (JSON Serialization and Browser Web APIs (Timers, URL, Intl, Clipboard)) • Hands-On Frontend Implementation & Verification (JSON Serialization and Browser Web APIs (Timers, URL, Intl, Clipboard)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JSON Serialization

Purpose: Formatting Indian Rupees (`Intl.NumberFormat("en-IN")`) and relative dates natively with 0KB library weight!

What Topic Is Next → Module 7 Capstone (Week 7 Lab Build: Async Weather & GitHub Profile Explorer (`fetch` + `AbortController`)) → Next Topic 8.1: Document Object Model (DOM) Tree Architecture

Module 7 Hands-On Capstone Project (100 Marks): Week 7 Lab Build: Async Weather & GitHub Profile Explorer (`fetch` + `AbortController`) — Build an asynchronous REST API explorer fetching live Weather and GitHub repository metrics using `async/await`, `Promise.all`, `AbortController` request cancellation, skeleton loading, and...

MODULE 8 OF 24 • STAGE 2 • 25 LIVE + LAB HOURS • 19 TOPICS (152 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: DOM, Events and Browser-Based Application Development

Build responsive, accessible, XSS-safe interactive browser applications using the DOM API — querying, `DocumentFragment` node creation, `dataset` & `classList`, Event Bubbling, Capturing, Event Delegation, form validation, keyboard/pointer interactions, History API, and LocalStorage persistence — culminating in Month 2 Mini-Project 2.

Objectives: ✓ Select, create, update, and remove DOM elements cleanly using `textContent` / `createElement` / `DocumentFragment` without XSS vulnerabilities or layout thrashing. • ✓ Master the 3-phase DOM event propagation model (Capturing -> Target -> Bubbling) and attach a single `O(1)` Event Delegation listener for dynamic lists. • ✓ Handle form `submit` (`e.preventDefault()`, `new FormData(form)`), real-time field validation, keyboard shortcuts (`Escape`, `ArrowDown`), and pointer interactions. • ✓ Coordinate URL state with the `History` API (`URLSearchParams`, `pushState`) and accessible focus management. • ✓ Deliver Month 2 Deliverable: Mini-Project 2 — JavaScript Task Management Application.

8.1 Document Object Model (DOM) Tree Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Document -> Element -> Text -> Comment node hierarchy • Live `HTMLCollection` vs static `NodeList` • How state-driven rendering relates to manual DOM mutation • UI, Accessibility & Edge-Case Pitfall Guard (Document Object Model (DOM) Tree Architecture) • Hands-On Frontend Implementation & Verification (Document Object Model (DOM) Tree Architecture) • Document Object Model (DOM) Tree Architecture — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Document Object Model (DOM) Tree Architecture

Purpose: Keeping a clean single source of truth `let tasks = []` in JS and rendering DOM deterministically from state.

What Topic Is Next → Next Topic 8.2: DOM Selection (getElementById, querySelector, querySelectorAll, closest)

8.2 DOM Selection (getElementById, querySelector, querySelectorAll, closest) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fast `document.getElementById` • CSS-powered `querySelector` / `querySelectorAll` • Upward ancestor lookup `el.closest("[data-task-id]")` (essential for Event Delegation!) • UI, Accessibility & Edge-Case Pitfall Guard (DOM Selection (getElementById, querySelector, querySelectorAll, closest)) • Hands-On Frontend Implementation & Verification (DOM Selection (getElementById, querySelector, querySelectorAll, closest)) • DOM Selection (getElementById, querySelector, querySelectorAll, closest) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DOM Selection (getElementById, querySelector, queryS...

Purpose: Using `e.target.closest("button[data-action]")` so clicking an SVG icon inside a button still resolves the `<button>` accurately!

What Topic Is Next → Next Topic 8.3: DOM Manipulation (createElement, textContent, append, replaceChildren, remove)

8.3 DOM Manipulation (createElement, textContent, append, replaceChildren, remove) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `document.createElement` • XSS-safe `el.textContent = userText` (vs dangerous `innerHTML`) • `append` • `replaceChildren` (clears & mounts in 1 call!) • UI, Accessibility & Edge-Case Pitfall Guard (DOM Manipulation (createElement, textContent, append, replaceChildren, remove)) • Hands-On Frontend Implementation & Verification (DOM Manipulation (createElement, textContent, append, replaceChildren, remove)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DOM Manipulation (createElement, textContent, append...

Purpose: Mounting 200 filtered task items in a single browser reflow using `DocumentFragment` + `list.replaceChildren(fragment)`.

What Topic Is Next → Next Topic 8.4: HTML Attributes and dataset Manipulation

8.4 HTML Attributes and dataset Manipulation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `getAttribute`/`setAttribute`/`removeAttribute` • Boolean attributes `el.toggleAttribute("disabled" • IsSaving)` • CamelCase `el.dataset.taskId` (`data-task-id`) • UI, Accessibility & Edge-Case Pitfall Guard (HTML Attributes and dataset Manipulation) • Hands-On Frontend Implementation & Verification (HTML

Attributes and dataset Manipulation) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTML Attributes

Purpose: Storing `data-task-id` and `data-action="delete|toggle|edit"` cleanly on task row controls.

What Topic Is Next → Next Topic 8.5: CSS Class Manipulation with classList

8.5 CSS Class Manipulation with classList [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `classList.add` • `remove` • Conditional `classList.toggle("is-completed" • Task.completed)` (2nd boolean force arg!) • UI, Accessibility & Edge-Case Pitfall Guard (CSS Class Manipulation with classList) • Hands-On Frontend Implementation & Verification (CSS Class Manipulation with classList) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS Class Manipulation with classList

Purpose: Toggling active filter pills, modal open states, and completed task strikethrough classes cleanly.

What Topic Is Next → Next Topic 8.6: Dynamic Styling and CSS Custom Properties from JS

8.6 Dynamic Styling and CSS Custom Properties from JS [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When to prefer `classList` over `el.style.*` • Setting dynamic runtime values cleanly via `el.style.setProperty("--progress" • `${pct}%`)` • `getBoundingClientRect()` • UI, Accessibility & Edge-Case Pitfall Guard (Dynamic Styling and CSS Custom Properties from JS) • Hands-On Frontend Implementation & Verification (Dynamic Styling and CSS Custom Properties from JS) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Styling

Purpose: Driving a task-completion progress bar smoothly via a single `--completion-pct` CSS variable.

What Topic Is Next → Next Topic 8.7: DOM Events (addEventListener, removeEventListener & AbortSignal)

8.7 DOM Events (addEventListener, removeEventListener & AbortSignal) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `addEventListener(type • Handler • { once • Passive • UI, Accessibility & Edge-Case Pitfall Guard (DOM Events (addEventListener, removeEventListener & AbortSignal)) • Hands-On Frontend Implementation & Verification (DOM Events (addEventListener, removeEventListener & AbortSignal)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DOM Events (addEventListener, removeEventListener &...

Purpose: Cleaning up multiple `keydown` + `pointerdown` modal listeners in one call via `controller.abort()`!

What Topic Is Next → Next Topic 8.8: The Event Object (target vs currentTarget, preventDefault)

8.8 The Event Object (target vs currentTarget, preventDefault) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `event.target` (deepest clicked child) vs `event.currentTarget` (element listener is attached to) • `event.preventDefault()` (cancels default navigation/reload • NOT propagation!) • UI, Accessibility & Edge-Case Pitfall Guard (The Event Object (target vs currentTarget, preventDefault)) • Hands-On Frontend Implementation & Verification (The Event Object (target vs currentTarget, preventDefault)) • The Event Object (target vs currentTarget, preventDefault) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for The Event Object (target vs currentTarget, preventDe...

Purpose: Preventing full-page HTTP reload on `<form>` `submit` via `event.preventDefault()`.

What Topic Is Next → Next Topic 8.9: Event Bubbling

8.9 Event Bubbling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Upward propagation `button -> li -> ul -> main -> body -> document -> window` • Which events bubble (`click` • `input` • `keydown` • UI, Accessibility & Edge-Case Pitfall Guard (Event Bubbling) • Hands-On Frontend Implementation & Verification (Event Bubbling) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Event Bubbling

Purpose: Listening to `focusin` (bubbles!) instead of `focus` (does not bubble) when delegating field focus tracking on a `<form>`.

What Topic Is Next → Next Topic 8.10: Event Capturing Phase

8.10 Event Capturing Phase [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Top-down propagation (`window -> document -> ... -> target`) before Bubbling • `{ capture: true }` option • Full 3-phase W3C event flow • UI, Accessibility & Edge-Case Pitfall Guard (Event Capturing Phase) • Hands-On Frontend Implementation & Verification (Event Capturing Phase) • Event Capturing Phase — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Event Capturing Phase

Purpose: Capturing non-bubbling scroll/media events or analytics telemetry before inner handlers run.

What Topic Is Next → Next Topic 8.11: Event Delegation Pattern

8.11 Event Delegation Pattern [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Attaching ONE listener on stable parent `<ul id="task-list">` using `event.target.closest("[data-action]")` — `O(1)` memory • Works automatically for dynamically added items! • UI, Accessibility & Edge-Case Pitfall Guard (Event Delegation Pattern) • Hands-On Frontend Implementation & Verification (Event Delegation Pattern) • Event Delegation Pattern — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Event Delegation Pattern • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Handling Complete, Edit, and Delete actions across 1,000 dynamic tasks with a single `click` listener!

What Topic Is Next → Next Topic 8.12: Form Handling, FormData and Live Validation

8.12 Form Handling, FormData and Live Validation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `submit` event (handles both click • Enter key!) • `Object.fromEntries(new FormData(form))` • `input` vs `change` vs `blur` ("reward early • UI, Accessibility & Edge-Case Pitfall Guard (Form Handling, FormData and Live Validation) • Hands-On Frontend Implementation & Verification (Form Handling, FormData and Live Validation) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Form Handling

Purpose: Showing red validation errors on `blur`/`submit` while clearing the error immediately on `input` as soon as the user fixes it!

What Topic Is Next → Next Topic 8.13: Keyboard Events and Shortcut Interactions

8.13 Keyboard Events and Shortcut Interactions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `keydown` (`event.key === "Escape"` • `"Enter"` • `"ArrowDown"` • `"/"` quick-search focus) • UI, Accessibility & Edge-Case Pitfall Guard (Keyboard Events and Shortcut Interactions) • Hands-On Frontend Implementation & Verification (Keyboard Events and Shortcut Interactions) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Keyboard Events

Purpose: Pressing `/` to focus the Task Search input, `Escape` to cancel inline editing/modals, and `ArrowUp`/`ArrowDown` in autocomplete.

What Topic Is Next → Next Topic 8.14: Mouse and Pointer Events

8.14 Mouse and Pointer Events [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unified `PointerEvent` (`pointerdown` • `pointermove` • `pointerup` — works across mouse • Touch • UI, Accessibility & Edge-Case Pitfall Guard (Mouse and Pointer Events) • Hands-On Frontend Implementation & Verification (Mouse and Pointer Events) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Mouse

Purpose: Implementing drag handles, tooltips, and double-click-to-edit task titles cleanly.

What Topic Is Next → Next Topic 8.15: Practical Browser Storage Sync for State Persistence

8.15 Practical Browser Storage Sync for State Persistence [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Hydrating initial state on `DOMContentLoaded` from `localStorage` • Persisting every mutation with schema version `{ version: 1 • Tasks: [...] }` • JSON import/export backup • UI, Accessibility & Edge-Case Pitfall Guard (Practical Browser Storage Sync for State Persistence) • Hands-On Frontend Implementation & Verification (Practical Browser Storage Sync for State Persistence) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Practical Browser Storage Sync for State Persistence

Purpose: Persisting all tasks, priorities, due dates, and active filter tabs seamlessly in Mini-Project 2.

What Topic Is Next → Next Topic 8.16: History API and URL Query State Synchronization

8.16 History API and URL Query State Synchronization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `new URL(location.href)` • `searchParams.set("status" • "pending")` • `history.replaceState` / `pushState` • UI, Accessibility & Edge-Case Pitfall Guard (History API and URL Query State Synchronization) • Hands-On Frontend Implementation & Verification (History API and URL Query State Synchronization) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for History API

Purpose: Syncing `?filter=completed&q=deploy` into the browser URL so refreshing or sharing the link preserves the exact filtered view!

What Topic Is Next → Next Topic 8.17: JavaScript-Driven Responsive UI Interactions

8.17 JavaScript-Driven Responsive UI Interactions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `window.matchMedia("(min-width: 48rem)").addEventListener("change" • ...)` (fires ONLY when crossing the breakpoint • Not 60x/sec like `resize`!) • `ResizeObserver` • UI, Accessibility & Edge-Case Pitfall Guard (JavaScript-Driven Responsive UI Interactions) • Hands-On Frontend Implementation & Verification (JavaScript-Driven Responsive UI Interactions) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JavaScript-Driven Responsive UI Interactions

Purpose: Auto-closing the mobile hamburger drawer cleanly when the viewport crosses `>= 768px` via `matchMedia`.

What Topic Is Next → Next Topic 8.18: Accessible DOM Controls & Focus Management

8.18 Accessible DOM Controls & Focus Management [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Returning focus to the trigger button when closing a modal • Focus trapping inside open dialogs • `role="status"` / `aria-live="polite"` screen-reader announcements (`"Task added"` • `"3 results found"`) • UI, Accessibility & Edge-Case Pitfall Guard (Accessible DOM Controls & Focus Management) • Hands-On Frontend Implementation & Verification (Accessible DOM Controls & Focus Management) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Accessible DOM Controls

Purpose: Moving focus to the next task or Add input after deleting a task instead of dropping keyboard focus back to `<body>`!

What Topic Is Next → Next Topic 8.19: DOM Performance (Batch Updates, Debounced Search & Reflow Prevention)

8.19 DOM Performance (Batch Updates, Debounced Search & Reflow Prevention) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Debouncing live search input (`200ms`) • Separating DOM measurement reads from mutation writes • `requestAnimationFrame` visual updates • UI, Accessibility & Edge-Case Pitfall Guard (DOM Performance (Batch Updates, Debounced Search & Reflow Prevention)) • Hands-On Frontend Implementation & Verification (DOM Performance (Batch Updates, Debounced Search & Reflow Prevention)) • DOM Performance (Batch Updates, Debounced Search & Reflow Prevention) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DOM Performance (Batch Updates, Debounced Search & R...

Purpose: Ensuring instant 60fps typing in Mini-Project 2 even with 500+ saved tasks.

What Topic Is Next → Module 8 Capstone (Project 2: JavaScript Task Management Application (ES6+, DOM & LocalStorage)) → Next Topic 9.1: TypeScript Introduction (JavaScript vs TypeScript)

Module 8 Hands-On Capstone Project (100 Marks): Project 2: JavaScript Task Management Application (ES6+, DOM & LocalStorage) — Build a production-grade, zero-framework Vanilla JavaScript ES6+ Task & Sprint Management Application supporting full CRUD, priority/category tagging, due-date status indicators, debounced r...

MODULE 9 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 27 TOPICS (216 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: TypeScript Complete Fundamentals

Write fearless, self-documenting TypeScript 5 code — `strict: true` compiler configuration, primitives, `unknown` vs `any` vs `never`, tuples, interfaces vs type aliases, discriminated unions, type narrowing & custom type predicates, generics & `keyof` constraints, built-in utility types (`Partial`, `Pick`, `Omit`, `Record`), conditional/mapped types, classes, `.d.ts` files, and migrating JS apps to TypeScript.

Objectives: ✓ Configure `tsconfig.json` with `"strict": true` (`strictNullChecks`, `noImplicitAny`, `noUncheckedIndexedAccess`) and understand zero-runtime type erasure. • ✓ Choose `unknown` + type narrowing over `any`, and model impossible branches exhaustively with `never` + discriminated unions. • ✓ Define clean domain contracts with `interface` and `type`, generics (`ApiResponse<T>`, `Paginated<T>`), and `extends keyof` constraints. • ✓ Derive non-redundant types using `Partial`, `Required`, `Pick`, `Omit`, `Record`, `Readonly`, `ReturnType`, and `Parameters`. • ✓ Migrate an existing JavaScript module into strict TypeScript with zero `any` escapes.

9.1 TypeScript Introduction (JavaScript vs TypeScript) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Compile-time structural type checking • IDE refactoring superpowers • Complete type erasure at runtime (browsers still run emitted JS!) • UI, Accessibility & Edge-Case Pitfall Guard (TypeScript Introduction (JavaScript vs TypeScript)) • Hands-On Frontend Implementation & Verification (TypeScript Introduction (JavaScript vs TypeScript)) • TypeScript Introduction (JavaScript vs TypeScript) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TypeScript Introduction (JavaScript vs TypeScript)

Purpose: Catching `Cannot read properties of undefined` and misspelled API prop names in the editor *before* saving the file.

What Topic Is Next → Next Topic 9.2: TypeScript Environment Setup (Node.js, npm & Vite)

9.2 TypeScript Environment Setup (Node.js, npm & Vite) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `npm install -D typescript` • `npx tsc --noEmit` typecheck script in `package.json` • Vite fast esbuild transpilation + `tsc` type checking • UI, Accessibility & Edge-Case Pitfall Guard (TypeScript Environment Setup (Node.js, npm & Vite)) • Hands-On Frontend Implementation & Verification (TypeScript Environment Setup (Node.js, npm & Vite)) • TypeScript Environment Setup (Node.js, npm & Vite) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TypeScript Environment Setup (Node.js, npm & Vite)

Purpose: Adding `"typecheck": "tsc --noEmit"` to `package.json` CI scripts because Vite's dev bundler strips types without failing on type errors!

What Topic Is Next → Next Topic 9.3: tsconfig.json and Strict Compiler Options

9.3 tsconfig.json and Strict Compiler Options [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `"strict": true` • `"target": "ES2022"` • `"moduleResolution": "bundler"` • `"noUncheckedIndexedAccess": true` • UI, Accessibility & Edge-Case Pitfall Guard (tsconfig.json and Strict Compiler Options) • Hands-On Frontend Implementation & Verification (tsconfig.json and Strict Compiler Options) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for tsconfig.json

Purpose: Enabling `strict` from Day 1 so `null` and `undefined` are always tracked explicitly.

What Topic Is Next → Next Topic 9.4: Basic Primitive Types (string, number, boolean, null, undefined, bigint, symbol)

9.4 Basic Primitive Types (string, number, boolean, null, undefined, bigint, symbol) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lowercase primitive type names (`string` • NOT wrapper `String`!) • `strictNullChecks` (`string | null`) • Optional `?` (`T | undefined`) • UI, Accessibility & Edge-Case Pitfall Guard (Basic Primitive Types (string, number, boolean, null, undefined, bigint, symbol)) • Hands-On Frontend Implementation & Verification (Basic Primitive Types (string, number, boolean, null, undefined, bigint, symbol)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Basic Primitive Types (string, number, boolean, null...

Purpose: Typing nullable API fields (`avatarUrl: string | null`) accurately.

What Topic Is Next → Next Topic 9.5: Special Types (any, unknown, never and void)

9.5 Special Types (any, unknown, never and void) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `any` disables the type checker (avoid!) • Safe top-type `unknown` (requires narrowing before use! — ideal for `catch (err: unknown)` & raw JSON) • Bottom-type `never` (exhaustive switch checks!) • `void` • UI, Accessibility & Edge-Case Pitfall Guard (Special Types (any, unknown, never and void)) • Hands-On Frontend Implementation & Verification (Special Types (any, unknown, never and void)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Special Types (any, unknown, never and void)

Purpose: Typing `catch (err: unknown)` safely (`err instanceof Error ? err.message : String(err)`) and enforcing exhaustive switch statements with `const _exhaustive: never = stat...

What Topic Is Next → Next Topic 9.6: Typed Arrays and ReadonlyArray<T>

9.6 Typed Arrays and ReadonlyArray<T> [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `string[]` vs `Array<string>` • Immutable `readonly Product[]` / `ReadonlyArray<Product>` preventing accidental `.push()`/`.sort()` mutation on props! • UI, Accessibility & Edge-Case Pitfall Guard (Typed Arrays and ReadonlyArray<T>) • Hands-On Frontend Implementation & Verification (Typed Arrays and ReadonlyArray<T>) • Typed Arrays and ReadonlyArray<T> — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Typed Arrays • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Typing React list props as `items: readonly Employee[]` so the compiler blocks accidental in-place `.sort()`!

What Topic Is Next → Next Topic 9.7: Tuples and Labeled Tuples

9.7 Tuples and Labeled Tuples [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fixed-length positional types `[lat: number • Lng: number]` • `readonly [T • (next: T) => void]` custom hook return tuples (`as const`)! • UI, Accessibility & Edge-Case Pitfall Guard (Tuples and Labeled Tuples) • Hands-On Frontend Implementation & Verification (Tuples and Labeled Tuples) • Idiomatic FDCP

Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Tuples

Purpose: Returning `return [isOpen, toggle] as const;` from a custom React `useToggle` hook so callers get `[boolean, () => void]` instead of `(boolean | (() => void))[]`!

What Topic Is Next → Next Topic 9.8: Enums vs Const String Literal Unions

9.8 Enums vs Const String Literal Unions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Numeric enums (reverse-mapping pitfalls!) • String `enum OrderStatus` • `const enum` caveats with bundlers • Why `const STATUS = { ... } as const` + union `type Status = ...` is preferred in modern TS • UI, Accessibility & Edge-Case Pitfall Guard (Enums vs Const String Literal Unions) • Hands-On Frontend Implementation & Verification (Enums vs Const String Literal Unions) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Enums vs Const String Literal Unions

Purpose: Defining zero-runtime-overhead `type Role = "ADMIN" | "MANAGER" | "EMPLOYEE"` unions while understanding Angular/enterprise string `enum`s.

What Topic Is Next → Next Topic 9.9: Type Inference

9.9 Type Inference [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Contextual typing in `.map((emp) => ...)` • Event listeners • `let` widening (`"asc"` -> `string`) vs `const` literal narrowing (`"asc"`) • `as const` deep readonly literals • UI, Accessibility & Edge-Case Pitfall Guard (Type Inference) • Hands-On Frontend Implementation & Verification (Type Inference) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Type Inference

Purpose: Letting TypeScript infer local variables cleanly while explicitly typing public function boundaries and state.

What Topic Is Next → Next Topic 9.10: Explicit Type Annotations (When Required vs Redundant)

9.10 Explicit Type Annotations (When Required vs Redundant) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Annotating function parameters (always!) • Public return types (catches accidental return-shape drift!) • Empty arrays `const items: Product[] = []` • Nullable state `User | null` • UI, Accessibility & Edge-Case Pitfall Guard (Explicit Type Annotations (When Required vs Redundant)) • Hands-On Frontend Implementation & Verification (Explicit Type Annotations (When Required vs Redundant)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Explicit Type Annotations (When Required vs Redundan...

Purpose: Adding explicit return types on exported API service functions so internal refactors never break callers silently.

What Topic Is Next → Next Topic 9.11: Typed Functions, Optional/Default Params and Overloads

9.11 Typed Functions, Optional/Default Params and Overloads [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Function type expressions `(q: string) => Promise<Product[]>` • Optional `page?: number` • Callback `void` return rule • Function overloads • UI, Accessibility & Edge-Case Pitfall Guard (Typed Functions, Optional/Default Params and Overloads) • Hands-On Frontend Implementation & Verification (Typed Functions, Optional/Default Params and Overloads) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Typed Functions

Purpose: Typing React callback props (`onSelect: (employeeId: string) => void`) and higher-order wrappers.

What Topic Is Next → Next Topic 9.12: Object Types, Optional (?) and readonly Modifiers

9.12 Object Types, Optional (?) and readonly Modifiers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Structural duck typing • Excess property checking on fresh object literals • `readonly id: string` • Index signatures `[key: string]: number` • UI, Accessibility & Edge-Case Pitfall Guard (Object Types, Optional (?) and readonly Modifiers) • Hands-On Frontend Implementation & Verification (Object Types, Optional (?) and readonly Modifiers) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Object Types

Purpose: Marking `readonly id: string` and `readonly createdAt: string` on domain models so UI code never overwrites identifiers.

What Topic Is Next → Next Topic 9.13: Interfaces and Extends Inheritance

9.13 Interfaces and Extends Inheritance [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `interface Employee extends Person • Auditable` • Declaration merging (extending global `Window` or Express/Vite `ImportMetaEnv`!) • Clean error messages in IDEs • UI, Accessibility & Edge-Case Pitfall Guard (Interfaces and Extends Inheritance) • Hands-On Frontend Implementation & Verification (Interfaces and Extends Inheritance) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interfaces

Purpose: Extending `ImportMetaEnv` in `vite-env.d.ts` (`readonly VITE_API_BASE_URL: string`) and composing React component `Props` interfaces.

What Topic Is Next → Next Topic 9.14: Type Aliases (type) vs Interfaces

9.14 Type Aliases (type) vs Interfaces [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When ONLY `type` works: unions (`A | B`) • Primitives • Tuples • Mapped & conditional types • UI, Accessibility & Edge-Case Pitfall Guard (Type Aliases (type) vs Interfaces) • Hands-On Frontend Implementation & Verification (Type Aliases (type) vs Interfaces) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Type Aliases (type) vs Interfaces

Purpose: Using `interface` for object/prop shapes and `type` for unions, state machines, and utility transformations.

What Topic Is Next → Next Topic 9.15: Union Types (|) and Discriminated (Tagged) Unions

9.15 Union Types (|) and Discriminated (Tagged) Unions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Discriminant literal tag (`status: "loading" | "success" | "error"`) • How checking `if (state.status === "success")` automatically narrows `state.data` without optional `?`! • UI, Accessibility & Edge-Case Pitfall Guard (Union Types (|) and Discriminated (Tagged) Unions) • Hands-On Frontend Implementation & Verification (Union Types (|) and Discriminated (Tagged) Unions) • Union Types (|) and Discriminated (Tagged) Unions — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Union Types (|) • Production Hardening & Security: input validation, least-privilege access

& OWASP safeguards

Purpose: Eliminating impossible `{ isLoading: true, error: "Failed", data: [...] }` boolean-soup states in React and Angular!

What Topic Is Next → Next Topic 9.16: Intersection Types (&)

9.16 Intersection Types (&) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Combining orthogonal capabilities `type WithPagination<T> = T & { page: number • PageSize: number }` • Conflicting property type -> `never`! • UI, Accessibility & Edge-Case Pitfall Guard (Intersection Types (&)) • Hands-On Frontend Implementation & Verification (Intersection Types (&)) • Intersection Types (&) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Intersection Types (&)

Purpose: Composing `EmployeeSearchFilter & PaginationParams` cleanly for API query hooks.

What Topic Is Next → Next Topic 9.17: Literal Types and Template Literal Types

9.17 Literal Types and Template Literal Types [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Exact string/number/boolean literals (`size: "sm" | "md" | "lg"`) • Template literal types `` `/${string}` `` or `` `${Axis}-${Align}` `` • UI, Accessibility & Edge-Case Pitfall Guard (Literal Types and Template Literal Types) • Hands-On Frontend Implementation & Verification (Literal Types and Template Literal Types) • Literal Types and Template Literal Types — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Literal Types • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Autocompleting valid design-system `<Button variant="primary" size="md" />` props in VS Code.

What Topic Is Next → Next Topic 9.18: Type Narrowing (typeof, instanceof, in and Custom Type Guards)

9.18 Type Narrowing (typeof, instanceof, in and Custom Type Guards) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Control-flow narrowing via `typeof` • `instanceof` • `"data" in res` • `Array.isArray` • UI, Accessibility & Edge-Case Pitfall Guard (Type Narrowing (typeof, instanceof, in and Custom Type Guards)) • Hands-On Frontend Implementation & Verification (Type Narrowing (typeof, instanceof, in and Custom Type Guards)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Type Narrowing (typeof, instanceof, in and Custom Ty...

Purpose: Filtering nullable arrays cleanly via `.filter((x): x is Product => x !== null)`!

What Topic Is Next → Next Topic 9.19: Type Assertions (as) vs the satisfies Operator

9.19 Type Assertions (as) vs the satisfies Operator [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `as User` lies to the compiler if JSON doesn't match • Non-null `!` danger (avoid!) • TS 4.9+ `satisfies` (validates type compliance WITHOUT widening literal keys/values!) • UI, Accessibility & Edge-Case Pitfall Guard (Type Assertions (as) vs the satisfies Operator) • Hands-On Frontend Implementation & Verification (Type Assertions (as) vs the satisfies Operator) • Type Assertions (as) vs the satisfies Operator — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Type Assertions (as) vs the satisfies Operator

Purpose: Validating a route/theme config map with `satisfies Record<RouteKey, RouteMeta>` while keeping exact literal types intact!

What Topic Is Next → Next Topic 9.20: Generics (Functions, Interfaces and Classes)

9.20 Generics (Functions, Interfaces and Classes) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Parameterized reusable contracts `ApiResponse<T>` • `PaginatedList<T>` • `groupBy<T • K extends PropertyKey>(items: readonly T[] • UI, Accessibility & Edge-Case Pitfall Guard (Generics (Functions, Interfaces and Classes)) • Hands-On Frontend Implementation & Verification (Generics (Functions, Interfaces and Classes)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Generics (Functions, Interfaces and Classes)

Purpose: Writing ONE generic `fetchJson<T>(url)` and `PaginatedResponse<T>` shared across Employees, Products, and Orders!

What Topic Is Next → Next Topic 9.21: Generic Constraints (extends and keyof)

9.21 Generic Constraints (extends and keyof) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `function pluck<T • K extends keyof T>(items: readonly T[] • Key: K): T[K][]` • Constraining `<T extends { id: string }>` • UI, Accessibility & Edge-Case Pitfall Guard (Generic Constraints (extends and keyof)) • Hands-On Frontend Implementation & Verification (Generic Constraints (extends and keyof)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Generic Constraints (extends and keyof)

Purpose: Building a type-safe `<DataTable<Row> columns={...} />` where column `accessorKey` MUST be a valid `keyof Row`!

What Topic Is Next → Next Topic 9.22: Built-in Utility Types (Partial, Required, Pick, Omit, Record, Readonly, ReturnType)

9.22 Built-in Utility Types (Partial, Required, Pick, Omit, Record, Readonly, ReturnType) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Deriving `CreateEmployeePayload = Omit<Employee • "id" | "createdAt">` • `UpdateEmployeePayload = Partial<CreateEmployeePayload>` • `Pick` summary cards • UI, Accessibility & Edge-Case Pitfall Guard (Built-in Utility Types (Partial, Required, Pick, Omit, Record, Readonly, ReturnType)) • Hands-On Frontend Implementation & Verification (Built-in Utility Types (Partial, Required, Pick, Omit, Record, Readonly, ReturnType)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Built-in Utility Types (Partial, Required, Pick, Omi...

Purpose: Deriving Create, Patch, and Card-Preview DTOs directly from the canonical `Employee` interface so changes stay 100% synchronized!

What Topic Is Next → Next Topic 9.23: Advanced Types (Indexed Access, Mapped and Conditional Types)

9.23 Advanced Types (Indexed Access, Mapped and Conditional Types) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Indexed lookup `Employee["department"]` • Mapped `{ [K in keyof T]: boolean }` (e.g. `FormTouched<T>` & `FormErrors<T>`!) • Conditional `T extends Promise<infer U> ? U : T` (`Awaited<T>`) • UI, Accessibility & Edge-Case Pitfall Guard (Advanced Types (Indexed Access, Mapped and Conditional Types))

• Hands-On Frontend Implementation & Verification (Advanced Types (Indexed Access, Mapped and Conditional Types)) • Advanced Types (Indexed Access, Mapped and Conditional Types) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Advanced Types (Indexed Access, Mapped and Condition...

Purpose: Generating `type FormErrors<Values> = Partial<Record<keyof Values, string>>` automatically for any form!

What Topic Is Next → Next Topic 9.24: TypeScript Modules and Type-Only Imports (import type)

9.24 TypeScript Modules and Type-Only Imports (import type) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `import type { Employee } from "./types"` / `import { type Employee • FormatSalary }` • Guaranteeing zero runtime circular-import side effects (`verbatimModuleSyntax`) • UI, Accessibility & Edge-Case Pitfall Guard (TypeScript Modules and Type-Only Imports (import type)) • Hands-On Frontend Implementation & Verification (TypeScript Modules and Type-Only Imports (import type)) • TypeScript Modules and Type-Only Imports (import type) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TypeScript Modules

Purpose: Using `import type` cleanly across React and Angular components so bundlers drop type-only imports immediately.

What Topic Is Next → Next Topic 9.25: TypeScript Classes, Abstract Classes and Parameter Properties

9.25 TypeScript Classes, Abstract Classes and Parameter Properties [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing interfaces (`class HttpEmployeeRepo implements EmployeeRepo`) • `public`/`private`/`protected`/`readonly` constructor parameter properties (used heavily in Angular DI!) • `abstract class` • UI, Accessibility & Edge-Case Pitfall Guard (TypeScript Classes, Abstract Classes and Parameter Properties) • Hands-On Frontend Implementation & Verification (TypeScript Classes, Abstract Classes and Parameter Properties) • TypeScript Classes, Abstract Classes and Parameter Properties — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TypeScript Classes

Purpose: Writing concise Angular services (`constructor(private readonly http: HttpClient) {}`) and repository adapters.

What Topic Is Next → Next Topic 9.26: Using JavaScript Libraries & Ambient Declaration Files (.d.ts)

9.26 Using JavaScript Libraries & Ambient Declaration Files (.d.ts) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `DefinitelyTyped` (`@types/node` • `@types/react`) • Authoring `vite-env.d.ts` / `ambient.d.ts` (`declare module "*.svg"` • `Window` analytics globals) • UI, Accessibility & Edge-Case Pitfall Guard (Using JavaScript Libraries & Ambient Declaration Files (.d.ts)) • Hands-On Frontend Implementation & Verification (Using JavaScript Libraries & Ambient Declaration Files (.d.ts)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Using JavaScript Libraries

Purpose: Typing `import.meta.env.VITE_API_URL` and third-party scripts cleanly without `@ts-ignore`!

What Topic Is Next → Next Topic 9.27: TypeScript Debugging, Source Maps and JS-to-TS Migration

9.27 TypeScript Debugging, Source Maps and JS-to-TS Migration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reading bottom-most nested TS assignability error lines first • Browser `sourceMap: true` original `.ts` breakpoint debugging • Incremental file-by-file `.js -> .ts` migration • UI, Accessibility & Edge-Case Pitfall Guard (TypeScript Debugging, Source Maps and JS-to-TS Migration) • Hands-On Frontend Implementation & Verification (TypeScript Debugging, Source Maps and JS-to-TS Migration) • TypeScript Debugging, Source Maps and JS-to-TS Migration — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TypeScript Debugging

Purpose: Migrating Month 2's JavaScript Task Management Application cleanly into 100% strict TypeScript!

What Topic Is Next → Module 9 Capstone (Week 9 Lab Build: Strictly Typed TypeScript 5 Domain SDK (`strict: true`)) → Next Topic 10.1: React Introduction and Declarative Component Architecture

Module 9 Hands-On Capstone Project (100 Marks): Week 9 Lab Build: Strictly Typed TypeScript 5 Domain SDK (`strict: true`) — Author a zero-`any` TypeScript 5 commerce & HR domain library modeling discriminated `ApiResult<T>` unions, generic repository helpers, type predicates, and utility types (`Partial`, `Pick`,...

MODULE 10 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 20 TOPICS (160 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: React.js Fundamentals and Application Architecture

Enter primary React specialization with Vite + React + TypeScript — declarative UI = f(state), JSX rules & XSS escaping, functional components, composition (`children`), typed props, conditional rendering (`0` pitfall!), list keys & reconciliation, synthetic events, `useState`, immutable state updates, controlled vs uncontrolled inputs, lifecycle mental model, CSS Modules/Tailwind styling, React DevTools, and feature-based architecture.

Objectives: ✓ Scaffold Vite + React + TypeScript apps and think declaratively (`UI = f(state)`) instead of mutating DOM nodes manually. • ✓ Compose small, single-responsibility functional components with strictly typed `Props` and `children` slots. • ✓ Render conditional UI (avoiding `{count && <Badge />}` rendering `"0"`!) and lists with stable unique `key={item.id}` identifiers. • ✓ Manage component state immutably with `useState`, functional updaters `setCount(c => c + 1)`, and controlled inputs. • ✓ Organize projects using scalable `features/*` + `components/ui/*` architecture and inspect renders in React Developer Tools.

10.1 React Introduction and Declarative Component Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Declarative `UI = f(state)` vs imperative DOM surgery • Virtual DOM tree diffing • One-way data flow • React 18/19 ecosystem • UI, Accessibility & Edge-Case Pitfall Guard (React Introduction and Declarative Component Architecture) • Hands-On Frontend Implementation & Verification (React Introduction and Declarative Component Architecture) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Introduction

Purpose: Building complex interactive web applications where UI never drifts out of sync with underlying data.

What Topic Is Next → Next Topic 10.2: Vite + React + TypeScript Project Setup

10.2 Vite + React + TypeScript Project Setup [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `npm create vite@latest app -- --template react-ts` • `createRoot(document.getElementById("root")!).render(<StrictMode><App /></StrictMode>)` • Why `<StrictMode>` double-invokes in dev to catch impure bugs! • UI, Accessibility & Edge-Case Pitfall Guard (Vite + React + TypeScript Project Setup) • Hands-On Frontend Implementation & Verification (Vite + React + TypeScript Project Setup) • Vite + React + TypeScript Project Setup — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Vite + React + TypeScript Project Setup

Purpose: Keeping `<StrictMode>` enabled so impure renders and missing `useEffect` cleanups are caught immediately during local development.

What Topic Is Next → Next Topic 10.3: React Application Structure (src, public, assets & Entry Points)

10.3 React Application Structure (src, public, assets & Entry Points) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `index.html` mount root • Hashed bundled imports from `src/assets` vs static unhashed root files in `public/` (`robots.txt` • `favicon.svg`) • UI, Accessibility & Edge-Case Pitfall Guard (React Application Structure (src, public, assets & Entry Points)) • Hands-On Frontend Implementation & Verification (React Application Structure (src, public, assets & Entry Points)) • React Application Structure (src, public, assets & Entry Points) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Application Structure (src, public, assets & E...

Purpose: Importing UI icons/illustrations from `src/assets` so Vite hashes filenames for infinite CDN caching.

What Topic Is Next → Next Topic 10.4: JSX Syntax, Expressions and Automatic XSS Escaping

10.4 JSX Syntax, Expressions and Automatic XSS Escaping [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single root / `<>...</>` fragment rule • `className` & `htmlFor` • `{jsExpression}` (expressions • Not `if`/`for` statements!) • UI, Accessibility & Edge-Case Pitfall Guard (JSX Syntax, Expressions and Automatic XSS Escaping) • Hands-On Frontend Implementation & Verification (JSX Syntax, Expressions and Automatic XSS Escaping) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JSX Syntax

Purpose: Safely rendering untrusted user comments `{comment.body}` because JSX escapes `<script>` tags as harmless text automatically.

What Topic Is Next → Next Topic 10.5: Functional Components and Purity Rules

10.5 Functional Components and Purity Rules [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: PascalCase naming (`function ProductCard()`) • Pure rendering (same props & state -> same JSX • Zero side effects during render!) • NEVER defining a component function *inside* another component! • UI, Accessibility & Edge-Case Pitfall Guard (Functional Components and Purity Rules) • Hands-On Frontend Implementation & Verification (Functional Components and Purity Rules) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Functional Components

Purpose: Keeping every component deterministic and fast to test.

What Topic Is Next → Next Topic 10.6: Component Composition

10.6 Component Composition [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Assembling complex pages from small focused `<PageHeader>` • `<FilterBar>` • `<DataGrid>` • `<ModalShell>` building blocks • UI, Accessibility & Edge-Case Pitfall Guard (Component Composition) • Hands-On Frontend Implementation & Verification (Component Composition) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Component Composition

Purpose: Keeping every component file `< 150` lines with a single clear responsibility.

What Topic Is Next → Next Topic 10.7: Props and Unidirectional Data Flow

10.7 Props and Unidirectional Data Flow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Read-only props passed parent -> child • Lifting shared state to the closest common parent • Child-to-parent communication via callback props `onDelete(id)` • UI, Accessibility & Edge-Case Pitfall Guard (Props and Unidirectional Data Flow) • Hands-On Frontend Implementation & Verification (Props and Unidirectional Data Flow) • Props and Unidirectional Data Flow — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Props

Purpose: Lifting `searchQuery` up so `<SearchBar value={q} onChange={setQ} />` and `<ResultsList query={q} />` stay synchronized.

What Topic Is Next → Next Topic 10.8: TypeScript Props Validation and Default Values

10.8 TypeScript Props Validation and Default Values [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Typing `interface ButtonProps extends ComponentPropsWithoutRef<"button">` • Destructuring defaults `{ variant = "primary" • IsLoading = false }: ButtonProps` • UI, Accessibility & Edge-Case Pitfall Guard (TypeScript Props Validation and Default Values) • Hands-On Frontend Implementation & Verification (TypeScript Props Validation and Default Values) • TypeScript Props Validation and Default Values — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TypeScript Props Validation

Purpose: Extending native HTML `<button>` / `<input>` attributes via `ComponentPropsWithoutRef<"button">` so `type`, `disabled`, `aria-label`, and `onClick` all work with full aut...

What Topic Is Next → Next Topic 10.9: The children Prop and Slot Composition

10.9 The children Prop and Slot Composition [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `children: React.ReactNode` • Named JSX slot props (`header?: React.ReactNode` • `footer?: React.ReactNode`) • Eliminating prop drilling via composition! • UI, Accessibility & Edge-Case Pitfall Guard (The children Prop and Slot Composition) • Hands-On Frontend Implementation & Verification (The children Prop and Slot Composition) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for The children Prop

Purpose: Building flexible `<Card header={...} footer={...}>{children}</Card>` and `<Modal>` containers that wrap any content.

What Topic Is Next → Next Topic 10.10: Conditional Rendering Patterns (Avoiding the 0 Trap!)

10.10 Conditional Rendering Patterns (Avoiding the 0 Trap!) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Ternary `isOpen ? <Panel /> : null` • Early `if (!user) return <Skeleton />` • Why `{items.length && <List />}` renders a literal `"0"` on screen when `items` is empty (always write `{items.length > 0 && <List />}`)! • UI, Accessibility & Edge-Case Pitfall Guard (Conditional Rendering Patterns (Avoiding the 0 Trap!)) • Hands-On Frontend Implementation & Verification (Conditional Rendering Patterns (Avoiding the 0 Trap!)) • Conditional Rendering Patterns (Avoiding the 0 Trap!) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Conditional Rendering Patterns (Avoiding the 0 Trap!...

Purpose: Rendering clean Loading / Error / Empty / Data states without stray `"0"` characters in JSX.

What Topic Is Next → Next Topic 10.11: Rendering Lists with .map()

10.11 Rendering Lists with .map() [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mapping typed arrays to JSX elements • Empty-array fallback `<EmptyState />` • Derived filtered/sorted arrays before return • UI, Accessibility & Edge-Case Pitfall Guard (Rendering Lists with .map()) • Hands-On Frontend Implementation & Verification (Rendering Lists with .map()) • Rendering Lists with .map() — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Rendering Lists with .map()

Purpose: Rendering filtered employee cards and table rows cleanly from typed arrays.

What Topic Is Next → Next Topic 10.12: Unique Keys and React Reconciliation

10.12 Unique Keys and React Reconciliation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why React needs stable sibling `key={item.id}` (or `crypto.randomUUID()` created at item-creation time • NEVER `Math.random()` during render!) • Why `key={index}` breaks state/inputs when reordering or deleting • UI, Accessibility & Edge-Case Pitfall Guard (Unique Keys and React Reconciliation) • Hands-On Frontend Implementation & Verification (Unique Keys and React Reconciliation) • Unique Keys and React Reconciliation — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Unique Keys

Purpose: Preserving focus, animations, and internal state accurately when filtering, sorting, or deleting rows.

What Topic Is Next → Next Topic 10.13: React Synthetic Events and Typed Event Handlers

10.13 React Synthetic Events and Typed Event Handlers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `React.MouseEvent<HTMLButtonElement>` • `React.ChangeEvent<HTMLInputElement>` • `React.FormEvent<HTMLFormElement>` • Passing arguments cleanly `onClick={() => onRemove(item.id)}` • UI, Accessibility & Edge-Case Pitfall Guard (React Synthetic Events and Typed Event Handlers) • Hands-On Frontend Implementation & Verification (React Synthetic Events and Typed Event Handlers) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Synthetic Events

Purpose: Writing strictly typed `handleSubmit` and `handleInputChange` handlers without `: any`.

What Topic Is Next → Next Topic 10.14: Component State vs Local Variables & Derived State

10.14 Component State vs Local Variables & Derived State [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why mutating `let count = 0` doesn't trigger a re-render • Why derived values (`const filtered = ...` • `const fullName = first + " " + last`) MUST be computed during render instead of duplicated into redundant `useState`! • UI, Accessibility & Edge-Case Pitfall Guard (Component State vs Local Variables & Derived State) • Hands-On Frontend Implementation & Verification (Component State vs Local Variables & Derived State) • Component State vs Local Variables & Derived State — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Component State vs Local Variables

Purpose: Keeping state minimal (`items`, `filter`, `query`) and deriving `visibleItems` and `completedCount` on the fly with zero sync bugs!

What Topic Is Next → Next Topic 10.15: useState Hook Mastery

10.15 useState Hook Mastery [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `const [user • SetUser] = useState<User | null>(null)` • Lazy initializer `useState(() => readExpensiveStorage())` (runs ONLY on initial mount • Not every render!) • UI, Accessibility & Edge-Case Pitfall Guard (useState Hook Mastery) • Hands-On Frontend Implementation & Verification (useState Hook Mastery) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for useState Hook Mastery

Purpose: Passing a lazy initializer function `useState(() => loadSavedCart())` so `JSON.parse(localStorage...)` runs once on mount instead of every keystroke!

What Topic Is Next → Next Topic 10.16: Immutable State Updates & Functional Updaters

10.16 Immutable State Updates & Functional Updaters [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `state.push(x) • SetState(state)` triggers ZERO re-render (`Object.is(prev • Next)` sees the exact same reference!) • Functional updater `setCount(prev => prev + 1)` • UI, Accessibility & Edge-Case Pitfall Guard (Immutable State Updates & Functional Updaters) • Hands-On Frontend Implementation & Verification (Immutable State Updates & Functional Updaters) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Immutable State Updates

Purpose: Updating quantity, removing items, and incrementing counters reliably even across multiple batched updates.

What Topic Is Next → Next Topic 10.17: Controlled Components (Single Source of Truth Inputs)

10.17 Controlled Components (Single Source of Truth Inputs) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Binding `value={email}` + `onChange={e => setEmail(e.target.value)}` • `checked={agreed}` on checkboxes • `<select value={role}>` • Avoiding `"changing an uncontrolled input to be controlled"` warning (`value={val ?? ""}`!) • UI, Accessibility & Edge-Case Pitfall Guard (Controlled Components (Single Source of Truth Inputs)) • Hands-On Frontend Implementation & Verification (Controlled Components (Single Source of Truth Inputs)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Controlled Components (Single Source of Truth Inputs...

Purpose: Instant input formatting, character counters, conditional fields, and live search bars.

What Topic Is Next → Next Topic 10.18: Uncontrolled Components and FormData / Ref Access

10.18 Uncontrolled Components and FormData / Ref Access [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `defaultValue` / `defaultChecked` • Reading `new FormData(e.currentTarget)` on submit • File inputs (`<input type="file">` is ALWAYS uncontrolled in React!) • UI, Accessibility & Edge-Case Pitfall Guard (Uncontrolled Components and FormData / Ref Access) • Hands-On Frontend Implementation & Verification (Uncontrolled Components and FormData / Ref Access) • Uncontrolled Components and FormData / Ref Access — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Uncontrolled Components

Purpose: Handling `<input type="file">` avatar/resume uploads and simple non-reactive forms with zero keystroke re-renders.

What Topic Is Next → Next Topic 10.19: Component Lifecycle Concepts (Mount, Update and Unmount)

10.19 Component Lifecycle Concepts (Mount, Update and Unmount) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How functional components mount (initial render + commit) • Update (state/prop change -> re-render + commit) • Unmount (cleanup) • Resetting component state cleanly by changing its `key={selectedUserId}`! • UI, Accessibility & Edge-Case Pitfall Guard (Component Lifecycle Concepts (Mount, Update and Unmount)) • Hands-On Frontend Implementation & Verification (Component Lifecycle Concepts (Mount, Update and Unmount)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Component Lifecycle Concepts (Mount, Update and Unmo...

Purpose: Resetting an `<EditProfileForm key={user.id} user={user} />` cleanly when switching users simply by passing `key={user.id}` (better than `useEffect` state-reset anti-patt...

What Topic Is Next → Next Topic 10.20: Styling React Apps, React DevTools & Feature-Based Project Structure

10.20 Styling React Apps, React DevTools & Feature-Based Project Structure [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: CSS Modules (`Button.module.css`) • Tailwind + `clsx`/`tailwind-merge` conditional classes • Inspecting props/hooks/render-reasons in React Developer Tools • Organizing `src/features/<domain>/{components • UI, Accessibility & Edge-Case Pitfall Guard (Styling React Apps, React DevTools & Feature-Based Project Structure) • Hands-On Frontend Implementation & Verification (Styling React Apps, React DevTools & Feature-Based Project Structure) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Styling React Apps

Purpose: Structuring scalable React codebases by business feature (`features/auth`, `features/employees`, `features/products`) instead of dumping 80 unrelated files into one giant...

What Topic Is Next → Module 10 Capstone (Week 10 Lab Build: Modular Vite + React + TypeScript Interactive Catalog UI) → Next Topic 11.1: Rules of Hooks and Fiber Hook Linked-List Internals

Module 10 Hands-On Capstone Project (100 Marks): Week 10 Lab Build: Modular Vite + React + TypeScript Interactive Catalog UI — Build a multi-component React + TypeScript Product & Expense Board featuring typed props, immutable `useState` updates, category filter chips, empty states, and stable `key` list rendering.

MODULE 11 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 22 TOPICS (176 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 11: React Hooks and Component Development

Master React's complete Hooks & component engineering toolkit — Rules of Hooks, `useEffect` synchronization & cleanup, `useRef`, `useContext`, `useReducer`, `useMemo`, `useCallback`, reusable Custom Hooks (`useDebounce`, `useLocalStorage`, `useFetch`), React Hook Form + Zod schema validation, Error Boundaries, `React.lazy` + `Suspense`, Fragments, Portals, and ref forwarding.

Objectives: ✓ Respect the Rules of Hooks and use `useEffect` strictly for synchronizing with external systems (network, timers, subscriptions, DOM) with accurate dependency arrays and cleanup functions. • ✓ Choose the right Hook for every job: `useRef` (DOM focus & non-rendering mutable values), `useContext` (theme/auth), `useReducer` (multi-event state machines), `useMemo`, and `useCallback`. • ✓ Extract and compose reusable Custom Hooks (`useDebouncedValue`, `useLocalStorageState`, `useMediaQuery`). • ✓ Build production forms with `React Hook Form` + `Zod` (`zodResolver`) and resilient UIs using Error Boundaries, `React.lazy`, `Suspense`, and `createPortal`.

11.1 Rules of Hooks and Fiber Hook Linked-List Internals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Call Hooks ONLY at top level (never inside `if` • `for` • Or early returns!) • ONLY inside React functions/custom `use*` hooks — because React tracks hooks by deterministic call-index order on each Fiber node! • UI, Accessibility & Edge-Case Pitfall Guard (Rules of Hooks and Fiber Hook Linked-List Internals) • Hands-On Frontend Implementation & Verification (Rules of Hooks and Fiber Hook Linked-List Internals) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Rules of Hooks

Purpose: Explaining in React interviews *why* conditional `if (x) useState()` shifts subsequent hook indices and crashes React.

What Topic Is Next → Next Topic 11.2: Deep-Dive useState Patterns

11.2 Deep-Dive useState Patterns [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Grouping cohesive fields vs independent state variables • Avoiding contradictory boolean flags via status union • Resetting state cleanly • UI, Accessibility & Edge-Case Pitfall Guard (Deep-Dive useState Patterns) • Hands-On Frontend Implementation & Verification (Deep-Dive useState Patterns) • Deep-Dive useState Patterns — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Deep-Dive useState Patterns

Purpose: Structuring clean modal, stepper, and filter states.

What Topic Is Next → Next Topic 11.3: useEffect (Synchronizing with External Systems)

11.3 useEffect (Synchronizing with External Systems) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Post-paint execution timing • Valid uses (timers • Window listeners • WebSocket/DOM sync • UI, Accessibility & Edge-Case Pitfall Guard (useEffect (Synchronizing with External Systems)) • Hands-On Frontend Implementation & Verification (useEffect (Synchronizing with External Systems)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime

Execution & Memory Lifecycle for useEffect (Synchronizing with External Systems)

Purpose: Synchronizing `document.title`, `localStorage`, or an `AbortController`-guarded data fetch with current `employeeId`.

What Topic Is Next → Next Topic 11.4: Effect Dependency Arrays and Stale Closure Prevention

11.4 Effect Dependency Arrays and Stale Closure Prevention [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: No array (every render) vs `[]` (mount/unmount) vs `[dep1 • Dep2]` (`Object.is` comparison!) • Why lying to `exhaustive-deps` causes stale closure bugs • Stabilizing object/function deps • UI, Accessibility & Edge-Case Pitfall Guard (Effect Dependency Arrays and Stale Closure Prevention) • Hands-On Frontend Implementation & Verification (Effect Dependency Arrays and Stale Closure Prevention) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Effect Dependency Arrays

Purpose: Moving non-reactive helper objects inside `useEffect` or using functional `setCount(c => c + 1)` so an interval doesn't restart every second!

What Topic Is Next → Next Topic 11.5: Effect Cleanup Functions

11.5 Effect Cleanup Functions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When cleanup runs (BEFORE re-running the effect with new deps • On unmount!) • Clearing `setInterval`/`setTimeout` • `removeEventListener` • UI, Accessibility & Edge-Case Pitfall Guard (Effect Cleanup Functions) • Hands-On Frontend Implementation & Verification (Effect Cleanup Functions) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Effect Cleanup Functions

Purpose: Preventing memory leaks, duplicate event listeners, and out-of-order network responses when switching routes rapidly.

What Topic Is Next → Next Topic 11.6: useRef (DOM References and Non-Rendering Mutable Boxes)

11.6 useRef (DOM References and Non-Rendering Mutable Boxes) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `const inputRef = useRef<HTMLInputElement>(null)` (`inputRef.current?.focus()`) • Storing timer IDs / previous values without triggering re-renders • Rule: never read/write `ref.current` during render! • UI, Accessibility & Edge-Case Pitfall Guard (useRef (DOM References and Non-Rendering Mutable Boxes)) • Hands-On Frontend Implementation & Verification (useRef (DOM References and Non-Rendering Mutable Boxes)) • useRef (DOM References and Non-Rendering Mutable Boxes) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for useRef (DOM References and Non-Rendering Mutable Box...

Purpose: Auto-focusing a Search or OTP input when a modal opens and storing `intervalId` / `latestCallback` refs.

What Topic Is Next → Next Topic 11.7: useContext (Scoped Shared State Without Prop Drilling)

11.7 useContext (Scoped Shared State Without Prop Drilling) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `createContext<AuthContextValue | null>(null)` • `<AuthContext.Provider>` • Wrapping `useContext` in a custom `useAuth()` hook that throws a clear error if used outside `<AuthProvider>`! • UI, Accessibility & Edge-Case Pitfall Guard (useContext (Scoped Shared State Without Prop Drilling)) • Hands-On Frontend Implementation & Verification (useContext (Scoped Shared State Without Prop Drilling)) • useContext (Scoped Shared State Without Prop Drilling) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for useContext (Scoped Shared State Without Prop Drillin...

Purpose: Sharing `currentUser`, `theme`, `locale`, and `showToast` cleanly across the component tree.

What Topic Is Next → Next Topic 11.8: useReducer (Predictable Event-Driven State Transitions)

11.8 useReducer (Predictable Event-Driven State Transitions) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pure `(state • Action) => nextState` reducer + TypeScript Discriminated Union `Action` types • Unit-testing complex cart/wizard reducers as plain TypeScript functions! • UI, Accessibility & Edge-Case Pitfall Guard (useReducer (Predictable Event-Driven State Transitions)) • Hands-On Frontend Implementation & Verification (useReducer (Predictable Event-Driven State Transitions)) • useReducer (Predictable Event-Driven State Transitions) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for useReducer (Predictable Event-Driven State Transitio...

Purpose: Managing multi-step Checkout Wizards or Shopping Cart actions (`ADD_ITEM`, `UPDATE_QTY`, `APPLY_COUPON`, `CLEAR_CART`) cleanly.

What Topic Is Next → Next Topic 11.9: useMemo (Memoizing Expensive Derived Computations)

11.9 useMemo (Memoizing Expensive Derived Computations) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `const stats = useMemo(() => computeHeavyStats(rows) • [rows])` • Preserving referential equality for downstream memoized children/effects • When NOT to overuse `useMemo` on trivial `a + b` math • UI, Accessibility & Edge-Case Pitfall Guard (useMemo (Memoizing Expensive Derived Computations)) • Hands-On Frontend Implementation & Verification (useMemo (Memoizing Expensive Derived Computations)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for useMemo (Memoizing Expensive Derived Computations)

Purpose: Caching an `O(n log n)` filter + sort over 5,000 table rows while an unrelated modal toggle re-renders the parent.

What Topic Is Next → Next Topic 11.10: useCallback (Referential Stability for Functions)

11.10 useCallback (Referential Stability for Functions) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `const handleDelete = useCallback((id: string) => setItems(prev => prev.filter(i => i.id !== id)) • [])` • Why `useCallback` matters ONLY when passed to `React.memo` children or effect dependencies! • UI, Accessibility & Edge-Case Pitfall Guard (useCallback (Referential Stability for Functions)) • Hands-On Frontend Implementation & Verification (useCallback (Referential Stability for Functions)) • useCallback (Referential Stability for Functions) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for useCallback (Referential Stability for Functions)

Purpose: Keeping `onToggleRow` stable so 500 `React.memo(<TableRow />)` rows don't all re-render when 1 row changes.

What Topic Is Next → Next Topic 11.11: Custom Hooks (Extracting Reusable Stateful Logic)

11.11 Custom Hooks (Extracting Reusable Stateful Logic) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `use*` naming contract • How two components calling `useToggle()` get completely independent isolated state (hooks share *logic* • NOT state!) • Clean typed return contracts • UI, Accessibility & Edge-Case Pitfall Guard (Custom Hooks (Extracting Reusable Stateful Logic)) • Hands-On Frontend

Implementation & Verification (Custom Hooks (Extracting Reusable Stateful Logic)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Custom Hooks (Extracting Reusable Stateful Logic)

Purpose: Extracting `useDebouncedValue<T>(value, delay)`, `useLocalStorage<T>(key, init)`, and `useDisclosure()` across dozens of pages.

What Topic Is Next → Next Topic 11.12: Hook Composition (Combining Custom Hooks)

11.12 Hook Composition (Combining Custom Hooks) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Building higher-level domain hooks (`useProductSearch` composing `useDebouncedValue` + `useSearchParams` + `useFetch`) • UI, Accessibility & Edge-Case Pitfall Guard (Hook Composition (Combining Custom Hooks)) • Hands-On Frontend Implementation & Verification (Hook Composition (Combining Custom Hooks)) • Hook Composition (Combining Custom Hooks) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Hook Composition (Combining Custom Hooks) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Keeping page components thin (`const { results, isLoading, query, setQuery } = useEmployeeSearch();`) while business/fetch mechanics live in testable hooks.

What Topic Is Next → Next Topic 11.13: Controlled Forms Architecture in React

11.13 Controlled Forms Architecture in React [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Managing multi-input `values` • `touched` • `errors` • `isSubmitting` states • UI, Accessibility & Edge-Case Pitfall Guard (Controlled Forms Architecture in React) • Hands-On Frontend Implementation & Verification (Controlled Forms Architecture in React) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Controlled Forms Architecture in React

Purpose: Understanding exact form state mechanics from first principles before adopting `React Hook Form`.

What Topic Is Next → Next Topic 11.14: Client-Side Form Validation Logic & UX Patterns

11.14 Client-Side Form Validation Logic & UX Patterns [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Field-level vs cross-field validation (`confirmPassword === password` • `endDate >= startDate`) • Mapping backend `400/422` field errors onto specific inputs • Focusing first invalid field on submit! • UI, Accessibility & Edge-Case Pitfall Guard (Client-Side Form Validation Logic & UX Patterns) • Hands-On Frontend Implementation & Verification (Client-Side Form Validation Logic & UX Patterns) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Client-Side Form Validation Logic

Purpose: Focusing the first invalid input automatically on submit failure so keyboard & mobile users immediately see what needs fixing.

What Topic Is Next → Next Topic 11.15: React Hook Form (Uncontrolled-Performance Typed Forms)

11.15 React Hook Form (Uncontrolled-Performance Typed Forms) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `useForm<FormValues>({ defaultValues • Mode: "onTouched" })` • `{...register("email")}` • `handleSubmit(onValid)` • UI, Accessibility & Edge-Case Pitfall Guard (React Hook Form (Uncontrolled-Performance Typed Forms)) • Hands-On Frontend Implementation & Verification (React Hook Form (Uncontrolled-Performance Typed Forms)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Hook Form (Uncontrolled-Performance Typed Form...

Purpose: Building 20-field Enterprise Employee & Checkout forms with ZERO unnecessary whole-form re-renders per keystroke!

What Topic Is Next → Next Topic 11.16: Zod Schema Validation & Type Inference

11.16 Zod Schema Validation & Type Inference [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `z.object({ email: z.string().trim().email() • Salary: z.coerce.number().min(0) })` • `.refine()` cross-field checks • `type FormValues = z.infer<typeof schema>` (single source of truth!) • UI, Accessibility & Edge-Case Pitfall Guard (Zod Schema Validation & Type Inference) • Hands-On Frontend Implementation & Verification (Zod Schema Validation & Type Inference) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Zod Schema Validation

Purpose: Defining ONE `employeeSchema` in Zod that both validates runtime user/API data AND generates the exact TypeScript type via `z.infer`!

What Topic Is Next → Next Topic 11.17: Error Boundaries (Graceful Render Crash Recovery)

11.17 Error Boundaries (Graceful Render Crash Recovery) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Catching uncaught rendering/lifecycle errors in subtree (`ErrorBoundary` / `react-error-boundary`) • Rendering a friendly fallback card with "Try Again" reset button instead of a blank white screen! What Error Boundaries do NOT catch (event handlers & async callbacks — use `try/catch` there) • UI, Accessibility & Edge-Case Pitfall Guard (Error Boundaries (Graceful Render Crash Recovery)) • Hands-On Frontend Implementation & Verification (Error Boundaries (Graceful Render Crash Recovery)) • Error Boundaries (Graceful Render Crash Recovery) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Error Boundaries (Graceful Render Crash Recovery) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Wrapping route outlets and third-party chart widgets in `<ErrorBoundary>` so one crashed chart never kills navigation or checkout!

What Topic Is Next → Next Topic 11.18: Code Splitting with React.lazy

11.18 Code Splitting with React.lazy [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `const AnalyticsPage = lazy(() => import("./pages/AnalyticsPage"))` • Vite automatic separate JS chunk emission • Keeping initial landing bundle small • Fast • UI, Accessibility & Edge-Case Pitfall Guard (Code Splitting with React.lazy) • Hands-On Frontend Implementation & Verification (Code Splitting with React.lazy) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Code Splitting with React.lazy

Purpose: Splitting heavy Admin Charts or Rich-Text Editor routes out of the initial storefront bundle.

What Topic Is Next → Next Topic 11.19: React Suspense Boundaries

11.19 React Suspense Boundaries [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<Suspense fallback={<PageSkeleton />}>` coordinating lazy route/component loading states declaratively without manual spinner flags • UI, Accessibility & Edge-Case Pitfall Guard (React Suspense Boundaries) • Hands-On Frontend Implementation & Verification (React Suspense Boundaries) • React Suspense Boundaries — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Suspense Boundaries • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Placing smooth layout-matched `<DashboardSkeleton />` Suspense boundaries around lazy route outlets.

What Topic Is Next → Next Topic 11.20: React Fragments (<>...<\/> and Keyed <Fragment>)

11.20 React Fragments (<>...<\/> and Keyed <Fragment>) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Grouping sibling `<td>` cells or `<dt>`/`<dd>` pairs without inserting invalid wrapper `<div>`s into `<tr>` or CSS Grid/Flex parents! Using `<Fragment key={item.id}>` when mapping fragments • UI, Accessibility & Edge-Case Pitfall Guard (React Fragments (<>...<\/> and Keyed <Fragment>)) • Hands-On Frontend Implementation & Verification (React Fragments (<>...<\/> and Keyed <Fragment>)) • React Fragments (<>...<\/> and Keyed <Fragment>) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Fragments (<>...<\/> and Keyed <Fragment>) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Rendering dynamic `<dt>{spec.label}</dt><dd>{spec.value}</dd>` pairs inside a `<dl>` using `<Fragment key={spec.id}>`.

What Topic Is Next → Next Topic 11.21: React Portals (createPortal for Modals, Drawers & Toasts)

11.21 React Portals (createPortal for Modals, Drawers & Toasts) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `createPortal(modalJsx • Document.body)` escaping parent `overflow: hidden` • `z-index` stacking contexts while preserving React event bubbling & Context tree inheritance! • UI, Accessibility & Edge-Case Pitfall Guard (React Portals (createPortal for Modals, Drawers & Toasts)) • Hands-On Frontend Implementation & Verification (React Portals (createPortal for Modals, Drawers & Toasts)) • React Portals (createPortal for Modals, Drawers & Toasts) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Portals (createPortal for Modals, Drawers & To...

Purpose: Rendering accessible `<Modal>`, `<Drawer>`, and `<ToastViewport>` into `document.body` without CSS clipping bugs.

What Topic Is Next → Next Topic 11.22: Ref Forwarding & Context Performance Patterns

11.22 Ref Forwarding & Context Performance Patterns [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Passing `ref` into custom `<Input>` components (React 19 `ref` as prop & `forwardRef` compatibility) • Splitting `StateContext` from `DispatchContext` & memoizing `value` so Context consumers don't re-render needlessly! • UI, Accessibility & Edge-Case Pitfall Guard (Ref Forwarding & Context Performance Patterns) • Hands-On Frontend Implementation & Verification (Ref Forwarding & Context Performance Patterns) • Ref Forwarding & Context Performance Patterns — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Ref Forwarding • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Ensuring `React Hook Form`'s `{...register("email")}` ref attaches cleanly to your custom design-system `<TextField>` component!

What Topic Is Next → Module 11 Capstone (Week 11 Lab Build: Custom Hooks & Zod-Validated Multi-Step Onboarding Form) → Next Topic 12.1: Client-Side SPA Routing with React Router

Module 11 Hands-On Capstone Project (100 Marks): Week 11 Lab Build: Custom Hooks & Zod-Validated Multi-Step Onboarding Form — Create reusable `useDebounce`, `useLocalStorage`, and `useFetch` custom hooks powering a multi-step Employee Onboarding form built with React Hook Form and Zod schema validation.

MODULE 12 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 24 TOPICS (192 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 12: React Router, API Integration and Application Development

Connect TypeScript React applications into full multi-page SPAs backed by live REST APIs — React Router (`BrowserRouter`, nested `<Outlet />` layouts, `useParams`, `useSearchParams`, `<Link>`/`<NavLink>`, Protected & Role-Gated Routes, lazy routes), Fetch & Axios service layers, request/response interceptors, CRUD + Multipart uploads, loading/error/empty/paginated UX, JWT auth flows, Vite env vars, and Mini-Project 3.

Objectives: ✓ Configure client-side SPA routing with nested persistent layouts, dynamic `:id` params, URL `useSearchParams` filter/page syncing, and `404` catch-all routes. • ✓ Implement Protected Authentication Routes (preserving `location.state.from` post-login redirect!) and Role-Based Authorization Guards. • ✓ Architect clean typed Axios / Fetch API service modules with centralized base URLs (`import.meta.env.VITE_API_URL`), Bearer header interceptors, and 401 expiry handling. • ✓ Handle complete `GET`, `POST`, `PUT`/`PATCH`, `DELETE`, and `FormData` file uploads with accessible Loading, Error, Empty, Search, and Paginated UI states. • ✓ Deliver Month 3 Deliverable: Mini-Project 3 — TypeScript + React Employee Management Dashboard.

12.1 Client-Side SPA Routing with React Router [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How SPA routers intercept link clicks via History API without full HTML reloads • Configuring SPA fallback (`rewrites` to `/index.html` on Vercel/Netlify/Nginx!) • UI, Accessibility & Edge-Case Pitfall Guard (Client-Side SPA Routing with React Router) • Hands-On Frontend Implementation & Verification (Client-Side SPA Routing with React Router) • Client-Side SPA Routing with React Router — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Client-Side SPA Routing with React Router • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Instant sub-frame route transitions with persistent shell navigation and audio/state continuity.

What Topic Is Next → Next Topic 12.2: BrowserRouter and Router Setup

12.2 BrowserRouter and Router Setup [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<BrowserRouter>` / `createBrowserRouter` • Mounting router cleanly with global `<AuthProvider>` • `<ErrorBoundary>` • UI, Accessibility & Edge-Case Pitfall Guard (BrowserRouter and Router Setup) • Hands-On Frontend Implementation & Verification (BrowserRouter and Router Setup) • BrowserRouter and

Router Setup — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for BrowserRouter

Purpose: Structuring all application routes in a single declarative `AppRouter.tsx` module.

What Topic Is Next → Next Topic 12.3: Route Definitions and Catch-All 404 Pages

12.3 Route Definitions and Catch-All 404 Pages [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<Routes>` & `<Route path element>` • Index routes (`index`) • Wildcard `path="*"` `<NotFoundPage />` with recovery link • UI, Accessibility & Edge-Case Pitfall Guard (Route Definitions and Catch-All 404 Pages) • Hands-On Frontend Implementation & Verification (Route Definitions and Catch-All 404 Pages) • Route Definitions and Catch-All 404 Pages — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Route Definitions

Purpose: Giving users a helpful 404 recovery screen when visiting mistyped URLs.

What Topic Is Next → Next Topic 12.4: Dynamic Route Parameters (useParams)

12.4 Dynamic Route Parameters (useParams) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `/employees/:employeeId` • Reading `const { employeeId } = useParams()` • Validating undefined/malformed param before API calls • UI, Accessibility & Edge-Case Pitfall Guard (Dynamic Route Parameters (useParams)) • Hands-On Frontend Implementation & Verification (Dynamic Route Parameters (useParams)) • Dynamic Route Parameters (useParams) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Route Parameters (useParams)

Purpose: Loading and editing individual employee profiles at `/employees/emp-104/edit`.

What Topic Is Next → Next Topic 12.5: Nested Routing and Persistent <Outlet /> Layouts

12.5 Nested Routing and Persistent <Outlet /> Layouts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Parent `<DashboardLayout>` (Sidebar + Topbar + `<Outlet />`) wrapping child routes (`/dashboard` • `/employees` • `/settings`) without remounting the shell! • UI, Accessibility & Edge-Case Pitfall Guard (Nested Routing and Persistent <Outlet /> Layouts) • Hands-On Frontend Implementation & Verification (Nested Routing and Persistent <Outlet /> Layouts) • Nested Routing and Persistent <Outlet /> Layouts — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Nested Routing

Purpose: Separating `<AuthLayout>` (centered card for Login/Register) from `<DashboardLayout>` (sidebar + header for authenticated screens).

What Topic Is Next → Next Topic 12.6: Navigation (<Link>, <NavLink> and useNavigate)

12.6 Navigation (<Link>, <NavLink> and useNavigate) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<Link to="...">` (renders real crawlable `<a>` — supports Cmd+Click new tab!) • `<NavLink className={({ isActive }) => ...}>` (`aria-current="page"`!) • Programmatic `navigate("/employees" • { replace: true })` after form submit • UI, Accessibility & Edge-Case Pitfall Guard (Navigation (<Link>, <NavLink> and useNavigate)) • Hands-On Frontend Implementation & Verification (Navigation (<Link>, <NavLink> and useNavigate)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Navigation (<Link>, <NavLink> and useNavigate)

Purpose: Highlighting active sidebar menu items automatically with `<NavLink>` and redirecting with `{ replace: true }` after login so Back doesn't loop to `/login`!

What Topic Is Next → Next Topic 12.7: Protected Routes (Authentication Guards)

12.7 Protected Routes (Authentication Guards) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<ProtectedRoute>` wrapper checking `isBootstrappingAuth` (show loader!) -> `!isAuthenticated` (`<Navigate to="/login" state={{ from: location }} replace />`) -> `<Outlet />` • UI, Accessibility & Edge-Case Pitfall Guard (Protected Routes (Authentication Guards)) • Hands-On Frontend Implementation & Verification (Protected Routes (Authentication Guards)) • Protected Routes (Authentication Guards) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Protected Routes (Authentication Guards) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Sending deep-linked users (`/employees/42` -> `/login` -> right back to `/employees/42` after login!).

What Topic Is Next → Next Topic 12.8: Role-Based Route Guards (Authorization)

12.8 Role-Based Route Guards (Authorization) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<RequireRole allowedRoles={["ADMIN" • "HR"]}>` returning a `<Forbidden403Page />` when an authenticated user lacks permission (plus reminding students backend APIs ALWAYS enforce RBAC too!) • UI, Accessibility & Edge-Case Pitfall Guard (Role-Based Route Guards (Authorization)) • Hands-On Frontend Implementation & Verification (Role-Based Route Guards (Authorization)) • Role-Based Route Guards (Authorization) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Role-Based Route Guards (Authorization) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Gating `/employees/new` and Delete actions to `ADMIN`/`HR` roles while `VIEWER` accounts have read-only dashboard access.

What Topic Is Next → Next Topic 12.9: Route-Based Lazy Code Splitting

12.9 Route-Based Lazy Code Splitting [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Combining `React.lazy(() => import(...))` + `<Suspense>` per route so `/login` downloads in milliseconds without pulling heavy dashboard chart/table code • UI, Accessibility & Edge-Case Pitfall Guard (Route-Based Lazy Code Splitting) • Hands-On Frontend Implementation & Verification (Route-Based Lazy Code Splitting) • Route-Based Lazy Code Splitting — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Route-Based Lazy Code Splitting • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Cutting initial First Load JS by 40–60% on multi-module portals.

What Topic Is Next → Next Topic 12.10: REST API Integration Architecture in React

12.10 REST API Integration Architecture in React [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Separating HTTP transport (`apiClient.ts`) from domain endpoints (`employeesApi.ts`) from UI Hooks/Components • Never hardcoding `fetch("http://localhost:8080/...")` inside JSX components! • UI, Accessibility & Edge-Case Pitfall Guard (REST API Integration Architecture in React) • Hands-On Frontend Implementation & Verification (REST API Integration Architecture in React) • REST API Integration Architecture in React — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for REST API Integration Architecture in React • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Swapping mock/staging/production API URLs or updating an endpoint path in ONE file instead of 15 components.

What Topic Is Next → Next Topic 12.11: Fetch API Integration with Cleanup in React

12.11 Fetch API Integration with Cleanup in React [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Typed `fetch` helper • `AbortController` cleanup on unmount/query change • Handling `204 No Content` empty body (`res.status === 204 ? null : await res.json()`!) • UI, Accessibility & Edge-Case Pitfall Guard (Fetch API Integration with Cleanup in React) • Hands-On Frontend Implementation & Verification (Fetch API Integration with Cleanup in React) • Fetch API Integration with Cleanup in React — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Fetch API Integration with Cleanup in React

Purpose: Handling `DELETE /api/v1/employees/12` (`204 No Content`) cleanly without crashing on `res.json()` over an empty body!

What Topic Is Next → Next Topic 12.12: Axios HTTP Client Configuration

12.12 Axios HTTP Client Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `axios.create({ baseURL • Timeout: 10_000 • Headers })` • Automatic JSON serialization • UI, Accessibility & Edge-Case Pitfall Guard (Axios HTTP Client Configuration) • Hands-On Frontend Implementation & Verification (Axios HTTP Client Configuration) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Axios HTTP Client Configuration

Purpose: Standardizing `timeout: 10000`, `Accept: application/json`, and centralized interceptors across enterprise React projects.

What Topic Is Next → Next Topic 12.13: Typed GET Requests and Query Parameters

12.13 Typed GET Requests and Query Parameters [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `apiClient.get<PaginatedResponse<Employee>>("/employees" • { params • Signal })` • Automatic URL encoding of `params` object • UI, Accessibility & Edge-Case Pitfall Guard (Typed GET Requests and Query Parameters) • Hands-On Frontend Implementation & Verification (Typed GET Requests and Query Parameters) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Typed GET Requests

Purpose: Fetching paginated, filtered employee directories cleanly with typed responses.

What Topic Is Next → Next Topic 12.14: Typed POST Requests (Creating Resources)

12.14 Typed POST Requests (Creating Resources) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `apiClient.post<Employee>("/employees" • Payload)` • Disabling submit during in-flight request • Navigating or updating list state on `201 Created` • UI, Accessibility & Edge-Case Pitfall Guard (Typed POST Requests (Creating Resources)) • Hands-On Frontend Implementation & Verification (Typed POST Requests (Creating Resources)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Typed POST Requests (Creating Resources)

Purpose: Creating new employees with instant toast confirmation and error mapping.

What Topic Is Next → Next Topic 12.15: PUT and PATCH Requests (Updating Resources)

12.15 PUT and PATCH Requests (Updating Resources) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pre-populating edit forms (`reset(loadedEmployee)` in React Hook Form) • Full `PUT` vs partial `PATCH` • Handling `409 Conflict` concurrent updates • UI, Accessibility & Edge-Case Pitfall Guard (PUT and PATCH Requests (Updating Resources)) • Hands-On Frontend Implementation & Verification (PUT and PATCH Requests (Updating Resources)) • PUT and PATCH Requests (Updating Resources) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for PATCH Requests (Updating Resources)

Purpose: Editing employee department, role, and salary cleanly without wiping unchanged fields.

What Topic Is Next → Next Topic 12.16: DELETE Requests and Accessible Confirmation Modals

12.16 DELETE Requests and Accessible Confirmation Modals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Never deleting destructive records on a single accidental click! Confirming via accessible `<ConfirmDeleteModal>` (`role="alertdialog"`) • Disabling row during delete • Refreshing/removing row • UI, Accessibility & Edge-Case Pitfall Guard (DELETE Requests and Accessible Confirmation Modals) • Hands-On Frontend Implementation & Verification (DELETE Requests and Accessible Confirmation Modals) • DELETE Requests and Accessible Confirmation Modals — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DELETE Requests

Purpose: Confirming `"Remove Neha Verma (EMP-102)?"` in a custom accessible modal (never blocking `window.confirm()`!) before firing `DELETE`.

What Topic Is Next → Next Topic 12.17: HTTP Headers (Authorization Bearer, Accept & Idempotency)

12.17 HTTP Headers (Authorization Bearer, Accept & Idempotency) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Authorization: Bearer <accessToken>` • `Accept: application/json` • `X-Request-Id` correlation headers • Why browsers block setting forbidden headers like `Host`/`Origin` • UI, Accessibility & Edge-Case Pitfall Guard (HTTP Headers (Authorization Bearer, Accept & Idempotency)) • Hands-On Frontend Implementation & Verification (HTTP Headers (Authorization Bearer, Accept & Idempotency)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP Headers (Authorization Bearer, Accept & Idempot...

Purpose: Attaching auth tokens and request correlation IDs uniformly across all protected API calls.

What Topic Is Next → Next Topic 12.18: Axios Request and Response Interceptors

12.18 Axios Request and Response Interceptors [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `apiClient.interceptors.request.use(...)` (inject token) • `interceptors.response.use(...)` (normalize errors • Handle `401` token refresh queue or clean logout redirect!) • UI, Accessibility & Edge-Case Pitfall Guard (Axios Request and Response Interceptors) • Hands-On Frontend Implementation & Verification (Axios Request and Response Interceptors) • Axios Request and Response Interceptors — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Axios Request

Purpose: Centralizing `401 Unauthorized` session-expired handling in ONE interceptor instead of repeating `if (err.status === 401)` in 40 components!

What Topic Is Next → Next Topic 12.19: Graceful API Error Handling (axios.isAxiosError & ProblemDetail)

12.19 Graceful API Error Handling (axios.isAxiosError & ProblemDetail) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Distinguishing offline network timeout (`!err.response`) from `400/422` field validation errors (`setError` on inputs!) vs `403` forbidden vs `500` server fault with Retry button • UI, Accessibility & Edge-Case Pitfall Guard (Graceful API Error Handling (axios.isAxiosError & ProblemDetail)) • Hands-On Frontend Implementation & Verification (Graceful API Error Handling (axios.isAxiosError & ProblemDetail)) • Graceful API Error Handling (axios.isAxiosError & ProblemDetail) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Graceful API Error Handling (axios.isAxiosError & Pr... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Turning raw HTTP failures into clear, actionable human messages (`"Email already registered"` on the email input!).

What Topic Is Next → Next Topic 12.20: Loading States (Layout-Preserving Skeletons vs Spinners)

12.20 Loading States (Layout-Preserving Skeletons vs Spinners) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Matching `<TableRowSkeleton />` dimensions to real rows to avoid layout shift • Disabling submit buttons (`aria-busy="true"`) • Avoiding flicker on sub-150ms fast responses • UI, Accessibility & Edge-Case Pitfall Guard (Loading States (Layout-Preserving Skeletons vs Spinners)) • Hands-On Frontend Implementation & Verification (Loading States (Layout-Preserving Skeletons vs Spinners)) • Loading States (Layout-Preserving Skeletons vs Spinners) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Loading States (Layout-Preserving Skeletons vs Spinn...

Purpose: Rendering 8 shimmer skeleton table rows on initial Employee Directory load.

What Topic Is Next → Next Topic 12.21: Empty States (Zero-Data vs Zero-Search-Matches UX)

12.21 Empty States (Zero-Data vs Zero-Search-Matches UX) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Distinguishing first-time empty collection (`"No employees yet -> + Add First Employee"`) from empty filtered search (`"No matches for 'xyz' -> Clear Filters"` button!) • UI, Accessibility & Edge-Case Pitfall Guard (Empty States (Zero-Data vs Zero-Search-Matches UX)) • Hands-On Frontend Implementation & Verification (Empty States (Zero-Data vs Zero-Search-Matches UX)) • Empty States (Zero-Data vs Zero-Search-Matches UX) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Empty States (Zero-Data vs Zero-Search-Matches UX) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Giving users a 1-click `"Reset Search & Filters"` button whenever a filter combination returns 0 rows.

What Topic Is Next → Next Topic 12.22: API-Driven Pagination and useSearchParams Sync

12.22 API-Driven Pagination and useSearchParams Sync [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Syncing `?page=2&size=10&dept=ENG&q=a` via React Router `useSearchParams()` • Resetting `page` back to `1` whenever search or filter changes! • UI, Accessibility & Edge-Case Pitfall Guard (API-Driven Pagination and useSearchParams Sync) • Hands-On Frontend Implementation & Verification (API-Driven Pagination and useSearchParams Sync) • API-Driven Pagination and useSearchParams Sync — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API-Driven Pagination • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Sharing `/employees?dept=Finance&page=3` over Slack so a colleague opens the exact same filtered page.

What Topic Is Next → Next Topic 12.23: Debounced Search, Multi-Faceted Filtering & Multipart File Uploads

12.23 Debounced Search, Multi-Faceted Filtering & Multipart File Uploads [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Combining `useDebouncedValue(searchInput • 300)` + `AbortController` • Uploading employee avatars/resumes via `new FormData()` + `onUploadProgress` percentage bar & file size/MIME client checks • UI, Accessibility & Edge-Case Pitfall Guard (Debounced Search, Multi-Faceted Filtering & Multipart File Uploads) • Hands-On Frontend Implementation & Verification (Debounced Search, Multi-Faceted Filtering & Multipart File Uploads) • Debounced Search, Multi-Faceted Filtering & Multipart File Uploads — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Debounced Search

Purpose: Uploading `<= 2MB` WebP/PNG employee photos with instant client size/type validation and upload progress bar.

What Topic Is Next → Next Topic 12.24: Authentication Flows, JWT Storage Security & Vite Environment Variables

12.24 Authentication Flows, JWT Storage Security & Vite Environment Variables [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Login/Register/Logout flows • Memory access token + `HttpOnly` refresh cookie best practice vs `localStorage` XSS trade-offs • `VITE_*` public env prefix rule (NEVER put private API secrets in `VITE_*` — it ships to every browser!) • UI, Accessibility & Edge-Case Pitfall Guard (Authentication Flows, JWT Storage Security & Vite Environment Variables) • Hands-On Frontend Implementation & Verification (Authentication Flows, JWT Storage Security & Vite Environment Variables) • Authentication Flows, JWT Storage Security & Vite Environment Variables — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory

Lifecycle for Authentication Flows

Purpose: Delivering complete Login, Registration, Protected Routes, and API service layers in Month 3 Mini-Project 3.

What Topic Is Next → Module 12 Capstone (Project 3: React & TypeScript Employee Management Dashboard) → Next Topic 13.1: React Rendering Phases (Render vs Commit)

Module 12 Hands-On Capstone Project (100 Marks): Project 3: React & TypeScript Employee Management Dashboard — Build a strictly typed single-page Employee & Department Management Dashboard using Vite, React 18+, TypeScript 5 (`strict: true`), React Router, React Hook Form + Zod, Axios REST API integr...

MODULE 13 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 21 TOPICS (168 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 13: Advanced React and State Management

Understand React rendering & fiber reconciliation deeply, and master enterprise client vs server state architecture using Context API, Redux Toolkit (`configureStore`, `createSlice`, `createAsyncThunk`, RTK Query), TanStack Query, and Zustand.

Objectives: ✓ Trace React render vs commit phases and fiber reconciliation so state placement minimizes cascading subtree re-renders. • ✓ Separate UI local state, shared Context state, global client state (Redux Toolkit / Zustand), and asynchronous server-cache state (TanStack Query / RTK Query). • ✓ Architect normalized Redux Toolkit stores with typed `RootState`/`AppDispatch`, Immer-powered slices, memoized selectors, and `createAsyncThunk` workflows. • ✓ Manage server data caching, background refetching, cache invalidation, and optimistic UI updates with TanStack Query. • ✓ Persist safe client preferences (theme, cart draft) across browser reloads without persisting sensitive auth secrets.

13.1 React Rendering Phases (Render vs Commit) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Trigger -> Render phase (pure virtual tree calculation) -> Commit phase (DOM mutations & `useLayoutEffect`) -> Browser Paint -> `useEffect` • UI, Accessibility & Edge-Case Pitfall Guard (React Rendering Phases (Render vs Commit)) • Hands-On Frontend Implementation & Verification (React Rendering Phases (Render vs Commit)) • React Rendering Phases (Render vs Commit) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Rendering Phases (Render vs Commit) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Understanding why render functions must be 100% pure (StrictMode double-invokes render in dev to catch impure mutations!).

What Topic Is Next → Next Topic 13.2: Virtual DOM Diffing and Fiber Reconciliation

13.2 Virtual DOM Diffing and Fiber Reconciliation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Element type comparison (`div` vs `section` unmounts state!) • Component position identity • `key` reconciliation in dynamic lists • React 18 automatic state batching • UI, Accessibility & Edge-Case Pitfall Guard (Virtual DOM Diffing and Fiber Reconciliation) • Hands-On Frontend Implementation & Verification (Virtual DOM Diffing and Fiber Reconciliation) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Virtual DOM Diffing

Purpose: Using `key={userId}` to reset an edit form cleanly when switching active user profiles without `useEffect` hacks.

What Topic Is Next → Next Topic 13.3: State Taxonomy: Local, Shared, Global, URL and Server State

13.3 State Taxonomy: Local, Shared, Global, URL and Server State [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Colocating local UI state (`isOpen`) • Lifting shared sibling state • URL search-param state (`?sort=price`) • Global client state • UI, Accessibility & Edge-Case Pitfall Guard (State Taxonomy: Local, Shared, Global, URL and Server State) • Hands-On Frontend Implementation & Verification (State Taxonomy: Local, Shared, Global, URL and Server State) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for State Taxonomy

Purpose: Keeping filter/page state in URL `useSearchParams` (shareable links!) instead of dumping everything into Redux.

What Topic Is Next → Next Topic 13.4: React Context API (`createContext` and `useContext`)

13.4 React Context API (`createContext` and `useContext`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Creating typed contexts with `null` default + custom guard hook `useTheme()` • Wrapping subtree `<Provider>` • Reading context deeply without prop drilling • UI, Accessibility & Edge-Case Pitfall Guard (React Context API (`createContext` and `useContext`)) • Hands-On Frontend Implementation & Verification (React Context API (`createContext` and `useContext`)) • React Context API (`createContext` and `useContext`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Context API (`createContext` and `useContext`)

Purpose: Providing low-frequency global concerns like `ThemeContext` (`light`/`dark`), `LocaleContext`, and `AuthUserContext`.

What Topic Is Next → Next Topic 13.5: Eliminating Prop Drilling with Composition and Context

13.5 Eliminating Prop Drilling with Composition and Context [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Component composition (`children` / slots) vs split State/Dispatch Contexts (`CartStateContext` + `CartDispatchContext`) to isolate re-renders • UI, Accessibility & Edge-Case Pitfall Guard (Eliminating Prop Drilling with Composition and Context) • Hands-On Frontend Implementation & Verification (Eliminating Prop Drilling with Composition and Context) • Eliminating Prop Drilling with Composition and Context — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Eliminating Prop Drilling with Composition • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Passing `<UserMenu />` as `children` into `<TopHeader>` so intermediate layout wrappers never need `user` props at all.

What Topic Is Next → Next Topic 13.6: Redux Architecture Fundamentals

13.6 Redux Architecture Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single source of truth store • Read-only state • Pure reducer functions `(state • Action) => nextState` • UI, Accessibility & Edge-Case Pitfall Guard (Redux Architecture Fundamentals) • Hands-On Frontend Implementation & Verification (Redux Architecture Fundamentals) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory

Lifecycle for Redux Architecture Fundamentals

Purpose: Debugging complex multi-step checkout/cart bugs deterministically via Redux DevTools action log & state diff.

What Topic Is Next → Next Topic 13.7: Redux Toolkit (`@reduxjs/toolkit`) Setup

13.7 Redux Toolkit (`@reduxjs/toolkit`) Setup [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Installing `@reduxjs/toolkit` • `react-redux` • `configureStore` • `<Provider store={store}>` • UI, Accessibility & Edge-Case Pitfall Guard (Redux Toolkit (`@reduxjs/toolkit`) Setup) • Hands-On Frontend Implementation & Verification (Redux Toolkit (`@reduxjs/toolkit`) Setup) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redux Toolkit (`@reduxjs/toolkit`) Setup

Purpose: Bootstrapping a strictly typed Redux Toolkit store in `src/store/index.ts` and `src/store/hooks.ts` with zero `any` casts.

What Topic Is Next → Next Topic 13.8: Redux Store Configuration and Reducer Composition

13.8 Redux Store Configuration and Reducer Composition [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Combining feature slices (`auth` • `cart` • `wishlist` • `ui`) in `configureStore({ reducer: { ... } })` • UI, Accessibility & Edge-Case Pitfall Guard (Redux Store Configuration and Reducer Composition) • Hands-On Frontend Implementation & Verification (Redux Store Configuration and Reducer Composition) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redux Store Configuration

Purpose: Organizing independent domain slices (`cartReducer`, `authReducer`, `uiReducer`) into a cohesive enterprise store.

What Topic Is Next → Next Topic 13.9: Creating Feature Slices with `createSlice`

13.9 Creating Feature Slices with `createSlice` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `name` • `initialState` with TypeScript interface • `reducers` with `PayloadAction<T>` • Immer safe "mutating" syntax (`state.items.push(...)`) compiling to immutable updates • UI, Accessibility & Edge-Case Pitfall Guard (Creating Feature Slices with `createSlice`) • Hands-On Frontend Implementation & Verification (Creating Feature Slices with `createSlice`) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Creating Feature Slices with `createSlice`

Purpose: Implementing `cartSlice` (`addItem`, `incrementQty`, `decrementQty`, `removeItem`, `applyCoupon`, `clearCart`) cleanly.

What Topic Is Next → Next Topic 13.10: Actions, Action Creators and Pure Reducers

13.10 Actions, Action Creators and Pure Reducers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Auto-generated action creators (`cartSlice.actions`) • `prepare` callbacks for ID/timestamp generation outside reducers • `extraReducers` builder pattern • UI, Accessibility & Edge-Case Pitfall Guard (Actions, Action Creators and Pure Reducers) • Hands-On Frontend Implementation & Verification (Actions, Action Creators and Pure Reducers) • Actions, Action Creators and Pure Reducers — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Actions

Purpose: Generating `nanoid()` and ISO timestamps inside `prepare()` so reducer functions remain 100% pure and deterministic!

What Topic Is Next → Next Topic 13.11: Memoized Selectors (`createSelector` / Reselect)

13.11 Memoized Selectors (`createSelector` / Reselect) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Colocated slice selectors • Derived state calculation (subtotal • GST 18% • Discount • UI, Accessibility & Edge-Case Pitfall Guard (Memoized Selectors (`createSelector` / Reselect)) • Hands-On Frontend Implementation & Verification (Memoized Selectors (`createSelector` / Reselect)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Memoized Selectors (`createSelector` / Reselect)

Purpose: Computing `selectCartSummary` (count, subtotal, tax, total) via `createSelector` instead of duplicating `totalPrice` inside state!

What Topic Is Next → Next Topic 13.12: Async Redux Workflows (`createAsyncThunk`)

13.12 Async Redux Workflows (`createAsyncThunk`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lifecycle actions (`pending` • `fulfilled` • `rejected`) • `thunkAPI` (`rejectWithValue` • UI, Accessibility & Edge-Case Pitfall Guard (Async Redux Workflows (`createAsyncThunk`)) • Hands-On Frontend Implementation & Verification (Async Redux Workflows (`createAsyncThunk`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Async Redux Workflows (`createAsyncThunk`)

Purpose: Handling login/session bootstrap (`loginThunk`, `fetchCurrentUserThunk`) with structured backend validation errors via `rejectWithValue`.

What Topic Is Next → Next Topic 13.13: Redux Middleware and Listener Middleware

13.13 Redux Middleware and Listener Middleware [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `createListenerMiddleware` reactive side-effects (syncing cart to `localStorage` • Firing analytics events on `orderPlaced`) • Custom logger middleware mechanics • UI, Accessibility & Edge-Case Pitfall Guard (Redux Middleware and Listener Middleware) • Hands-On Frontend Implementation & Verification (Redux Middleware and Listener Middleware) • Redux Middleware and Listener Middleware — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redux Middleware

Purpose: Syncing `cartSlice` changes to `localStorage` inside `listenerMiddleware` rather than polluting pure slice reducers with `localStorage.setItem`!

What Topic Is Next → Next Topic 13.14: RTK Query Fundamentals (`createApi` and `fetchBaseQuery`)

13.14 RTK Query Fundamentals (`createApi` and `fetchBaseQuery`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Defining declarative API slices • `builder.query` & `builder.mutation` • Auto-generated React hooks (`useGetProductsQuery`) • `providesTags` & `invalidatesTags` automated refetching • UI, Accessibility & Edge-Case Pitfall Guard (RTK Query Fundamentals (`createApi` and `fetchBaseQuery`)) • Hands-On Frontend Implementation & Verification (RTK Query Fundamentals (`createApi` and `fetchBaseQuery`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for RTK Query Fundamentals (`createApi` and `fetchBaseQu...

Purpose: Eliminating hundreds of lines of manual `loading`/`error` thunks in Redux apps with tag-invalidated `productsApi` endpoints.

What Topic Is Next → Next Topic 13.15: Server State vs Client State Architecture

13.15 Server State vs Client State Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why server state is fundamentally different (remote ownership • Asynchronous staleness • Shared multi-user mutations • Background revalidation • UI, Accessibility & Edge-Case Pitfall Guard (Server State vs Client State Architecture) • Hands-On Frontend Implementation & Verification (Server State vs Client State Architecture) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Server State vs Client State Architecture

Purpose: Keeping UI/cart/modal state in client stores while delegating REST API caching/retries/staleness to TanStack Query or RTK Query.

What Topic Is Next → Next Topic 13.16: TanStack Query (`@tanstack/react-query`) Fundamentals

13.16 TanStack Query (`@tanstack/react-query`) Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `QueryClient` & `<QueryClientProvider>` • `useQuery({ queryKey • QueryFn • StaleTime • UI, Accessibility & Edge-Case Pitfall Guard (TanStack Query (`@tanstack/react-query`) Fundamentals) • Hands-On Frontend Implementation & Verification (TanStack Query (`@tanstack/react-query`) Fundamentals) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TanStack Query (`@tanstack/react-query`) Fundamental...

Purpose: Caching paginated product catalog pages with `staleTime: 60_000` so back-button navigation renders instantaneously with zero spinner flicker!

What Topic Is Next → Next Topic 13.17: Mutations and Cache Invalidation (`useMutation`)

13.17 Mutations and Cache Invalidation (`useMutation`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `useMutation({ mutationFn • OnSuccess • OnError })` • `queryClient.invalidateQueries({ queryKey: ['products'] })` • UI, Accessibility & Edge-Case Pitfall Guard (Mutations and Cache Invalidation (`useMutation`)) • Hands-On Frontend Implementation & Verification (Mutations and Cache Invalidation (`useMutation`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Mutations

Purpose: Creating or deleting an employee/product and automatically refetching active list queries via `invalidateQueries`.

What Topic Is Next → Next Topic 13.18: Optimistic UI Updates and Rollback

13.18 Optimistic UI Updates and Rollback [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `onMutate` snapshot (`cancelQueries` -> `getQueryData` -> `setQueryData`) • Instant heart/wishlist toggle feedback • Automatic rollback in `onError` • Final sync in `onSettled` • UI, Accessibility & Edge-Case Pitfall Guard (Optimistic UI Updates and Rollback) • Hands-On Frontend Implementation & Verification (Optimistic UI Updates and Rollback) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Optimistic UI Updates

Purpose: Making Wishlist heart toggles and Todo completion checkboxes feel instantaneous (0ms perceived latency) with safe rollback if the network fails.

What Topic Is Next → Next Topic 13.19: Lightweight Client State with Zustand Fundamentals

13.19 Lightweight Client State with Zustand Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `create<StoreState>()((set • Get) => ({ ... }))` • Fine-grained selector subscriptions `useStore(s => s.sidebarOpen)` • Zero-provider ergonomic global UI stores • UI, Accessibility & Edge-Case Pitfall Guard (Lightweight Client State with Zustand Fundamentals) • Hands-On Frontend Implementation & Verification (Lightweight Client State with Zustand Fundamentals) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Lightweight Client State with Zustand Fundamentals

Purpose: Managing lightweight UI modals, filter drawers, or toast queues in seconds when full Redux ceremony isn't needed.

What Topic Is Next → Next Topic 13.20: Safe Client State Persistence

13.20 Safe Client State Persistence [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Persisting cart/theme/onboarding flags to `localStorage` with schema version migration & Zod parsing on hydration • Cross-tab `storage` event sync • UI, Accessibility & Edge-Case Pitfall Guard (Safe Client State Persistence) • Hands-On Frontend Implementation & Verification (Safe Client State Persistence) • Safe Client State Persistence — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Safe Client State Persistence • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Validating persisted `localStorage` cart JSON with Zod on startup so an upgraded app deployment never crashes on old user browser storage!

What Topic Is Next → Next Topic 13.21: Choosing the Right State Management Architecture

13.21 Choosing the Right State Management Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Practical decision matrix: Component `useState`/`useReducer` vs URL `SearchParams` vs React Context vs Zustand vs Redux Toolkit vs TanStack Query / RTK Query • UI, Accessibility & Edge-Case Pitfall Guard (Choosing the Right State Management Architecture) • Hands-On Frontend Implementation & Verification (Choosing the Right State Management Architecture) • Choosing the Right State Management Architecture — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Choosing the Right State Management Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Defending your exact state-management choices clearly in Senior Frontend architecture interviews ("Why TanStack Query + Redux Toolkit instead of Context alone?").

What Topic Is Next → Module 13 Capstone (Week 13 Lab Build: Redux Toolkit Multi-Slice Cart + TanStack Query Catalog) → Next Topic 14.1: Scalable Component Architecture (Presentational vs Feature Containers)

Module 13 Hands-On Capstone Project (100 Marks): Week 13 Lab Build: Redux Toolkit Multi-Slice Cart + TanStack Query Catalog — Combine TanStack Query server-state caching (paginated products + optimistic wishlist toggle) with a typed Redux Toolkit store (`cartSlice`, `authSlice`, `toastSlice`, memoized `createSelect...

MODULE 14 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 23 TOPICS (184 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 14: Advanced React Application Architecture

Design scalable feature-sliced React codebases, reusable accessible design-system primitives, Compound Components, Error Boundaries, Code Splitting (`lazy`/`Suspense`), memoization (`memo`/`useMemo`/`useCallback`), list virtualization, debounced search, infinite scrolling, and file upload/download workflows.

Objectives: ✓ Structure large React repositories by domain feature (`src/features/catalog`, `cart`, `checkout`, `auth`) rather than giant flat `components/` folders. • ✓ Build composable UI primitives using Compound Components, polymorphic props, portals, and advanced custom hooks. • ✓ Eliminate real rendering bottlenecks measured in React Profiler using state colocation, `React.memo`, `useMemo`, `useCallback`, and list virtualization. • ✓ Split route and heavy chart/editor bundles via `React.lazy` + `<Suspense>` and isolate runtime crashes with `<ErrorBoundary>`. • ✓ Implement debounced/throttled search, `IntersectionObserver` infinite scrolling, and multipart file upload with preview & progress.

14.1 Scalable Component Architecture (Presentational vs Feature Containers) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Separating stateless UI primitives (`Button` • `Badge` • `Modal`) • Domain components (`ProductCard`) • UI, Accessibility & Edge-Case Pitfall Guard (Scalable Component Architecture (Presentational vs Feature Containers)) • Hands-On Frontend Implementation & Verification (Scalable Component Architecture (Presentational vs Feature Containers)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Scalable Component Architecture (Presentational vs F...

Purpose: Keeping `src/components/ui/*` 100% decoupled from business API endpoints so UI components can be reused across apps and tested in isolation.

What Topic Is Next → Next Topic 14.2: Feature-Based Folder Architecture

14.2 Feature-Based Folder Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Organizing large apps into `src/features/<feature>/{api • Components • Hooks • Types • UI, Accessibility & Edge-Case Pitfall Guard (Feature-Based Folder Architecture) • Hands-On Frontend Implementation & Verification (Feature-Based Folder Architecture) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Feature-Based Folder Architecture

Purpose: Enabling 6 frontend engineers to work on `auth`, `catalog`, `cart`, and `orders` simultaneously with minimal Git merge conflicts.

What Topic Is Next → Next Topic 14.3: Design Systems and Tokenized UI Foundations

14.3 Design Systems and Tokenized UI Foundations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Design tokens (spacing • Colors • Radii • Typography • UI, Accessibility & Edge-Case Pitfall Guard (Design Systems and Tokenized UI Foundations) • Hands-On Frontend Implementation & Verification (Design Systems and Tokenized UI Foundations) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Design Systems

Purpose: Building typed `<Button variant="primary" size="md">` components whose Tailwind overrides (`className="mt-4"`) merge deterministically via `cn()`.

What Topic Is Next → Next Topic 14.4: Working with Enterprise Component Libraries (Headless UI / Radix Concepts)

14.4 Working with Enterprise Component Libraries (Headless UI / Radix Concepts) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Styled libraries (MUI • AntD) vs Headless accessible primitives (Radix UI • Headless UI • Shadcn pattern) — owning markup/Tailwind while delegating WAI-ARIA focus math • UI, Accessibility & Edge-Case Pitfall Guard (Working with Enterprise Component Libraries (Headless UI / Radix Concepts)) • Hands-On Frontend Implementation & Verification (Working with Enterprise Component Libraries (Headless UI / Radix Concepts)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Working with Enterprise Component Libraries (Headles...

Purpose: Understanding how headless primitives deliver full WAI-ARIA keyboard compliance (`Dialog`, `Combobox`, `Menu`) with 100% custom Tailwind branding.

What Topic Is Next → Next Topic 14.5: Compound Components Pattern

14.5 Compound Components Pattern [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implicit parent-child state sharing via scoped React Context (`<Tabs>` • `<Tabs.List>` • `<Tabs.Trigger>` • `<Tabs.Panel>`) • UI, Accessibility & Edge-Case Pitfall Guard (Compound Components Pattern) • Hands-On Frontend Implementation & Verification (Compound Components Pattern) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Compound Components Pattern

Purpose: Building flexible `<Accordion>` and `<Tabs>` APIs where consumers freely arrange icons, badges, and markup inside triggers/panels.

What Topic Is Next → Next Topic 14.6: Higher-Order Components (HOCs) — Legacy & Cross-Cutting Awareness

14.6 Higher-Order Components (HOCs) — Legacy & Cross-Cutting Awareness [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Function `(WrappedComponent) => EnhancedComponent` • Prop forwarding • `displayName` • Why Hooks replaced 95% of HOCs • UI, Accessibility & Edge-Case Pitfall Guard (Higher-Order Components (HOCs) — Legacy & Cross-Cutting Awareness) • Hands-On Frontend Implementation & Verification (Higher-Order Components (HOCs) — Legacy & Cross-Cutting Awareness) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Higher-Order Components (HOCs)

Purpose: Reading and refactoring `withAuth(Component)` or `withAnalytics(Component)` wrappers encountered in existing enterprise React codebases.

What Topic Is Next → Next Topic 14.7: Render Props Pattern and Function-as-Child

14.7 Render Props Pattern and Function-as-Child [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inverting rendering control via `renderItem={(item) => ...}` or `children={({ isOpen • Close }) => ...}` for headless data/virtualizer components • UI, Accessibility & Edge-Case Pitfall Guard (Render Props Pattern and Function-as-Child) • Hands-On Frontend Implementation & Verification (Render Props Pattern and Function-as-Child) • Render Props Pattern and Function-as-Child — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Render Props Pattern • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Typing generic `<DataTable<T> data={rows} renderRow={(row) => ...} />` and `<VirtualList items={...} renderItem={...} />` components.

What Topic Is Next → Next Topic 14.8: Advanced Custom Hooks (`useIntersectionObserver`, `useMediaQuery`, `useOnClickOutside`)

14.8 Advanced Custom Hooks (`useIntersectionObserver`, `useMediaQuery`, `useOnClickOutside`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Encapsulating browser observers • SSR-safe `window` guards • Stable callback refs (`useRef` latest-handler pattern) • Cleanup guarantees • UI, Accessibility & Edge-Case Pitfall Guard (Advanced Custom Hooks (`useIntersectionObserver`, `useMediaQuery`, `useOnClickOutside`)) • Hands-On Frontend Implementation & Verification (Advanced Custom Hooks (`useIntersectionObserver`, `useMediaQuery`, `useOnClickOutside`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Advanced Custom Hooks (`useIntersectionObserver`, `u...

Purpose: Closing dropdowns/popovers cleanly on outside pointer down or `Escape` key via a reusable `useOnClickOutside(ref, close)` hook.

What Topic Is Next → Next Topic 14.9: Identifying Unnecessary Re-Renders and State Colocation

14.9 Identifying Unnecessary Re-Renders and State Colocation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why components re-render (parent re-rendered • Own state changed • Consumed context changed) • Pushing state down (`SearchInput` isolation) • UI, Accessibility & Edge-Case Pitfall Guard (Identifying Unnecessary Re-Renders and State Colocation) • Hands-On Frontend Implementation & Verification (Identifying Unnecessary Re-Renders and State Colocation) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Identifying Unnecessary Re-Renders

Purpose: Moving a typing `searchText` state out of a 500-row dashboard layout into a dedicated `<HeaderSearchBox>` so typing never re-renders the data grid!

What Topic Is Next → Next Topic 14.10: Component Memoization with `React.memo`

14.10 Component Memoization with `React.memo` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Shallow prop comparison `Object.is` • When `React.memo` actually helps (expensive pure child rendered frequently with identical props) • Why inline objects/arrays defeat `memo` • UI, Accessibility & Edge-Case Pitfall Guard (Component Memoization with `React.memo`) • Hands-On Frontend Implementation & Verification (Component Memoization with `React.memo`) • Component Memoization with `React.memo` — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Component Memoization with `React.memo`

Purpose: Wrapping `<DataGridRow>` in `React.memo` so selecting 1 checkbox in a 500-row table re-renders only 1 row instead of all 500.

What Topic Is Next → Next Topic 14.11: Expensive Computation Caching with `useMemo`

14.11 Expensive Computation Caching with `useMemo` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Memoizing expensive CPU work (`O(n log n)` sorting/filtering 5 • 000 rows) • Preserving referential identity for downstream `memo`/`useEffect` dependencies • UI, Accessibility & Edge-Case Pitfall Guard (Expensive Computation Caching with `useMemo`) • Hands-On Frontend Implementation & Verification (Expensive Computation Caching with `useMemo`) • Expensive Computation Caching with `useMemo` — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Expensive Computation Caching with `useMemo`

Purpose: Caching client-side multi-facet filtering & sorting across 5,000 catalog items while unrelated UI drawers open and close.

What Topic Is Next → Next Topic 14.12: Stable Function References with `useCallback`

14.12 Stable Function References with `useCallback` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Preserving function identity across renders when passing callbacks to `React.memo` children or custom hook `useEffect` dependency arrays • UI, Accessibility & Edge-Case Pitfall Guard (Stable Function References with `useCallback`) • Hands-On Frontend Implementation & Verification (Stable Function References with `useCallback`) • Stable Function References with `useCallback` — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stable Function References with `useCallback` • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Passing a stable `handleToggleSelect = useCallback((id: string) => ..., [])` using functional `setSelected(prev => ...)` to memoized table rows.

What Topic Is Next → Next Topic 14.13: Code Splitting and Dynamic `import()`

14.13 Code Splitting and Dynamic `import()` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why a single monolithic `index.js` hurts First Contentful Paint & Time to Interactive • Vite automatic Rollup chunk generation on dynamic `import('./HeavyChart')` • UI, Accessibility & Edge-Case Pitfall Guard (Code Splitting and Dynamic `import()`) • Hands-On Frontend Implementation & Verification (Code Splitting and Dynamic `import()`) • Code Splitting and Dynamic `import()` — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Code Splitting • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Splitting the `/admin/*` dashboard, PDF invoice generator, and rich chart pages out of the initial customer storefront bundle.

What Topic Is Next → Next Topic 14.14: Lazy Component Loading (`React.lazy`) and `<Suspense>`

14.14 Lazy Component Loading (`React.lazy`) and `<Suspense>` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Wrapping route groups • Heavy modals in `<Suspense fallback={<PageSkeleton />}>` • Avoiding layout jarring with dimension-matched skeletons • Preloading on link hover • UI, Accessibility & Edge-Case Pitfall Guard (Lazy Component Loading (`React.lazy`) and `<Suspense>`) • Hands-On Frontend Implementation & Verification (Lazy Component Loading (`React.lazy`) and `<Suspense>`) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Lazy Component Loading (`React.lazy`)

Purpose: Lazy-loading `/checkout`, `/orders`, and `/admin` routes with smooth shimmer skeletons instead of tiny jumping spinners.

What Topic Is Next → Next Topic 14.15: Error Boundaries and Graceful Crash Recovery

14.15 Error Boundaries and Graceful Crash Recovery [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Catching render/lifecycle errors in subtree (`getDerivedStateFromError` • `componentDidCatch` • `react-error-boundary`) • Retry reset keys • UI, Accessibility & Edge-Case Pitfall Guard (Error Boundaries and Graceful Crash Recovery) • Hands-On Frontend Implementation & Verification (Error Boundaries and Graceful Crash Recovery) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Error Boundaries

Purpose: Wrapping independent dashboard widgets (`<SalesChart>`, `<RecentOrders>`) in granular Error Boundaries so one broken API payload never white-screens the whole app!

What Topic Is Next → Next Topic 14.16: React Portals (`createPortal`) for Modals, Drawers and Toasts

14.16 React Portals (`createPortal`) for Modals, Drawers and Toasts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `createPortal(children • Document.body)` • Escaping parent `overflow: hidden` / `transform` stacking contexts while preserving React event bubbling & Context access • UI, Accessibility & Edge-Case Pitfall Guard (React Portals (`createPortal`) for Modals, Drawers and Toasts) • Hands-On Frontend Implementation & Verification (React Portals (`createPortal`) for Modals, Drawers and Toasts) • React Portals (`createPortal`) for Modals, Drawers and Toasts — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Portals (`createPortal`) for Modals

Purpose: Rendering accessible `<Modal>`, `<Drawer>`, and `<ToastViewport>` outside clipped cards into `#portal-root` while keeping Redux/Router context.

What Topic Is Next → Next Topic 14.17: List Virtualization / Windowing (`@tanstack/react-virtual` Concepts)

14.17 List Virtualization / Windowing (`@tanstack/react-virtual` Concepts) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why mounting 10 • 000 DOM nodes freezes layout/scroll • Rendering only the ~20 visible viewport rows + overscan buffer with translated container height • UI, Accessibility & Edge-Case Pitfall Guard (List Virtualization / Windowing (`@tanstack/react-virtual` Concepts)) • Hands-On Frontend Implementation & Verification (List Virtualization / Windowing (`@tanstack/react-virtual` Concepts)) • List Virtualization / Windowing (`@tanstack/react-virtual` Concepts) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for List Virtualization / Windowing (`@tanstack/react-vi...

Purpose: Rendering 10,000+ log entries, country/city dropdowns, or enterprise inventory rows at a locked 60 FPS with < 30 DOM nodes!

What Topic Is Next → Next Topic 14.18: Debouncing High-Frequency User Input

14.18 Debouncing High-Frequency User Input [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing `debounce(fn • Delay)` • `useDebouncedValue(value • 300)` from scratch with `setTimeout`/`clearTimeout` cleanup • UI, Accessibility & Edge-Case Pitfall Guard (Debouncing High-Frequency User Input) • Hands-On Frontend Implementation & Verification (Debouncing High-Frequency User Input) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Debouncing High-Frequency User Input

Purpose: Waiting 300ms after the user pauses typing in product search before firing `GET /api/products?q=...` (cuts API load by ~85%!).

What Topic Is Next → Next Topic 14.19: Throttling Continuous Scroll, Resize and Pointer Events

14.19 Throttling Continuous Scroll, Resize and Pointer Events [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Difference between Debounce (wait for silence) vs Throttle (execute at most once every `N` ms / `requestAnimationFrame`) • Cleanup on unmount • UI, Accessibility & Edge-Case Pitfall Guard (Throttling Continuous Scroll, Resize and Pointer Events) • Hands-On Frontend Implementation & Verification (Throttling Continuous Scroll, Resize and Pointer Events) • Throttling Continuous Scroll, Resize and Pointer Events — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Throttling Continuous Scroll • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Updating sticky scroll progress bars, window resize charts, or drag coordinates at steady 16ms/100ms intervals.

What Topic Is Next → Next Topic 14.20: Infinite Scrolling with `IntersectionObserver`

14.20 Infinite Scrolling with `IntersectionObserver` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sentinel `<div ref={loadMoreRef}>` + native `IntersectionObserver` (`rootMargin: '200px'`) + cursor/page `useInfiniteQuery` • Preventing duplicate concurrent page loads • UI, Accessibility & Edge-Case Pitfall Guard (Infinite Scrolling with `IntersectionObserver`) • Hands-On Frontend Implementation & Verification (Infinite Scrolling with `IntersectionObserver`) • Infinite Scrolling with `IntersectionObserver` — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Infinite Scrolling with `IntersectionObserver` • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Prefetching the next 20 products seamlessly 200px before the shopper reaches the bottom of the mobile feed.

What Topic Is Next → Next Topic 14.21: Client and API Search Optimization (Autocomplete, Highlighting & Caching)

14.21 Client and API Search Optimization (Autocomplete, Highlighting & Caching) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Minimum character threshold (`>= 2`) • Trimmed normalized keys • Cached query hits • `AbortController` race cancellation • UI, Accessibility & Edge-Case Pitfall Guard (Client and API Search Optimization (Autocomplete, Highlighting & Caching)) • Hands-On Frontend Implementation & Verification (Client and API Search Optimization (Autocomplete, Highlighting & Caching)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Client

Purpose: Building an Amazon/Flipkart-style instant product search bar with recent searches, keyboard navigation, and zero out-of-order flicker.

What Topic Is Next → Next Topic 14.22: File Upload, Image Preview and Client Download Handling

14.22 File Upload, Image Preview and Client Download Handling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<input type="file" accept>` • Drag- • -drop dropzone • MIME & max-size (`<= 2MB`) validation • UI, Accessibility & Edge-Case Pitfall Guard (File Upload, Image Preview and Client Download Handling) • Hands-On Frontend Implementation & Verification (File Upload, Image Preview and Client Download Handling) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for File Upload

Purpose: Uploading profile avatars / product images with instant thumbnail preview, progress bar, and exporting filtered orders to CSV.

What Topic Is Next → Next Topic 14.23: Building Accessible Semantic Reusable UI Components

14.23 Building Accessible Semantic Reusable UI Components [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Focus management (`autoFocus` return to trigger on modal close) • `aria-expanded` • `aria-controls` • `aria-live="polite"` status toasts • UI, Accessibility & Edge-Case Pitfall Guard (Building Accessible Semantic Reusable UI Components) • Hands-On Frontend Implementation & Verification (Building Accessible Semantic Reusable UI Components) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Building Accessible Semantic Reusable UI Components

Purpose: Ensuring every custom Modal, Dropdown, Accordion, and Toast works 100% via `Tab`, `Shift+Tab`, `Enter`, `Space`, `Escape`, and Arrow keys.

What Topic Is Next → Module 14 Capstone (Week 14 Lab Build: Accessible Design-System Primitives & Virtualized Search Grid) → Next Topic 15.1: Browser and Frontend Security Threat Model

Module 14 Hands-On Capstone Project (100 Marks): Week 14 Lab Build: Accessible Design-System Primitives & Virtualized Search Grid — Build Compound `<Tabs>` and `<Accordion>`, a Portal focus-trapped `<Modal>`, `<ErrorBoundary>`, `IntersectionObserver` infinite scroll, debounced autocomplete, and a 10,000-row virtualized l...

MODULE 15 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 23 TOPICS (184 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 15: Frontend Security, Performance, Testing and Quality

Harden React applications against browser security vulnerabilities (XSS, CSRF, CORS, CSP, token storage, leaked `VITE_*` secrets), optimize Core Web Vitals (LCP, INP, CLS), bundles, images, fonts, and caching, and write automated Vitest, React Testing Library, and Playwright tests.

Objectives: ✓ Identify and prevent Cross-Site Scripting (XSS), `javascript:` URL injection, CSRF, insecure token storage, and leaked frontend secrets. • ✓ Diagnose and optimize Core Web Vitals (LCP < 2.5s, INP < 200ms, CLS < 0.1) using Chrome DevTools Performance/Network panels and Lighthouse. • ✓ Optimize images (`WebP`/`AVIF`, `srcset`, LCP `fetchpriority="high"`), web fonts (`font-display: swap`), tree-shaking, and HTTP `Cache-Control` immutable hashed assets. • ✓ Write fast, user-centric unit and component tests using Vitest, React Testing Library (`getByRole`, `userEvent`), and Mock Service Worker (`MSW`) / API mocks. • ✓ Understand end-to-end browser verification fundamentals with Playwright.

15.1 Browser and Frontend Security Threat Model [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Same-Origin Policy (scheme + host + port) • Untrusted client boundary • OWASP frontend risks (XSS • CSRF • UI, Accessibility & Edge-Case Pitfall Guard (Browser and Frontend Security Threat Model) • Hands-On Frontend Implementation & Verification (Browser and Frontend Security Threat Model) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Browser

Purpose: Reviewing PRs with a security mindset: anything shipped to the browser can be inspected, modified, or replayed by an attacker.

What Topic Is Next → Next Topic 15.2: Cross-Site Scripting (XSS) Prevention in React

15.2 Cross-Site Scripting (XSS) Prevention in React [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reflected • Stored • DOM-based XSS • How React auto-escapes JSX `{text}` children • UI, Accessibility & Edge-Case Pitfall Guard (Cross-Site Scripting (XSS) Prevention in React) • Hands-On Frontend Implementation & Verification (Cross-Site Scripting (XSS) Prevention in React) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cross-Site Scripting (XSS) Prevention in React

Purpose: Validating user-supplied profile URLs (`https:`/`http:` protocol allowlist!) and sanitizing CMS HTML with `DOMPurify` before rendering.

What Topic Is Next → Next Topic 15.3: Cross-Site Request Forgery (CSRF) Fundamentals

15.3 Cross-Site Request Forgery (CSRF) Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How ambient browser cookies auto-attach on cross-site requests • `SameSite=Lax`/`Strict` cookies • Anti-CSRF tokens (`X-XSRF-TOKEN`) in cookie-authenticated SPAs • UI, Accessibility & Edge-Case Pitfall Guard (Cross-Site Request Forgery (CSRF) Fundamentals) • Hands-On Frontend Implementation & Verification (Cross-Site Request Forgery (CSRF) Fundamentals) • Cross-Site Request Forgery (CSRF) Fundamentals — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cross-Site Request Forgery (CSRF) Fundamentals

Purpose: Configuring `withCredentials: true` + `X-XSRF-TOKEN` when integrating React/Angular SPAs with Laravel Sanctum or Spring Security cookie sessions.

What Topic Is Next → Next Topic 15.4: Cross-Origin Resource Sharing (CORS) Deep Dive

15.4 Cross-Origin Resource Sharing (CORS) Deep Dive [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why browsers enforce CORS • Simple requests vs `OPTIONS` preflight (`Content-Type: application/json` • `Authorization`) • `Access-Control-Allow-Origin` • UI, Accessibility & Edge-Case Pitfall Guard (Cross-Origin Resource Sharing (CORS) Deep Dive) • Hands-On Frontend Implementation & Verification (Cross-Origin Resource Sharing (CORS) Deep Dive) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cross-Origin Resource Sharing (CORS) Deep Dive

Purpose: Diagnosing CORS errors instantly in DevTools Network tab instead of installing unsafe "Disable CORS" browser extensions!

What Topic Is Next → Next Topic 15.5: Content Security Policy (CSP) and Security Headers

15.5 Content Security Policy (CSP) and Security Headers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Content-Security-Policy` (`default-src 'self'` • `script-src` • `connect-src` • `frame-ancestors 'none'`) • UI, Accessibility & Edge-Case Pitfall Guard (Content Security Policy (CSP) and Security Headers) • Hands-On Frontend Implementation & Verification (Content Security Policy (CSP) and Security Headers) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Content Security Policy (CSP)

Purpose: Configuring `vercel.json` / `_headers` security headers so injected inline scripts and iframe clickjacking are blocked by the browser.

What Topic Is Next → Next Topic 15.6: Secure Authentication and Token Storage Strategies

15.6 Secure Authentication and Token Storage Strategies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `HttpOnly • Secure • SameSite` cookies vs in-memory access token + `HttpOnly` refresh cookie vs risks of long-lived JWTs in `localStorage` under XSS • UI, Accessibility & Edge-Case Pitfall Guard (Secure Authentication and Token Storage Strategies) • Hands-On Frontend Implementation & Verification (Secure Authentication and Token Storage Strategies) • Secure Authentication and Token Storage Strategies — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Secure Authentication

Purpose: Holding short-lived access tokens in memory (module closure/Redux) paired with an `HttpOnly` refresh endpoint, and clearing all client caches on logout.

What Topic Is Next → Next Topic 15.7: Preventing Exposed Secrets in Frontend Bundles

15.7 Preventing Exposed Secrets in Frontend Bundles [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Vite bundles `import.meta.env.VITE_*` directly into public static JS • Keeping private API keys (OpenAI • Stripe Secret • DB URLs) strictly behind backend proxy endpoints • UI, Accessibility & Edge-Case Pitfall Guard (Preventing Exposed Secrets in Frontend Bundles) • Hands-On Frontend Implementation & Verification (Preventing Exposed Secrets in Frontend Bundles) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Preventing Exposed Secrets in Frontend Bundles

Purpose: Using publishable keys (`VITE_RAZORPAY_KEY_ID`) on the frontend while keeping `RAZORPAY_KEY_SECRET` strictly on the backend server.

What Topic Is Next → Next Topic 15.8: Client-Side Input Validation vs Server Validation

15.8 Client-Side Input Validation vs Server Validation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why Zod/RHF client validation improves instant UX & accessibility while backend validation remains mandatory for security • Trimming/normalizing payloads • Open-redirect `?next=` validation • UI, Accessibility & Edge-Case Pitfall Guard (Client-Side Input Validation vs Server Validation) • Hands-On Frontend Implementation & Verification (Client-Side Input Validation vs Server Validation) • Client-Side Input Validation vs Server Validation — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Client-Side Input Validation vs Server Validation

Purpose: Validating `?redirect=/orders` starts with a single `/` (not `//evil.com`) after login so attackers cannot craft phishing redirect links!

What Topic Is Next → Next Topic 15.9: Measuring Web Page Performance (FCP, TTFB, TBT & Waterfall)

15.9 Measuring Web Page Performance (FCP, TTFB, TBT & Waterfall) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Navigation Timing • TTFB • First Contentful Paint (FCP) • Total Blocking Time (TBT) • UI, Accessibility & Edge-Case Pitfall Guard (Measuring Web Page Performance (FCP, TTFB, TBT & Waterfall)) • Hands-On Frontend Implementation & Verification (Measuring Web Page Performance (FCP, TTFB, TBT & Waterfall)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Measuring Web Page Performance (FCP, TTFB, TBT & Wat...

Purpose: Simulating "Fast 4G + 4x CPU Slowdown" in Chrome DevTools to experience your app on a mid-range Android phone instead of a developer workstation.

What Topic Is Next → Next Topic 15.10: Core Web Vitals: LCP, INP and CLS

15.10 Core Web Vitals: LCP, INP and CLS [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Largest Contentful Paint (`<= 2.5s`) • Interaction to Next Paint (`<= 200ms`) • Cumulative Layout Shift (`<= 0.1`) • `web-vitals` real-user monitoring (RUM) basics • UI, Accessibility & Edge-Case Pitfall Guard (Core Web Vitals: LCP, INP and CLS) • Hands-On Frontend Implementation & Verification (Core Web Vitals: LCP, INP and CLS) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Core Web Vitals

Purpose: Fixing poor CLS by reserving image/skeleton aspect ratios and fixing poor LCP by adding `fetchpriority="high"` + eager loading on the hero banner.

What Topic Is Next → Next Topic 15.11: Modern Image Optimization (`WebP`/`AVIF`, `srcset` & `<picture>`)

15.11 Modern Image Optimization (`WebP`/`AVIF`, `srcset` & `<picture>`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Choosing SVG (icons/logos) vs WebP/AVIF (photos) • Responsive `srcset` + `sizes` • `<picture>` art direction • `loading="lazy"` below fold vs eager hero • UI, Accessibility & Edge-Case Pitfall Guard (Modern Image Optimization (`WebP`/`AVIF`, `srcset` & `<picture>`)) • Hands-On Frontend Implementation & Verification (Modern Image Optimization (`WebP`/`AVIF`, `srcset` & `<picture>`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Modern Image Optimization (`WebP`/`AVIF`, `srcset` &...

Purpose: Serving a 38 KB `400w` WebP thumbnail on mobile product cards instead of downloading a 2.4 MB desktop PNG!

What Topic Is Next → Next Topic 15.12: Web Font Optimization and Preventing FOIT/CLS

15.12 Web Font Optimization and Preventing FOIT/CLS [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Self-hosted `woff2` subsets • `<link rel="preload" as="font" crossorigin>` • `font-display: swap` / `optional` • Metric-matched fallback font stacks • UI, Accessibility & Edge-Case Pitfall Guard (Web Font Optimization and Preventing FOIT/CLS) • Hands-On Frontend Implementation & Verification (Web Font Optimization and Preventing FOIT/CLS) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Web Font Optimization

Purpose: Loading variable `Inter.woff2` (`28 KB`) with `font-display: swap` instead of 6 render-blocking Google Font weights.

What Topic Is Next → Next Topic 15.13: Bundle Size Optimization and Tree-Shaking

15.13 Bundle Size Optimization and Tree-Shaking [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: ES module static imports enabling Rollup/Vite tree-shaking (`import { format } from 'date-fns'` vs full `moment`) • Inspecting bundles with `rollup-plugin-visualizer` • UI, Accessibility & Edge-Case Pitfall Guard (Bundle Size Optimization and Tree-Shaking) • Hands-On Frontend Implementation & Verification (Bundle Size Optimization and Tree-Shaking) • Bundle Size Optimization and Tree-Shaking — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bundle Size Optimization • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Replacing heavy legacy `moment` / `lodash` default imports with native `Intl.DateTimeFormat` / ES6 methods to save 70+ KB gzipped.

What Topic Is Next → Next Topic 15.14: Code Splitting Initial JavaScript Payload

15.14 Code Splitting Initial JavaScript Payload [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Route-based splitting • On-demand modal/drawer chunk loading • Vite `build.rollupOptions.output.manualChunks` vendor separation (`react-vendor`) • UI, Accessibility & Edge-Case Pitfall Guard (Code Splitting Initial JavaScript Payload) • Hands-On Frontend Implementation & Verification (Code Splitting Initial JavaScript Payload) • Code Splitting Initial JavaScript Payload — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Code Splitting Initial JavaScript Payload

Purpose: Ensuring first-time landing visitors download only the lightweight Home/Catalog JS while Checkout/Admin chunks load on demand.

What Topic Is Next → Next Topic 15.15: Browser Caching Strategies (`Cache-Control` & Content Hashing)

15.15 Browser Caching Strategies (`Cache-Control` & Content Hashing) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why Vite emits content-hashed filenames (`index-B7k9x2.js`) • `Cache-Control: public • Max-age=31536000 • Immutable` for `/assets/*` vs `no-cache` for `index.html` • UI, Accessibility & Edge-Case Pitfall Guard (Browser Caching Strategies (`Cache-Control` & Content Hashing)) • Hands-On Frontend Implementation & Verification (Browser Caching Strategies (`Cache-Control` & Content Hashing)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Browser Caching Strategies (`Cache-Control` & Conten...

Purpose: Ensuring repeat visitors load JS/CSS in 0ms from disk cache while always receiving fresh `index.html` immediately after a new deployment!

What Topic Is Next → Next Topic 15.16: Chrome DevTools Performance, Memory & Coverage Profiling

15.16 Chrome DevTools Performance, Memory & Coverage Profiling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Performance flamecharts (identifying Long Tasks & forced synchronous layouts) • Coverage tab (unused CSS/JS bytes) • Memory heap snapshots (detached DOM leaks) • UI, Accessibility & Edge-Case Pitfall Guard (Chrome DevTools Performance, Memory & Coverage Profiling) • Hands-On Frontend Implementation & Verification (Chrome DevTools Performance, Memory & Coverage Profiling) • Chrome DevTools Performance, Memory & Coverage Profiling — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Chrome DevTools Performance

Purpose: Finding a leaked `setInterval` or unremoved `window` listener holding detached DOM trees in Chrome Memory Heap Snapshot.

What Topic Is Next → Next Topic 15.17: Lighthouse Audits (Performance, Accessibility, Best Practices, SEO)

15.17 Lighthouse Audits (Performance, Accessibility, Best Practices, SEO) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running Lighthouse in Incognito against production preview builds • Interpreting actionable diagnostics vs lab score variance • UI, Accessibility & Edge-Case Pitfall Guard (Lighthouse Audits (Performance, Accessibility, Best Practices, SEO)) • Hands-On Frontend Implementation & Verification (Lighthouse Audits (Performance, Accessibility, Best Practices, SEO)) • Lighthouse Audits (Performance, Accessibility, Best Practices, SEO) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Lighthouse Audits (Performance, Accessibility, Best... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Running Lighthouse CI checks before every milestone submission to catch missing `alt` attributes, low contrast, or oversized bundles.

What Topic Is Next → Next Topic 15.18: Cross-Browser and Multi-Device Compatibility Testing

15.18 Cross-Browser and Multi-Device Compatibility Testing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Testing across Chromium (Chrome/Edge) • WebKit (Safari iOS `100dvh` • Date inputs • Flex/scroll quirks) • UI, Accessibility & Edge-Case Pitfall Guard (Cross-Browser and Multi-Device Compatibility Testing) • Hands-On Frontend Implementation & Verification (Cross-Browser and Multi-Device Compatibility Testing) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cross-Browser

Purpose: Using `min-h-dvh` instead of `100vh` so mobile Safari's dynamic address bar never clips sticky bottom "Place Order" buttons!

What Topic Is Next → Next Topic 15.19: Unit Testing Fundamentals for Frontend Logic

15.19 Unit Testing Fundamentals for Frontend Logic [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Testing Trophy / Pyramid for frontend (Static TypeScript -> Pure Unit Tests -> Component Integration Tests -> Critical E2E) • Arrange-Act-Assert (`AAA`) • Deterministic tests • UI, Accessibility & Edge-Case Pitfall Guard (Unit Testing Fundamentals for Frontend Logic) • Hands-On Frontend Implementation & Verification (Unit Testing Fundamentals for Frontend Logic) • Unit Testing Fundamentals for Frontend Logic — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Unit Testing Fundamentals for Frontend Logic

Purpose: Unit-testing pure cart pricing calculators, coupon rules, date formatters, and Redux slice reducers in < 50ms.

What Topic Is Next → Next Topic 15.20: Vitest / Jest Setup, Matchers and Lifecycle

15.20 Vitest / Jest Setup, Matchers and Lifecycle [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring `vitest` + `jsdom` + `@testing-library/jest-dom` (`toBeInTheDocument` • `toHaveTextContent` • `toBeDisabled`) • `describe`/`it`/`expect` • UI, Accessibility & Edge-Case Pitfall Guard (Vitest / Jest Setup, Matchers and Lifecycle) • Hands-On Frontend Implementation & Verification (Vitest / Jest Setup, Matchers and Lifecycle) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Vitest / Jest Setup

Purpose: Testing debounced search hooks instantly using `vi.useFakeTimers()` + `vi.advanceTimersByTime(300)` without sleeping real wall-clock seconds!

What Topic Is Next → Next Topic 15.21: Component Testing with React Testing Library (RTL)

15.21 Component Testing with React Testing Library (RTL) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `render(<Component />)` • Accessible query priority (`getByRole` • `getByLabelText` over `getByTestId`) • `queryBy*` (asserting absence) • UI, Accessibility & Edge-Case Pitfall Guard (Component Testing with React Testing Library (RTL)) • Hands-On Frontend Implementation & Verification (Component Testing with React Testing Library (RTL)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer –

Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Component Testing with React Testing Library (RTL)

Purpose: Querying `screen.getByRole('button', { name: /add to cart/i })` so your tests simultaneously verify screen-reader accessibility!

What Topic Is Next → Next Topic 15.22: Simulating Realistic User Interactions (`@testing-library/user-event`)

15.22 Simulating Realistic User Interactions (`@testing-library/user-event`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `const user = userEvent.setup()` • `await user.type(...)` • `await user.click(...)` • `await user.keyboard('{Escape}')` • UI, Accessibility & Edge-Case Pitfall Guard (Simulating Realistic User Interactions (`@testing-library/user-event`)) • Hands-On Frontend Implementation & Verification (Simulating Realistic User Interactions (`@testing-library/user-event`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Simulating Realistic User Interactions (`@testing-li...

Purpose: Testing a complete Login/Checkout form: empty submit shows validation errors -> valid typing clears errors -> submit calls handler once.

What Topic Is Next → Next Topic 15.23: Mocking REST APIs (`MSW` / `vi.fn`), Coverage & Playwright E2E Intro

15.23 Mocking REST APIs (`MSW` / `vi.fn`), Coverage & Playwright E2E Intro [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Network-level API mocking with Mock Service Worker (`msw` `http.get`/`HttpResponse.json`) vs `vi.spyOn` • Testing loading -> success & `500` error states • Coverage thresholds • Playwright browser E2E intro • UI, Accessibility & Edge-Case Pitfall Guard (Mocking REST APIs (`MSW` / `vi.fn`), Coverage & Playwright E2E Intro) • Hands-On Frontend Implementation & Verification (Mocking REST APIs (`MSW` / `vi.fn`), Coverage & Playwright E2E Intro) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Mocking REST APIs (`MSW` / `vi.fn`)

Purpose: Testing full `<CatalogPage>` loading skeletons, rendered API products, and `500` retry banners realistically without brittle Axios internal mocks.

What Topic Is Next → Module 15 Capstone (Week 15 Lab Build: Lighthouse 90+ Hardened App & Vitest + RTL Test Suite) → Next Topic 16.1: Next.js Architecture: CSR vs SSR vs SSG vs ISR

Module 15 Hands-On Capstone Project (100 Marks): Week 15 Lab Build: Lighthouse 90+ Hardened App & Vitest + RTL Test Suite — Audit and harden a React storefront against XSS/open-redirects, tune LCP/CLS/bundle splitting to >= 90 Lighthouse scores, and write 20+ Vitest + React Testing Library (`userEvent` + mocked A...

MODULE 16 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 19 TOPICS (152 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 16: Next.js Fundamentals and Production React Applications

Learn Next.js App Router fundamentals (CSR vs SSR vs SSG/ISR, Server vs Client Components, layouts, dynamic routes, server data fetching, loading/error UI, SEO metadata, `next/image`, `next/font`, Route Handlers) and build Major Project 4: Production React E-Commerce Frontend Application.

Objectives: ✓ Contrast Client-Side Rendering (Vite SPA), Server-Side Rendering (SSR), and Static Generation (SSG/ISR) so you pick the right tool for dashboards vs SEO storefronts. • ✓ Navigate the Next.js App Router (`app/layout.tsx`, `page.tsx`, `loading.tsx`, `error.tsx`, `[slug]/page.tsx`) and React Server vs Client Component (`'use client'`) boundaries. • ✓ Configure dynamic SEO metadata (`generateMetadata`), `next/image`, `next/font`, environment variables (`NEXT_PUBLIC_*` vs server-only), and lightweight Route Handlers. • ✓ Architect, test, and deliver **Project 4: React E-Commerce Frontend Application** using React, TypeScript, Redux Toolkit / TanStack Query, React Router, Tailwind CSS, and Vitest/RTL.

16.1 Next.js Architecture: CSR vs SSR vs SSG vs ISR [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why pure CSR SPAs send an empty `<div id="root">` initially vs Next.js pre-rendered HTML for SEO/LCP • When Vite React SPA is ideal (auth dashboards) vs Next.js (public storefronts/marketing) • UI, Accessibility & Edge-Case Pitfall Guard (Next.js Architecture: CSR vs SSR vs SSG vs ISR) • Hands-On Frontend Implementation & Verification (Next.js Architecture: CSR vs SSR vs SSG vs ISR) • Next.js Architecture: CSR vs SSR vs SSG vs ISR — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Next.js Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Explaining clearly in interviews when to choose a Vite + React SPA vs Next.js App Router based on SEO, TTFB, and hosting requirements.

What Topic Is Next → Next Topic 16.2: Next.js Project Setup (`create-next-app` with TypeScript & Tailwind)

16.2 Next.js Project Setup (`create-next-app` with TypeScript & Tailwind) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Scaffolding with TypeScript • ESLint • Tailwind CSS • `src/app` directory • UI, Accessibility & Edge-Case Pitfall Guard (Next.js Project Setup (`create-next-app` with TypeScript & Tailwind)) • Hands-On Frontend Implementation & Verification (Next.js Project Setup (`create-next-app` with TypeScript & Tailwind)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Next.js Project Setup (`create-next-app` with TypeSc...

Purpose: Reading `next build` route output symbols (`○` static vs `ƒ` dynamic server-rendered) to verify no static page accidentally opted into per-request SSR.

What Topic Is Next → Next Topic 16.3: File-System Routing with the Next.js App Router

16.3 File-System Routing with the Next.js App Router [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Folder segments (`app/products/page.tsx`) • Route groups `(marketing)` / `(shop)` without affecting URL paths • Private `_components` folders • UI, Accessibility & Edge-Case Pitfall Guard (File-System Routing with the Next.js App Router) • Hands-On Frontend Implementation & Verification (File-System Routing with the Next.js App Router) • File-System Routing with the Next.js App Router — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for File-System Routing with the Next.js App Router

Purpose: Grouping `(marketing)/about` and `(shop)/products` with distinct layouts without polluting public URLs with `/marketing/about`.

What Topic Is Next → Next Topic 16.4: Pages and Persistent Nested Layouts (`page.tsx` & `layout.tsx`)

16.4 Pages and Persistent Nested Layouts (`page.tsx` & `layout.tsx`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Root `app/layout.tsx` (`<html>` • `<body>`) • Nested section layouts preserving state across sibling route navigations • UI, Accessibility & Edge-Case Pitfall Guard (Pages and Persistent Nested Layouts (`page.tsx` & `layout.tsx`)) • Hands-On Frontend Implementation & Verification (Pages and Persistent Nested Layouts (`page.tsx` & `layout.tsx`)) • Pages and Persistent Nested Layouts (`page.tsx` & `layout.tsx`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pages

Purpose: Keeping a persistent category filter sidebar layout mounted in `app/products/layout.tsx` while navigating between category pages.

What Topic Is Next → Next Topic 16.5: Dynamic Routes and Typed Route Params (`[slug]` & `[...catchAll]`)

16.5 Dynamic Routes and Typed Route Params (`[slug]` & `[...catchAll]`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dynamic folder `[slug]/page.tsx` • Awaiting typed `params: Promise<{ slug: string }>` • `searchParams` • `notFound()` (`not-found.tsx`) • UI, Accessibility & Edge-Case Pitfall Guard (Dynamic Routes and Typed Route Params (`[slug]` & `[...catchAll]`)) • Hands-On Frontend Implementation & Verification (Dynamic Routes and Typed Route Params (`[slug]` & `[...catchAll]`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Routes

Purpose: Rendering `/products/[slug]` product detail pages and calling `notFound()` cleanly when a product slug does not exist.

What Topic Is Next → Next Topic 16.6: Navigation and Prefetching with `next/link` and `useRouter`

16.6 Navigation and Prefetching with `next/link` and `useRouter` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<Link href="...">` automatic viewport prefetching • `usePathname()` active nav styling • `useRouter()` (`next/navigation` • NOT legacy `next/router`!) programmatic navigation • UI, Accessibility & Edge-Case Pitfall Guard (Navigation and Prefetching with `next/link` and `useRouter`) • Hands-On Frontend Implementation & Verification (Navigation and Prefetching with `next/link` and `useRouter`) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Navigation

Purpose: Highlighting active navigation links via `usePathname()` inside a lightweight `<NavLinks>` Client Component.

What Topic Is Next → Next Topic 16.7: React Server Components (RSC) Fundamentals

16.7 React Server Components (RSC) Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why App Router components are Server Components by default: zero client JS bundle cost • Direct `async/await` data access on server • No browser hooks/events • UI, Accessibility & Edge-Case Pitfall Guard (React Server Components (RSC) Fundamentals) • Hands-On Frontend Implementation & Verification (React Server Components (RSC) Fundamentals) • React Server Components (RSC) Fundamentals — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Server Components (RSC) Fundamentals

Purpose: Rendering heavy Markdown/specification tables and category listings on the server so 0 KB of parser JS ships to the browser!

What Topic Is Next → Next Topic 16.8: Client Components (`'use client'`) for Interactivity

16.8 Client Components (`'use client'`) for Interactivity [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Adding `'use client'` directive at file top when using `useState` • `useEffect` • Event handlers (`onClick`) • Redux/Query hooks • UI, Accessibility & Edge-Case Pitfall Guard (Client Components (`'use client'`) for Interactivity) • Hands-On Frontend Implementation & Verification (Client Components (`'use client'`) for Interactivity) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Client Components (`'use client'`) for Interactivity

Purpose: Isolating `<AddToCartButton product={p} />` and `<ImageGallery />` as small leaf Client Components inside a Server-rendered product page.

What Topic Is Next → Next Topic 16.9: Server vs Client Component Composition Boundaries

16.9 Server vs Client Component Composition Boundaries [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Serializability rule for props crossing Server -> Client boundary (plain JSON/primitives • NO functions/classes!) • Passing Server Components as `{children}` into Client wrappers • UI, Accessibility & Edge-Case Pitfall Guard (Server vs Client Component Composition Boundaries) • Hands-On Frontend Implementation & Verification (Server vs Client Component Composition Boundaries) • Server vs Client Component Composition Boundaries — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Server vs Client Component Composition Boundaries

Purpose: Wrapping an entire app in a `<Providers>` (`'use client'`) component via `{children}` without converting Server Component pages into Client Components!

What Topic Is Next → Next Topic 16.10: Server Data Fetching vs Client Interactive Fetching

16.10 Server Data Fetching vs Client Interactive Fetching [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Async Server Component `fetch()` + `Promise.all` parallel waterfalls prevention vs client-side TanStack Query/SWR for live interactive search/polling • UI, Accessibility & Edge-Case Pitfall Guard (Server Data Fetching vs Client Interactive Fetching) • Hands-On Frontend Implementation & Verification (Server Data Fetching vs Client Interactive Fetching) • Server Data Fetching vs Client Interactive Fetching — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Server Data Fetching vs Client Interactive Fetching • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Fetching `[product, reviews]` in parallel via `await Promise.all([getProduct(id), getReviews(id)])` to halve server response time!

What Topic Is Next → Next Topic 16.11: Streaming Loading UI (`loading.tsx` and `<Suspense>`)

16.11 Streaming Loading UI (`loading.tsx` and `<Suspense>`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Instant route transitions via `loading.tsx` • Granular `<Suspense fallback={<ReviewsSkeleton />}>` streaming slow sections without blocking primary content • UI, Accessibility & Edge-Case Pitfall Guard (Streaming Loading UI (`loading.tsx` and `<Suspense>`)) • Hands-On Frontend Implementation & Verification (Streaming Loading UI (`loading.tsx` and `<Suspense>`)) • Streaming Loading UI (`loading.tsx` and `<Suspense>`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready

Career Program Production Runtime Execution & Memory Lifecycle for Streaming Loading UI (`loading.tsx` and `<Suspense>`... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Rendering product title/price immediately while slower customer reviews stream in inside their own `<Suspense>` boundary.

What Topic Is Next → Next Topic 16.12: Route Error Handling (`error.tsx` and `global-error.tsx`)

16.12 Route Error Handling (`error.tsx` and `global-error.tsx`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Creating `'use client'` `error.tsx` boundaries receiving `{ error • Reset }` • Recovering route segments without losing the persistent top navbar • UI, Accessibility & Edge-Case Pitfall Guard (Route Error Handling (`error.tsx` and `global-error.tsx`)) • Hands-On Frontend Implementation & Verification (Route Error Handling (`error.tsx` and `global-error.tsx`)) • Route Error Handling (`error.tsx` and `global-error.tsx`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Route Error Handling (`error.tsx` and `global-error....

Purpose: Providing a clean "Retry loading catalog" `reset()` action inside `app/products/error.tsx` when an upstream API times out.

What Topic Is Next → Next Topic 16.13: Static and Dynamic Metadata for Technical SEO

16.13 Static and Dynamic Metadata for Technical SEO [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Exporting `export const metadata: Metadata` • `export async function generateMetadata({ params })` for dynamic `<title>` • `description` • `openGraph` • UI, Accessibility & Edge-Case Pitfall Guard (Static and Dynamic Metadata for Technical SEO) • Hands-On Frontend Implementation & Verification (Static and Dynamic Metadata for Technical SEO) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Static

Purpose: Generating unique product-specific SEO titles, descriptions, and OpenGraph social share cards on `/products/[slug]`.

What Topic Is Next → Next Topic 16.14: Automatic Image Optimization with `next/image`

16.14 Automatic Image Optimization with `next/image` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<Image src alt width height priority sizes />` • Automatic WebP/AVIF resizing • Preventing CLS • `remotePatterns` allowlist in `next.config.ts` • UI, Accessibility & Edge-Case Pitfall Guard (Automatic Image Optimization with `next/image`) • Hands-On Frontend Implementation & Verification (Automatic Image Optimization with `next/image`) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Automatic Image Optimization with `next/image`

Purpose: Setting `priority` on the above-the-fold hero `<Image>` and responsive `sizes` on product grid cards.

What Topic Is Next → Next Topic 16.15: Zero-Layout-Shift Fonts with `next/font`

16.15 Zero-Layout-Shift Fonts with `next/font` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `next/font/google` • `next/font/local` build-time self-hosting (zero external runtime Google requests!) • CSS variable integration with Tailwind • UI, Accessibility & Edge-Case Pitfall Guard (Zero-Layout-Shift Fonts with `next/font`) • Hands-On Frontend Implementation & Verification (Zero-Layout-Shift Fonts with `next/font`) • Zero-Layout-Shift Fonts with `next/font` — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Zero-Layout-Shift Fonts with `next/font`

Purpose: Loading `Inter` and `JetBrains_Mono` at build time via `next/font/google` with automatic size-adjusted fallback metrics.

What Topic Is Next → Next Topic 16.16: Route Handlers (`app/api/.../route.ts`) Fundamentals

16.16 Route Handlers (`app/api/.../route.ts`) Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Web-standard `Request` • `NextResponse.json(...)` handlers (`GET` • `POST`) for BFF proxy routes • Webhooks • UI, Accessibility & Edge-Case Pitfall Guard (Route Handlers (`app/api/.../route.ts`) Fundamentals) • Hands-On Frontend Implementation & Verification (Route Handlers (`app/api/.../route.ts`) Fundamentals) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Route Handlers (`app/api/.../route.ts`) Fundamentals

Purpose: Proxying a third-party API requiring a private `SERVER_API_KEY` through `app/api/recommendations/route.ts` so the secret never reaches browsers.

What Topic Is Next → Next Topic 16.17: Environment Variables (`NEXT_PUBLIC_*` vs Server-Only)

16.17 Environment Variables (`NEXT_PUBLIC_*` vs Server-Only) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.env.local` (gitignored!) vs `.env.example` • Server-only `process.env.API_SECRET` vs browser-inlined `process.env.NEXT_PUBLIC_API_URL` • UI, Accessibility & Edge-Case Pitfall Guard (Environment Variables (`NEXT_PUBLIC_*` vs Server-Only)) • Hands-On Frontend Implementation & Verification (Environment Variables (`NEXT_PUBLIC_*` vs Server-Only)) • Environment Variables (`NEXT_PUBLIC_*` vs Server-Only) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Environment Variables (`NEXT_PUBLIC_*` vs Server-Onl... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Keeping internal microservice credentials strictly non-prefixed so Next.js strips them completely from client bundles.

What Topic Is Next → Next Topic 16.18: Next.js Caching and Revalidation Fundamentals

16.18 Next.js Caching and Revalidation Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Time-based ISR `fetch(url • { next: { revalidate: 60 } })` • Dynamic uncached `cache: 'no-store'` • On-demand `revalidatePath` / `revalidateTag` concepts • UI, Accessibility & Edge-Case Pitfall Guard (Next.js Caching and Revalidation Fundamentals) • Hands-On Frontend Implementation & Verification (Next.js Caching and Revalidation Fundamentals) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Next.js Caching

Purpose: Revalidating public catalog pages every 60 seconds (`revalidate: 60`) while keeping user order dashboards strictly dynamic (`no-store`).

What Topic Is Next → Next Topic 16.19: Production React Build & Project 4 E-Commerce Delivery

16.19 Production React Build & Project 4 E-Commerce Delivery [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Integrating React + TypeScript + Redux Toolkit + TanStack Query + React Router + RHF/Zod + Tailwind + Vitest/RTL into Major Project 4 (React E-Commerce Frontend) & production deployment check • UI, Accessibility & Edge-Case Pitfall Guard (Production React Build & Project 4 E-Commerce Delivery) • Hands-On Frontend Implementation & Verification (Production React Build & Project 4 E-Commerce Delivery) • Production React Build & Project 4 E-Commerce Delivery — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend

Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Production React Build • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Delivering a complete, tested, responsive E-Commerce storefront (listing, filters, sort, details, cart, wishlist, auth, checkout, orders) ready for recruiter demos.

What Topic Is Next → Module 16 Capstone (Project 4: Production React E-Commerce Frontend Application) → Next Topic 17.1: Introduction to Angular and Enterprise Ecosystem

Module 16 Hands-On Capstone Project (100 Marks): Project 4: Production React E-Commerce Frontend Application — Architect, optimize, and test a production-style React + TypeScript E-Commerce Storefront featuring product catalog discovery, multi-facet filters, sorting, instant search, product variant d...

MODULE 17 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 21 TOPICS (168 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 17: Angular Fundamentals and Architecture

Enter enterprise Angular development: understand Angular's batteries-included architecture vs React, master Angular CLI, strict TypeScript decorators, Standalone Components, template bindings, Signals (`signal`, `computed`, `effect`), modern `@if`/`@for`/`@switch` control flow, `@Input`/`@Output`, Pipes, Lifecycle Hooks, and View Encapsulation.

Objectives: ✓ Contrast Angular's opinionated enterprise framework (built-in DI, Router, HttpClient, Reactive Forms, RxJS, Signals) with React's library ecosystem. • ✓ Scaffold strict Angular projects with Angular CLI (`ng new`) and build modular Standalone Components (`standalone: true`). • ✓ Master Angular template interpolation, property/event/two-way `[(ngModel)]` bindings, `@Input()`/`input()`, `@Output()`/`output()`, and reactive Signals (`signal`, `computed`, `effect`). • ✓ Use modern built-in `@if`, `@for` (with mandatory `track`), and `@switch` blocks alongside classic `*ngIf`/`*ngFor`, attribute directives, built-in & custom `@Pipe`s, and component lifecycle hooks.

17.1 Introduction to Angular and Enterprise Ecosystem [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why banking • Insurance • Healthcare • Large enterprise teams standardize on Angular • UI, Accessibility & Edge-Case Pitfall Guard (Introduction to Angular and Enterprise Ecosystem) • Hands-On Frontend Implementation & Verification (Introduction to Angular and Enterprise Ecosystem) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Introduction to Angular

Purpose: Positioning yourself for both React Product roles AND Enterprise Angular Frontend roles in interviews.

What Topic Is Next → Next Topic 17.2: Angular vs React: Architectural Comparison

17.2 Angular vs React: Architectural Comparison [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Framework (opinionated CLI • Router • Forms • HTTP included) vs UI Library (compose Vite + Router + RTK/Query + RHF) • UI, Accessibility & Edge-Case Pitfall Guard (Angular vs React: Architectural Comparison) • Hands-On Frontend Implementation & Verification (Angular vs React: Architectural Comparison) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Angular vs React

Purpose: Answering the classic interview question "Since you know both React and Angular, what are the real engineering trade-offs between them?" objectively.

What Topic Is Next → Next Topic 17.3: Node, npm and Angular CLI (`@angular/cli`) Setup

17.3 Node, npm and Angular CLI (`@angular/cli`) Setup [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Verifying Node LTS • Installing/running `@angular/cli` (`ng version` • `ng generate component/service/pipe/guard` • `ng serve` • UI, Accessibility & Edge-Case Pitfall Guard (Node, npm and Angular CLI (`@angular/cli`) Setup) • Hands-On Frontend Implementation & Verification (Node, npm and Angular CLI (`@angular/cli`) Setup) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Node

Purpose: Using `ng g c features/employees/employee-card` to scaffold consistent, spec-ready components across large teams.

What Topic Is Next → Next Topic 17.4: Creating an Angular Workspace (`ng new`)

17.4 Creating an Angular Workspace (`ng new`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Scaffolding with Standalone defaults • SCSS/CSS • Routing • Strict TypeScript + `strictTemplates` compiler checks • UI, Accessibility & Edge-Case Pitfall Guard (Creating an Angular Workspace (`ng new`)) • Hands-On Frontend Implementation & Verification (Creating an Angular Workspace (`ng new`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Creating an Angular Workspace (`ng new`)

Purpose: Enabling `strictTemplates: true` in `tsconfig.json` so template typos and wrong `@Input` types fail immediately at compile time!

What Topic Is Next → Next Topic 17.5: Angular Project Structure (`angular.json`, `main.ts`, `app.config.ts`)

17.5 Angular Project Structure (`angular.json`, `main.ts`, `app.config.ts`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `angular.json` build/architect targets • `src/main.ts` (`bootstrapApplication(AppComponent • AppConfig)`) • `app.config.ts` providers (`provideRouter` • UI, Accessibility & Edge-Case Pitfall Guard (Angular Project Structure (`angular.json`, `main.ts`, `app.config.ts`)) • Hands-On Frontend Implementation & Verification (Angular Project Structure (`angular.json`, `main.ts`, `app.config.ts`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Angular Project Structure (`angular.json`, `main.ts`...

Purpose: Registering application-wide router, HTTP, and animation providers cleanly inside `app.config.ts`.

What Topic Is Next → Next Topic 17.6: TypeScript for Angular (Decorators, Classes & Access Modifiers)

17.6 TypeScript for Angular (Decorators, Classes & Access Modifiers) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@Component` • `@Injectable` • `@Directive` • `@Pipe` metadata decorators • UI, Accessibility & Edge-Case Pitfall Guard (TypeScript for Angular (Decorators, Classes & Access Modifiers)) • Hands-On Frontend Implementation & Verification (TypeScript for Angular (Decorators, Classes & Access Modifiers))

• Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TypeScript for Angular (Decorators, Classes & Access...

Purpose: Marking template-bound signals/methods `protected readonly` so they are accessible to the component's own HTML template without leaking as public API.

What Topic Is Next → Next Topic 17.7: Angular Component Architecture (`@Component`)

17.7 Angular Component Architecture (`@Component`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `selector` (`app-...` kebab-case prefix) • `imports` • `templateUrl` vs inline `template` • `styleUrl` • UI, Accessibility & Edge-Case Pitfall Guard (Angular Component Architecture (`@Component`)) • Hands-On Frontend Implementation & Verification (Angular Component Architecture (`@Component`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Angular Component Architecture (`@Component`)

Purpose: Keeping complex enterprise forms/tables cleanly separated between declarative `.component.html` markup and typed `.component.ts` view logic.

What Topic Is Next → Next Topic 17.8: Standalone Components vs Legacy `NgModule` Awareness

17.8 Standalone Components vs Legacy `NgModule` Awareness [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Standalone Components import exactly what they use directly (`imports: [CommonModule • RouterLink]`) vs how legacy `NgModule` (`declarations` • `imports` • `exports`) workspaces work • UI, Accessibility & Edge-Case Pitfall Guard (Standalone Components vs Legacy `NgModule` Awareness) • Hands-On Frontend Implementation & Verification (Standalone Components vs Legacy `NgModule` Awareness) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Standalone Components vs Legacy `NgModule` Awareness

Purpose: Writing modern Standalone Angular code for greenfield apps while understanding `SharedModule`/`FeatureModule` when maintaining existing bank apps.

What Topic Is Next → Next Topic 17.9: Angular Templates and Expression Syntax

17.9 Angular Templates and Expression Syntax [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Template expressions vs statements • Safe navigation `user?.department?.name` • Nullish coalescing `??` • Template reference variables `#searchInput` • UI, Accessibility & Edge-Case Pitfall Guard (Angular Templates and Expression Syntax) • Hands-On Frontend Implementation & Verification (Angular Templates and Expression Syntax) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Angular Templates

Purpose: Referencing `#searchBox` directly in templates (`(keyup.enter)="onSearch(searchBox.value)"`) cleanly.

What Topic Is Next → Next Topic 17.10: Interpolation (`{{ }}`) and Property Binding (`[prop]`)

17.10 Interpolation (`{{ }}`) and Property Binding (`[prop]`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Auto-escaped text interpolation `{{ user.name }}` • DOM property binding `[disabled]="isSubmitting()"` • Attribute binding `[attr.aria-label]` • Class `[class.active]` & style `[style.width.%]` bindings • UI, Accessibility & Edge-Case Pitfall Guard (Interpolation (`{{ }}`) and Property Binding (`[prop]`)) • Hands-On Frontend Implementation & Verification (Interpolation (`{{ }}`) and Property Binding (`[prop]`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interpolation (`{{ }}`)

Purpose: Using `[disabled]="form.invalid"` (boolean DOM property!) rather than string interpolation `disabled="{{ form.invalid }}"` (which always disables!).

What Topic Is Next → Next Topic 17.11: Event Binding (`(event)`) and Key Filters

17.11 Event Binding (`(event)`) and Key Filters [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `(click)="onSave()"` • Typed `$event` (`MouseEvent` • `Event`) • Built-in Angular key pseudo-events `(keydown.enter)` • UI, Accessibility & Edge-Case Pitfall Guard (Event Binding (`(event)`) and Key Filters) • Hands-On Frontend Implementation & Verification (Event Binding (`(event)`) and Key Filters) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Event Binding (`(event)`)

Purpose: Closing modals on `(keydown.escape)="close()"` and submitting search on `(keydown.enter)="search()"` declaratively.

What Topic Is Next → Next Topic 17.12: Two-Way Data Binding (`[(ngModel)]` and Custom `[(value)]`)

17.12 Two-Way Data Binding (`[(ngModel)]` and Custom `[(value)]`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `FormsModule` `[(ngModel)]="filterText"` • Banana-in-a-box desugaring `[value]="x" (valueChange)="x = $event"` • Modern `model()` signal two-way binding • UI, Accessibility & Edge-Case Pitfall Guard (Two-Way Data Binding (`[(ngModel)]` and Custom `[(value)]`)) • Hands-On Frontend Implementation & Verification (Two-Way Data Binding (`[(ngModel)]` and Custom `[(value)]`)) • Two-Way Data Binding (`[(ngModel)]` and Custom `[(value)]`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Two-Way Data Binding (`[(ngModel)]` and Custom `[(va...

Purpose: Binding quick toolbar filter selects or custom `<app-rating [(rating)]="score">` controls cleanly.

What Topic Is Next → Next Topic 17.13: Parent–Child Component Communication (`@Input`/`input` & `@Output`/`output`)

17.13 Parent–Child Component Communication (`@Input`/`input` & `@Output`/`output`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Passing data down via `@Input({ required: true })` / signal `input.required<Employee>()` • Emitting typed events up via `@Output() deleted = new EventEmitter<string>()` / `output<string>()` • UI, Accessibility & Edge-Case Pitfall Guard (Parent–Child Component Communication (`@Input`/`input` & `@Output`/`output`)) • Hands-On Frontend Implementation & Verification (Parent–Child Component Communication (`@Input`/`input` & `@Output`/`output`)) • Parent–Child Component Communication (`@Input`/`input` & `@Output`/`output`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Parent–Child Component Communication (`@Input`/`inpu... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Passing `employee` into `<app-employee-row [employee]="emp" (remove)="deleteEmployee($event)">` with compile-time required input safety.

What Topic Is Next → Next Topic 17.14: Angular Signals (`signal`, `computed` and `effect`)

17.14 Angular Signals (`signal`, `computed` and `effect`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fine-grained reactivity in modern Angular: writable `signal(0)` (`.set()` • `.update()`) • Derived memoized `computed(() => ...)` • Side-effect `effect()` • UI, Accessibility & Edge-Case Pitfall Guard (Angular Signals (`signal`, `computed` and `effect`)) • Hands-On Frontend Implementation & Verification (Angular Signals (`signal`, `computed` and `effect`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Angular Signals (`signal`, `computed` and `effect`)

Purpose: Deriving filtered/sorted KPI totals via `computed()` so Angular knows *precisely* which template expressions changed without scanning unchanged components!

What Topic Is Next → Next Topic 17.15: Directives Overview (Component, Structural and Attribute Directives)

17.15 Directives Overview (Component, Structural and Attribute Directives) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Components are directives with templates • Structural directives reshape DOM (`*ngIf` • `*ngFor`) • Attribute directives customize element behavior/appearance (`ngClass` • UI, Accessibility & Edge-Case Pitfall Guard (Directives Overview (Component, Structural and Attribute Directives)) • Hands-On Frontend Implementation & Verification (Directives Overview (Component, Structural and Attribute Directives)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Directives Overview (Component, Structural and Attri...

Purpose: Creating a custom `[appHasRole]="'ADMIN'"` or `[appAutoFocus]` directive to encapsulate reusable DOM behavior cleanly.

What Topic Is Next → Next Topic 17.16: Modern Built-In Control Flow (`@if`, `@for`, `@empty`, `@switch`) & `*ngIf`/`*ngFor`

17.16 Modern Built-In Control Flow (`@if`, `@for`, `@empty`, `@switch`) & `*ngIf`/`*ngFor` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Zero-import `@if (user) { ... } @else { ... }` • `@for (item of items() • Track item.id) { ... } @empty { ... }` • `@switch (status)` • UI, Accessibility & Edge-Case Pitfall Guard (Modern Built-In Control Flow (`@if`, `@for`, `@empty`, `@switch`) & `*ngIf`/`*ngFor`) • Hands-On Frontend Implementation & Verification (Modern Built-In Control Flow (`@if`, `@for`, `@empty`, `@switch`) & `*ngIf`/`*ngFor`) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Modern Built-In Control Flow (`@if`, `@for`, `@empty...

Purpose: Using `@for (emp of employees(); track emp.id)` with `@empty` to get optimal DOM reuse and zero-boilerplate empty states out of the box!

What Topic Is Next → Next Topic 17.17: Built-In Pipes (`currency`, `date`, `number`, `uppercase`, `json`, `async`)

17.17 Built-In Pipes (`currency`, `date`, `number`, `uppercase`, `json`, `async`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Declarative pure template formatting `{{ amount | currency:'INR':'symbol':'1.0-0' }}` • `{{ joinedAt | date:'dd MMM yyyy' }}` • Chaining pipes & pipe parameters • UI, Accessibility & Edge-Case Pitfall Guard (Built-In Pipes (`currency`, `date`, `number`, `uppercase`, `json`, `async`)) • Hands-On Frontend Implementation & Verification (Built-In Pipes (`currency`, `date`, `number`, `uppercase`, `json`, `async`)) • Built-In Pipes (`currency`, `date`, `number`, `uppercase`, `json`, `async`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Built-In Pipes (`currency`, `date`, `number`, `upper...

Purpose: Formatting Indian Rupee amounts and timestamps cleanly in templates via pure memoized pipes instead of component getter methods.

What Topic Is Next → Next Topic 17.18: Creating Custom Pure Pipes (`@Pipe`)

17.18 Creating Custom Pure Pipes (`@Pipe`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `PipeTransform` interface (`transform(value • ...args)`) • Pure (`pure: true` default — runs only when input reference changes!) vs impure pipes (`pure: false` performance danger!) • UI, Accessibility & Edge-Case Pitfall Guard (Creating Custom Pure Pipes (`@Pipe`)) • Hands-On Frontend Implementation & Verification (Creating Custom Pure Pipes (`@Pipe`)) • Creating Custom Pure Pipes (`@Pipe`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Creating Custom Pure Pipes (`@Pipe`)

Purpose: Writing pure `TruncatePipe`, `InitialsPipe` (`"Ravi Mishra" -> "RM"`), and `FileSizePipe` (`1048576 -> "1.0 MB"`).

What Topic Is Next → Next Topic 17.19: Component Lifecycle Hooks (`ngOnInit`, `ngOnChanges`, `ngAfterViewInit`, `ngOnDestroy`)

17.19 Component Lifecycle Hooks (`ngOnInit`, `ngOnChanges`, `ngAfterViewInit`, `ngOnDestroy`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Execution sequence from `constructor` -> `ngOnChanges` -> `ngOnInit` -> `ngAfterViewInit` -> `ngOnDestroy` (• Modern `DestroyRef.onDestroy` / `viewChild()` signal queries) • UI, Accessibility & Edge-Case Pitfall Guard (Component Lifecycle Hooks (`ngOnInit`, `ngOnChanges`, `ngAfterViewInit`, `ngOnDestroy`)) • Hands-On Frontend Implementation & Verification (Component Lifecycle Hooks (`ngOnInit`, `ngOnChanges`, `ngAfterViewInit`, `ngOnDestroy`)) • Component Lifecycle Hooks (`ngOnInit`, `ngOnChanges`, `ngAfterViewInit`, `ngOnDestroy`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Component Lifecycle Hooks (`ngOnInit`, `ngOnChanges`... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Initializing data subscriptions in `ngOnInit`, measuring DOM elements in `ngAfterViewInit`, and cleaning up timers/streams via `DestroyRef` / `ngOnDestroy`.

What Topic Is Next → Next Topic 17.20: View Encapsulation (`Emulated`, `None` and `ShadowDom`)

17.20 View Encapsulation (`Emulated`, `None` and `ShadowDom`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How default `ViewEncapsulation.Emulated` scopes component CSS with `[_ngcontent-ng-c...]` attributes • `:host` selector styling • Why `::ng-deep` is deprecated • UI, Accessibility & Edge-Case Pitfall Guard (View Encapsulation (`Emulated`, `None` and `ShadowDom`)) • Hands-On Frontend Implementation & Verification (View Encapsulation (`Emulated`, `None` and `ShadowDom`)) • View Encapsulation (`Emulated`, `None` and `ShadowDom`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for View Encapsulation (`Emulated`, `None` and `ShadowDo...

Purpose: Styling the component host element cleanly (`:host { display: block; }`) without leaking CSS rules into sibling features.

What Topic Is Next → Next Topic 17.21: Debugging with Angular DevTools

17.21 Debugging with Angular DevTools [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inspecting component hierarchy • Live `@Input`/Signal values • Dependency injection resolution tree • Change Detection flamegraph profiler in Chrome Angular DevTools • UI, Accessibility & Edge-Case Pitfall Guard (Debugging with Angular DevTools) • Hands-On Frontend Implementation & Verification (Debugging with Angular DevTools) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Debugging with Angular DevTools

Purpose: Profiling change-detection cycles and verifying Signal values in real time using the official Angular DevTools Chrome extension.

What Topic Is Next → Module 17 Capstone (Week 17 Lab Build: Standalone Angular Inventory Manager with Signals & Pipes) → Next Topic 18.1: Service Layer Architecture (`@Injectable`)

Module 17 Hands-On Capstone Project (100 Marks): Week 17 Lab Build: Standalone Angular Inventory Manager with Signals & Pipes — Build a Standalone Angular 17+ inventory tracker using `signal()`, `computed()`, `input()`/`output()`, `@if`/`@for` (`track item.id`), built-in `currency:'INR'` formatting, and custom `Initi...

MODULE 18 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 24 TOPICS (192 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 18: Angular Services, Dependency Injection and Routing

Architect enterprise Angular service layers, Hierarchical Dependency Injection (`providedIn: 'root'`, `InjectionToken`, `inject()`), multi-level Angular Routing (params, query params, child routes, lazy `loadComponent`/`loadChildren`, functional `CanActivateFn` guards, resolvers), and typed `HttpClient` REST API integration with functional HTTP Interceptors.

Objectives: ✓ Extract business logic and API orchestration out of components into testable `@Injectable` services injected via `inject()` or constructor DI. • ✓ Configure multi-view Angular Router navigation with nested `<router-outlet>`, route/query params, lazy-loaded feature routes, and functional route guards (`CanActivateFn`, `CanDeactivateFn`). • ✓ Perform typed `GET`, `POST`, `PUT`/`PATCH`, and `DELETE` REST operations via `provideHttpClient(withInterceptors([...]))`. • ✓ Implement functional HTTP Interceptors attaching `Authorization: Bearer` headers and centralized RxJS `catchError` handling for `401`/`403`/`500` responses.

18.1 Service Layer Architecture (`@Injectable`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Thin presentation components + cohesive `@Injectable` domain services (`EmployeeApiService` • `AuthService` • `NotificationService`) • UI, Accessibility & Edge-Case Pitfall Guard (Service Layer Architecture (`@Injectable`)) • Hands-On Frontend Implementation & Verification (Service Layer Architecture (`@Injectable`)) • Service Layer Architecture (`@Injectable`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service Layer Architecture (`@Injectable`)

Purpose: Keeping components focused strictly on UI state & bindings while `EmployeeApiService` owns endpoint URLs, DTO mapping, and caching.

What Topic Is Next → Next Topic 18.2: Dependency Injection (DI) Design Pattern in Angular

18.2 Dependency Injection (DI) Design Pattern in Angular [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inversion of Control: requesting abstractions/tokens from Angular's Injector instead of `new EmployeeService()` • Swapping mocks effortlessly in unit tests • UI, Accessibility & Edge-Case Pitfall Guard (Dependency Injection (DI) Design Pattern in Angular) • Hands-On Frontend Implementation & Verification (Dependency Injection (DI) Design Pattern in Angular) • Dependency Injection (DI) Design Pattern in Angular — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dependency Injection (DI) Design Pattern in Angular • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Providing a fake in-memory `AuthService` stub in `TestBed.configureTestingModule` with zero changes to component code!

What Topic Is Next → Next Topic 18.3: Providers (`useClass`, `useValue`, `useFactory`, `InjectionToken`)

18.3 Providers (`useClass`, `useValue`, `useFactory`, `InjectionToken`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Typing non-class configuration values (`API_BASE_URL`) safely via `new InjectionToken<string>('API_BASE_URL')` • Factory providers • UI, Accessibility & Edge-Case Pitfall Guard (Providers (`useClass`, `useValue`, `useFactory`, `InjectionToken`)) • Hands-On Frontend Implementation & Verification (Providers (`useClass`, `useValue`, `useFactory`, `InjectionToken`)) • Providers (`useClass`, `useValue`, `useFactory`, `InjectionToken`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Providers (`useClass`, `useValue`, `useFactory`, `In... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Injecting `API_BASE_URL` via a typed `InjectionToken<string>` configured once in `app.config.ts`.

What Topic Is Next → Next Topic 18.4: Modern `inject()` Function vs Constructor Injection

18.4 Modern `inject()` Function vs Constructor Injection [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Field-initializer `private readonly auth = inject(AuthService) • ` (injection context rules!) • Using `inject()` inside functional guards • Interceptors • UI, Accessibility & Edge-Case Pitfall Guard (Modern `inject()` Function vs Constructor Injection) • Hands-On Frontend Implementation & Verification (Modern `inject()` Function vs Constructor Injection) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Modern `inject()` Function vs Constructor Injection

Purpose: Writing concise 6-line `authGuard: CanActivateFn` functions that call `inject(AuthService)` and `inject(Router)` directly!

What Topic Is Next → Next Topic 18.5: Hierarchical Injectors (`root` vs Route vs Component Providers)

18.5 Hierarchical Injectors (`root` vs Route vs Component Providers) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: App-wide tree-shakeable singletons (`providedIn: 'root'`) vs route-scoped providers vs per-component instances (`@Component({ providers: [DraftWizardService] })`) • UI, Accessibility & Edge-Case Pitfall Guard (Hierarchical Injectors (`root` vs Route vs Component Providers)) • Hands-On Frontend Implementation & Verification (Hierarchical Injectors (`root` vs Route vs Component Providers)) • Hierarchical Injectors (`root` vs Route vs Component Providers) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Hierarchical Injectors (`root` vs Route vs Component... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Providing `WizardStepService` at the parent `<app-checkout-wizard>` component level so its state auto-disposes when leaving checkout!

What Topic Is Next → Next Topic 18.6: Angular Router Architecture (`provideRouter`)

18.6 Angular Router Architecture (`provideRouter`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring `Routes` in `app.routes.ts` • Registering `provideRouter(routes • WithComponentInputBinding())` in `app.config.ts` • HTML5 pushState navigation • UI, Accessibility & Edge-Case Pitfall Guard (Angular Router Architecture (`provideRouter`)) • Hands-On Frontend Implementation & Verification (Angular Router Architecture (`provideRouter`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Angular Router Architecture (`provideRouter`)

Purpose: Enabling `withComponentInputBinding()` so route params `:id` and query params bind directly to component `input()` signals!

What Topic Is Next → Next Topic 18.7: Route Configuration, Redirects and Wildcard 404 Routes

18.7 Route Configuration, Redirects and Wildcard 404 Routes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `path` (no leading slash!) • `redirectTo` with `pathMatch: 'full'` • `title` page-title resolver • `**` 404 fallback • UI, Accessibility & Edge-Case Pitfall Guard (Route Configuration, Redirects and Wildcard 404 Routes) • Hands-On Frontend Implementation & Verification (Route Configuration, Redirects and Wildcard 404 Routes) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Route Configuration

Purpose: Setting route `title: 'Employees | Enterprise Admin'` so browser tab titles and screen-reader route announcements update automatically.

What Topic Is Next → Next Topic 18.8: Rendering Views with `<router-outlet>`

18.8 Rendering Views with `<router-outlet>` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Primary `<router-outlet />` placement inside shell layouts • Sibling header/sidebar persistence during route swaps • UI, Accessibility & Edge-Case Pitfall Guard (Rendering Views with `<router-outlet>`) • Hands-On Frontend Implementation & Verification (Rendering Views with `<router-outlet>`) • Rendering Views with `<router-outlet>` — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Rendering Views with `<router-outlet>` • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Rendering `<app-sidebar>` + `<app-topbar>` once in `AdminShellComponent` while child admin pages swap cleanly inside `<router-outlet />`.

What Topic Is Next → Next Topic 18.9: Declarative Navigation (`routerLink` and `routerLinkActive`)

18.9 Declarative Navigation (`routerLink` and `routerLinkActive`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `routerLink="/employees"` • Array link `[routerLink]="['/employees' • Emp.id]"` • `routerLinkActive="bg-indigo-600 text-white"` • UI, Accessibility & Edge-Case Pitfall Guard (Declarative Navigation (`routerLink` and `routerLinkActive`)) • Hands-On Frontend Implementation & Verification (Declarative Navigation (`routerLink` and `routerLinkActive`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Declarative Navigation (`routerLink` and `routerLink...

Purpose: Setting `{ exact: true }` on the root `/admin` link so it isn't permanently highlighted on `/admin/users` and `/admin/orders`!

What Topic Is Next → Next Topic 18.10: Dynamic Route Parameters (`ActivatedRoute` & Input Binding)

18.10 Dynamic Route Parameters (`ActivatedRoute` & Input Binding) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reading `:id` reactively via `ActivatedRoute.paramMap` (or `withComponentInputBinding()` `id = input.required<string>()`) • Reacting when navigating from `/users/1` to `/users/2` • UI, Accessibility & Edge-Case Pitfall Guard (Dynamic Route Parameters (`ActivatedRoute` & Input Binding)) • Hands-On Frontend Implementation & Verification (Dynamic Route Parameters (`ActivatedRoute` & Input Binding)) • Dynamic Route Parameters (`ActivatedRoute` & Input Binding) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Route Parameters (`ActivatedRoute` & Input B... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Subscribing to `route.paramMap.pipe(switchMap(p => api.getById(p.get('id')!)))` so clicking "Next Employee" updates data without remounting!

What Topic Is Next → Next Topic 18.11: Query Parameters (`?search=&status=&page=`)

18.11 Query Parameters (`?search=&status=&page=`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `[queryParams]="{ page: 2 }"` • `queryParamsHandling="merge"` • Programmatic `router.navigate([] • { queryParams • UI, Accessibility & Edge-Case Pitfall Guard (Query Parameters (`?search=&status=&page=`)) • Hands-On Frontend Implementation & Verification (Query Parameters (`?search=&status=&page=`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Query Parameters (`?search=&status=&page=`)

Purpose: Syncing table search, status filter, sort column, and page number to URL query params so bookmarked/shared admin links restore exact filters!

What Topic Is Next → Next Topic 18.12: Nested Child Routes (`children: [...]`)

18.12 Nested Child Routes (`children: [...]`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-level layouts: Auth layout (`/login` • `/forgot-password`) vs Admin layout (`/admin/overview` • `/admin/users` • `/admin/settings`) • UI, Accessibility & Edge-Case Pitfall Guard (Nested Child Routes (`children: [...]`)) • Hands-On Frontend Implementation & Verification (Nested Child Routes (`children: [...]`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Nested Child Routes (`children: [...]`)

Purpose: Nesting `Profile`, `Attendance`, and `Payroll` child routes inside `/employees/:id` with a shared employee header card.

What Topic Is Next → Next Topic 18.13: Lazy-Loaded Feature Routes (`loadComponent` & `loadChildren`)

18.13 Lazy-Loaded Feature Routes (`loadComponent` & `loadChildren`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Splitting standalone pages via `loadComponent: () => import(...).then(m => m.UsersPageComponent)` • Entire feature route files via `loadChildren` • UI, Accessibility & Edge-Case Pitfall Guard (Lazy-Loaded Feature Routes (`loadComponent` & `loadChildren`)) • Hands-On Frontend Implementation & Verification (Lazy-Loaded Feature Routes (`loadComponent` & `loadChildren`)) • Lazy-Loaded Feature Routes (`loadComponent` & `loadChildren`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Lazy-Loaded Feature Routes (`loadComponent` & `loadC... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Keeping initial `main.js` small by lazy-loading `/reports`, `/audit-logs`, and `/settings` feature route trees on navigation.

What Topic Is Next → Next Topic 18.14: Functional Route Guards (`CanActivateFn`, `CanMatchFn`, `CanDeactivateFn`)

18.14 Functional Route Guards (`CanActivateFn`, `CanMatchFn`, `CanDeactivateFn`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Returning `boolean | UrlTree` (`router.createUrlTree(['/login'] • { queryParams: { returnUrl: state.url } })`) • Unsaved-form navigation warning via `CanDeactivateFn` • UI, Accessibility & Edge-Case Pitfall Guard (Functional Route Guards (`CanActivateFn`, `CanMatchFn`, `CanDeactivateFn`)) • Hands-On Frontend Implementation & Verification (Functional Route Guards (`CanActivateFn`, `CanMatchFn`, `CanDeactivateFn`)) • Functional Route Guards (`CanActivateFn`, `CanMatchFn`, `CanDeactivateFn`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Functional Route Guards (`CanActivateFn`, `CanMatchF...

Purpose: Redirecting unauthenticated visitors via `UrlTree` (instead of side-effect `router.navigate` + `return false`) to avoid router race conditions!

What Topic Is Next → Next Topic 18.15: Route Resolvers (`ResolveFn`) and Pre-Navigation Data

18.15 Route Resolvers (`ResolveFn`) and Pre-Navigation Data [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Writing `ResolveFn<Employee>` vs skeleton-in-component UX trade-offs (when pre-fetching blocks navigation vs immediate shell + skeleton) • UI, Accessibility & Edge-Case Pitfall Guard (Route Resolvers (`ResolveFn`) and Pre-Navigation Data) • Hands-On Frontend Implementation & Verification (Route Resolvers (`ResolveFn`) and Pre-Navigation Data) • Route Resolvers (`ResolveFn`) and Pre-Navigation Data — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Route Resolvers (`ResolveFn`) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Understanding when lightweight metadata/lookup resolvers help vs why modern UX prefers instant route transition + in-page skeleton loaders.

What Topic Is Next → Next Topic 18.16: Configuring `provideHttpClient` and `HttpClient` Fundamentals

18.16 Configuring `provideHttpClient` and `HttpClient` Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `provideHttpClient(withFetch() • WithInterceptors([...]))` in `app.config.ts` • Cold Observable nature of `HttpClient` (request fires ONLY on `.subscribe()` or `| async` / `toSignal`!) • Auto JSON parsing • UI, Accessibility & Edge-Case Pitfall Guard (Configuring `provideHttpClient` and `HttpClient` Fundamentals) • Hands-On Frontend Implementation & Verification (Configuring `provideHttpClient` and `HttpClient` Fundamentals) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Configuring `provideHttpClient`

Purpose: Typing `this.http.get<PagedResponse<Employee>>(...)` for compile-time DTO verification across the service layer.

What Topic Is Next → Next Topic 18.17: Typed HTTP `GET` Requests with `HttpParams` and `HttpHeaders`

18.17 Typed HTTP `GET` Requests with `HttpParams` and `HttpHeaders` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable `new HttpParams().set('page' • Page).set('q' • Query)` builder • Typed generics `http.get<Employee[]>(url • UI, Accessibility & Edge-Case Pitfall Guard (Typed HTTP `GET` Requests with `HttpParams` and `HttpHeaders`) • Hands-On Frontend Implementation & Verification (Typed HTTP `GET` Requests with `HttpParams` and `HttpHeaders`) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Typed HTTP `GET` Requests with `HttpParams`

Purpose: Building safe URL-encoded query parameters via immutable `HttpParams` instead of manual string concatenation.

What Topic Is Next → Next Topic 18.18: HTTP `POST` Requests (Creating Resources)

18.18 HTTP `POST` Requests (Creating Resources) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sending typed `CreateEmployeeDto` JSON payloads `this.http.post<Employee>(url • Dto)` • Disabling submit controls while request is in flight • UI, Accessibility & Edge-Case Pitfall Guard (HTTP `POST` Requests (Creating Resources)) • Hands-On Frontend Implementation & Verification (HTTP `POST` Requests (Creating Resources)) • HTTP `POST` Requests (Creating Resources) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP `POST` Requests (Creating Resources)

Purpose: Creating new employee/product records with typed request & response interfaces and `finalize(() => this.saving.set(false))`.

What Topic Is Next → Next Topic 18.19: HTTP `PUT` and `PATCH` Requests (Updating Resources)

18.19 HTTP `PUT` and `PATCH` Requests (Updating Resources) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Full replacement `PUT /employees/:id` vs partial `PATCH /employees/:id/status` (`Partial<UpdateEmployeeDto>`) • Optimistic vs confirmed state refresh • UI, Accessibility & Edge-Case Pitfall Guard (HTTP `PUT` and `PATCH` Requests (Updating Resources)) • Hands-On Frontend Implementation & Verification (HTTP `PUT` and `PATCH` Requests (Updating Resources)) • HTTP `PUT` and `PATCH` Requests (Updating Resources) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP `PUT` • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Updating an employee's active status via `PATCH` and editing full profile details via `PUT`.

What Topic Is Next → Next Topic 18.20: HTTP `DELETE` Requests and Safe Confirmation Flows

18.20 HTTP `DELETE` Requests and Safe Confirmation Flows [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `this.http.delete<void>(`${url}/${id}`)` • Confirmation dialog before destructive deletion • Handling `204 No Content` responses • UI, Accessibility & Edge-Case Pitfall Guard (HTTP `DELETE` Requests and Safe Confirmation Flows) • Hands-On Frontend Implementation & Verification (HTTP `DELETE` Requests and Safe Confirmation Flows) • HTTP `DELETE` Requests and Safe Confirmation Flows — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP `DELETE` Requests

Purpose: Guarding row deletions behind an accessible confirmation modal and refreshing list state cleanly upon `204` completion.

What Topic Is Next → Next Topic 18.21: Functional HTTP Interceptors (`HttpInterceptorFn`)

18.21 Functional HTTP Interceptors (`HttpInterceptorFn`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cloning immutable requests `req.clone({ setHeaders: { Authorization: `Bearer ${token}` } })` • Chaining `next(authReq)` • Logging request duration with `tap`/`finalize` • UI, Accessibility & Edge-Case Pitfall Guard (Functional HTTP Interceptors (`HttpInterceptorFn`)) • Hands-On Frontend Implementation & Verification (Functional HTTP Interceptors (`HttpInterceptorFn`)) • Functional HTTP Interceptors (`HttpInterceptorFn`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Functional HTTP Interceptors (`HttpInterceptorFn`)

Purpose: Attaching JWT Bearer tokens and `X-Request-Id` headers centrally in 12 lines via `authInterceptor: HttpInterceptorFn`.

What Topic Is Next → Next Topic 18.22: Centralized RxJS HTTP Error Handling (`HttpErrorResponse`)

18.22 Centralized RxJS HTTP Error Handling (`HttpErrorResponse`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Distinguishing network failure (`err.status === 0`) vs validation (`422`) vs expired session (`401`) vs forbidden (`403`) vs server error (`500+`) • `catchError` + `throwError(() => domainErr)` • UI, Accessibility & Edge-Case Pitfall Guard (Centralized RxJS HTTP Error Handling (`HttpErrorResponse`)) • Hands-On Frontend Implementation & Verification (Centralized RxJS HTTP Error Handling (`HttpErrorResponse`)) • Centralized RxJS HTTP Error Handling (`HttpErrorResponse`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Centralized RxJS HTTP Error Handling (`HttpErrorResp... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Mapping raw `HttpErrorResponse` payloads into user-friendly domain messages in an `errorInterceptor` while rethrowing for field-level `422` forms.

What Topic Is Next → Next Topic 18.23: End-to-End Angular Authentication Workflow

18.23 End-to-End Angular Authentication Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `AuthService` holding `currentUser = signal<User | null>(null)` + `isAuthenticated = computed(...)` • Login/session restore on startup (`APP_INITIALIZER` / `provideAppInitializer`) • Logout cleanup • UI, Accessibility & Edge-Case Pitfall Guard (End-to-End Angular Authentication Workflow) • Hands-On Frontend Implementation & Verification (End-to-End Angular Authentication Workflow) • End-to-End Angular Authentication Workflow — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for End-to-End Angular Authentication Workflow

Purpose: Restoring authenticated user state cleanly before initial route guard evaluation so refreshing `/admin/users` doesn't falsely kick logged-in admins to `/login`!

What Topic Is Next → Next Topic 18.24: Role-Based Protected Routes (`ADMIN`, `MANAGER`, `VIEWER`)

18.24 Role-Based Protected Routes (`ADMIN`, `MANAGER`, `VIEWER`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Parameterized `hasRoleGuard(['ADMIN' • 'MANAGER'])` • Hiding unauthorized UI actions via custom `*appHasRole` / `@if (auth.hasRole('ADMIN'))` alongside route guards • UI, Accessibility & Edge-Case Pitfall Guard (Role-Based Protected Routes (`ADMIN`, `MANAGER`, `VIEWER`)) • Hands-On Frontend Implementation & Verification (Role-Based Protected Routes (`ADMIN`, `MANAGER`, `VIEWER`)) • Role-Based Protected Routes (`ADMIN`, `MANAGER`, `VIEWER`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Role-Based Protected Routes (`ADMIN`, `MANAGER`, `VI...

Purpose: Restricting `/admin/payroll` and Delete actions to `ADMIN` role while allowing `VIEWER` read-only dashboard access.

What Topic Is Next → Module 18 Capstone (Week 18 Lab Build: Guarded Angular Router & `HttpClient` Interceptor Portal) → Next Topic 19.1: Template-Driven Forms (`FormsModule` and `ngForm`)

Module 18 Hands-On Capstone Project (100 Marks): Week 18 Lab Build: Guarded Angular Router & `HttpClient` Interceptor Portal — Build a multi-route Angular portal with nested `<router-outlet>`, lazy `loadComponent` routes, functional `authGuard`/`roleGuard`, typed `HttpClient` CRUD service, and `authInterceptor` + `e...

MODULE 19 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 25 TOPICS (200 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 19: Angular Forms, RxJS and State Management

Master Angular Template-Driven vs Strictly Typed Reactive Forms (`FormGroup`, `FormControl`, `NonNullableFormBuilder`, sync/async custom validators, dynamic `FormArray`), deep RxJS reactive streams (`Observable`, `Subject`, `BehaviorSubject`, `switchMap`/`mergeMap`/`concatMap`/`exhaustMap`, `combineLatest`, `AsyncPipe`, Signals interop), and NgRx Store architecture.

Objectives: ✓ Build complex, strictly typed Angular Reactive Forms using `NonNullableFormBuilder`, cross-field validators, async uniqueness validators, and dynamic `FormArray` line items. • ✓ Think reactively with RxJS Observables, Subjects, `BehaviorSubject`, automatic teardown (`takeUntilDestroyed` / `AsyncPipe`), and Signal interop (`toSignal` / `toObservable`). • ✓ Choose the exact flattening operator (`switchMap` for search cancel, `exhaustMap` for login double-click prevention, `concatMap` for ordered saves, `mergeMap` for parallel tasks) in real apps and interviews. • ✓ Manage shared Angular state cleanly using Signal/BehaviorSubject Services AND enterprise NgRx (`Store`, `createActionGroup`, `createReducer`, `createEffect`, `createSelector`).

19.1 Template-Driven Forms (`FormsModule` and `ngForm`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `#f="ngForm"` • `ngModel` with `name` • `required`/`email`/`minlength` attributes • `#emailCtrl="ngModel"` `dirty`/`touched`/`invalid` CSS states (`ng-touched` • UI, Accessibility & Edge-Case Pitfall Guard (Template-Driven Forms (`FormsModule` and `ngForm`)) • Hands-On Frontend Implementation & Verification (Template-Driven Forms (`FormsModule` and `ngForm`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Template-Driven Forms (`FormsModule` and `ngForm`)

Purpose: Understanding quick 2-field newsletter/filter forms and maintaining existing Template-Driven forms in legacy Angular modules.

What Topic Is Next → Next Topic 19.2: Strictly Typed Reactive Forms (`ReactiveFormsModule`, `FormControl`, `FormGroup`)

19.2 Strictly Typed Reactive Forms (`ReactiveFormsModule`, `FormControl`, `FormGroup`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Explicit TypeScript model in TS class (`FormControl<string>` • `FormGroup<{...}>`) • `[formGroup]` • `formControlName` • UI, Accessibility & Edge-Case Pitfall Guard (Strictly Typed Reactive Forms (`ReactiveFormsModule`, `FormControl`, `FormGroup`)) • Hands-On Frontend Implementation & Verification (Strictly Typed Reactive Forms (`ReactiveFormsModule`, `FormControl`, `FormGroup`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Strictly Typed Reactive Forms (`ReactiveFormsModule`...

Purpose: Testing complex 20-field enterprise payroll/tax validation rules in plain TypeScript unit tests in 5ms!

What Topic Is Next → Next Topic 19.3: `NonNullableFormBuilder` Ergonomics

19.3 `NonNullableFormBuilder` Ergonomics [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Injecting `fb = inject(NonNullableFormBuilder)` • `fb.group({ ... })` • `patchValue` (partial fill) vs `setValue` (strict full shape) • `reset()` restoring initial defaults • UI, Accessibility & Edge-Case Pitfall Guard (`NonNullableFormBuilder` Ergonomics) • Hands-On Frontend Implementation & Verification (`NonNullableFormBuilder` Ergonomics) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for `NonNullableFormBuilder` Ergonomics

Purpose: Pre-populating edit forms cleanly via `this.form.patchValue(employee)` when loading an existing record from the API.

What Topic Is Next → Next Topic 19.4: Built-In Validators (`Validators`) and Accessible Error Display

19.4 Built-In Validators (`Validators`) and Accessible Error Display [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `required` • `email` • `minLength` • `maxLength` • UI, Accessibility & Edge-Case Pitfall Guard (Built-In Validators (`Validators`) and Accessible Error Display) • Hands-On Frontend Implementation & Verification (Built-In Validators (`Validators`) and Accessible Error Display) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Built-In Validators (`Validators`)

Purpose: Calling `if (this.form.invalid) { this.form.markAllAsTouched(); return; }` so clicking Submit immediately highlights every invalid field!

What Topic Is Next → Next Topic 19.5: Custom Synchronous & Asynchronous Validators (`ValidatorFn`, `AsyncValidatorFn`)

19.5 Custom Synchronous & Asynchronous Validators (`ValidatorFn`, `AsyncValidatorFn`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Field `ValidatorFn` returning `ValidationErrors | null` • Cross-field group validator (`password === confirmPassword` • `endDate >= startDate`) • Debounced `AsyncValidatorFn` checking email availability • UI, Accessibility & Edge-Case Pitfall Guard (Custom Synchronous & Asynchronous Validators (`ValidatorFn`, `AsyncValidatorFn`)) • Hands-On Frontend Implementation & Verification (Custom Synchronous & Asynchronous Validators (`ValidatorFn`, `AsyncValidatorFn`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Custom Synchronous

Purpose: Validating Indian PAN/GSTIN format synchronously and checking employee-code uniqueness asynchronously on `updateOn: 'blur'`.

What Topic Is Next → Next Topic 19.6: Dynamic Forms with `FormArray`

19.6 Dynamic Forms with `FormArray` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `this.fb.array([this.createSkillGroup()])` • `addSkill()` • `removeAt(index)` • `formArrayName` + `@for (ctrl of skills.controls • UI, Accessibility & Edge-Case Pitfall Guard (Dynamic Forms with `FormArray`) • Hands-On Frontend Implementation & Verification (Dynamic Forms with `FormArray`) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Forms with `FormArray`

Purpose: Building multi-row Invoice Line Items, Employee Work Experience entries, or Dynamic Filter Rule Builders with live row totals.

What Topic Is Next → Next Topic 19.7: Enterprise Form UX: Dirty State, Conditional Validators & Server `422` Mapping

19.7 Enterprise Form UX: Dirty State, Conditional Validators & Server `422` Mapping [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reacting to `roleControl.valueChanges` via `setValidators` + `updateValueAndValidity()` • Mapping backend `422` field errors via `ctrl.setErrors({ server: msg })` • UI, Accessibility & Edge-Case Pitfall Guard (Enterprise Form UX: Dirty State, Conditional Validators & Server `422` Mapping) • Hands-On Frontend Implementation & Verification (Enterprise Form UX: Dirty State, Conditional Validators & Server `422` Mapping) • Enterprise Form UX: Dirty State, Conditional Validators & Server `422` Mapping — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Enterprise Form UX • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Making `contractorVendorName` required dynamically ONLY when `employmentType === 'CONTRACT'`, and always calling `updateValueAndValidity()`!

What Topic Is Next → Next Topic 19.8: Introduction to Reactive Programming & RxJS

19.8 Introduction to Reactive Programming & RxJS [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Push vs Pull streams • Treating events • Form valueChanges • Router params • UI, Accessibility & Edge-Case Pitfall Guard (Introduction to Reactive Programming & RxJS) • Hands-On Frontend Implementation & Verification (Introduction to Reactive Programming & RxJS) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Introduction to Reactive Programming

Purpose: Combining search input + department dropdown + page changes into a single declarative HTTP stream without nested flags or race bugs.

What Topic Is Next → Next Topic 19.9: Observables: Cold vs Hot Streams (`of`, `from`, `interval`, `timer`, `shareReplay`)

19.9 Observables: Cold vs Hot Streams (`of`, `from`, `interval`, `timer`, `shareReplay`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Observable vs Promise (lazy • Multi-value over time • Cancellable `.unsubscribe()`!) • Unicast Cold HTTP streams vs multicast Hot `shareReplay({ bufferSize: 1 • UI, Accessibility & Edge-Case Pitfall Guard (Observables: Cold vs Hot Streams (`of`, `from`, `interval`, `timer`, `shareReplay`)) • Hands-On Frontend Implementation & Verification (Observables: Cold vs Hot Streams (`of`, `from`, `interval`, `timer`, `shareReplay`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Observables: Cold vs Hot Streams (`of`, `from`, `int...

Purpose: Using `shareReplay({ bufferSize: 1, refCount: true })` so 3 `| async` bindings in a template share ONE HTTP request instead of firing 3 duplicate GETs!

What Topic Is Next → Next Topic 19.10: Observers (`next`, `error`, `complete`) and Stream Contract

19.10 Observers (`next`, `error`, `complete`) and Stream Contract [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: RxJS stream grammar: `0..N` `next` emissions followed optionally by EITHER one `complete` OR one `error` (terminal!) • Why uncaught errors kill long-lived streams • UI, Accessibility & Edge-Case Pitfall Guard (Observers (`next`, `error`, `complete`) and Stream Contract) • Hands-On Frontend Implementation & Verification (Observers (`next`, `error`, `complete`) and Stream Contract) • Observers (`next`, `error`, `complete`) and Stream Contract — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Observers (`next`, `error`, `complete`) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Placing `catchError(...)` INSIDE the inner `switchMap` HTTP call so one failed search query does NOT permanently kill the outer `searchControl.valueChanges` stream!

What Topic Is Next → Next Topic 19.11: Subscription Lifecycle and Automatic Teardown (`takeUntilDestroyed`)

19.11 Subscription Lifecycle and Automatic Teardown (`takeUntilDestroyed`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `HttpClient` one-shot streams auto-complete vs why `interval` • `valueChanges` • `route.paramMap` • `Subject` leak memory if unsubscribed • UI, Accessibility & Edge-Case Pitfall Guard (Subscription Lifecycle and Automatic Teardown (`takeUntilDestroyed`)) • Hands-On Frontend Implementation & Verification (Subscription Lifecycle and Automatic Teardown (`takeUntilDestroyed`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Subscription Lifecycle

Purpose: Piping `.pipe(takeUntilDestroyed(this.destroyRef))` on every manual long-lived subscription when `AsyncPipe` or `toSignal` isn't used.

What Topic Is Next → Next Topic 19.12: Core Pipeable Operators (`map`, `filter`, `tap`, `debounceTime`, `distinctUntilChanged`, `startWith`)

19.12 Core Pipeable Operators (`map`, `filter`, `tap`, `debounceTime`, `distinctUntilChanged`, `startWith`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Transforming emissions (`map`) • Guarding (`filter`) • Side-effect logging (`tap`) • Deduplicating identical trimmed search terms (`distinctUntilChanged()`) • UI, Accessibility & Edge-Case Pitfall Guard (Core Pipeable Operators (`map`, `filter`, `tap`, `debounceTime`, `distinctUntilChanged`, `startWith`)) • Hands-On Frontend Implementation & Verification (Core Pipeable Operators (`map`, `filter`, `tap`, `debounceTime`, `distinctUntilChanged`, `startWith`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Core Pipeable Operators (`map`, `filter`, `tap`, `de...

Purpose: Building `valueChanges.pipe(startWith(''), map(s => s.trim()), debounceTime(300), distinctUntilChanged())` for clean typeahead streams.

What Topic Is Next → Next Topic 19.13: Higher-Order Flattening Operators (`switchMap`, `mergeMap`, `concatMap`, `exhaustMap`)

19.13 Higher-Order Flattening Operators (`switchMap`, `mergeMap`, `concatMap`, `exhaustMap`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `switchMap` (cancel previous in-flight: live search/route param) • `exhaustMap` (ignore clicks while busy: login/checkout submit!) • `concatMap` (strict FIFO queue: sequential updates) • `mergeMap` (parallel fan-out) • UI, Accessibility & Edge-Case Pitfall Guard (Higher-Order Flattening Operators (`switchMap`, `mergeMap`, `concatMap`, `exhaustMap`)) • Hands-On Frontend Implementation & Verification (Higher-Order Flattening Operators (`switchMap`, `mergeMap`, `concatMap`, `exhaustMap`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Higher-Order Flattening Operators (`switchMap`, `mer...

Purpose: Preventing double-charge order submissions with `exhaustMap` and cancelling stale search XHRs automatically with `switchMap`!

What Topic Is Next → Next Topic 19.14: Combination Operators (`combineLatest`, `forkJoin`, `withLatestFrom`)

19.14 Combination Operators (`combineLatest`, `forkJoin`, `withLatestFrom`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `forkJoin({ depts • Roles • Stats })` (`Promise.all` equivalent waiting for completion) vs `combineLatest([query$ • Dept$ • UI, Accessibility & Edge-Case Pitfall Guard (Combination Operators (`combineLatest`, `forkJoin`, `withLatestFrom`)) • Hands-On Frontend Implementation & Verification (Combination Operators (`combineLatest`, `forkJoin`, `withLatestFrom`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Combination Operators (`combineLatest`, `forkJoin`,...

Purpose: Loading initial reference dropdowns in parallel via `forkJoin` and driving reactive table reloads via `combineLatest([search$, status$, page$])`.

What Topic Is Next → Next Topic 19.15: Resilient Stream Error Recovery (`catchError` and `retry`)

19.15 Resilient Stream Error Recovery (`catchError` and `retry`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `retry({ count: 2 • Delay: 800 })` on transient `502`/`503` GET errors (never blindly retry non-idempotent POSTs!) • Fallback state mapping via `catchError` • UI, Accessibility & Edge-Case Pitfall Guard (Resilient Stream Error Recovery (`catchError` and `retry`)) • Hands-On Frontend Implementation & Verification (Resilient Stream Error Recovery (`catchError` and `retry`)) • Resilient Stream Error Recovery (`catchError` and `retry`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Resilient Stream Error Recovery (`catchError` and `r...

Purpose: Retrying transient network hiccups on read-only dashboard analytics endpoints with controlled backoff.

What Topic Is Next → Next Topic 19.16: Multicasting with `Subject`, `BehaviorSubject` and `ReplaySubject`

19.16 Multicasting with `Subject`, `BehaviorSubject` and `ReplaySubject` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Subject<void>` event bus • `BehaviorSubject<T>(initial)` stateful stream with `.getValue()` & instant replay to late subscribers • Exposing read-only `.asObservable()` • UI, Accessibility & Edge-Case Pitfall Guard (Multicasting with `Subject`, `BehaviorSubject` and `ReplaySubject`) • Hands-On Frontend Implementation & Verification (Multicasting with `Subject`, `BehaviorSubject` and `ReplaySubject`) • Multicasting with `Subject`, `BehaviorSubject` and `ReplaySubject` — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Multicasting with `Subject`

Purpose: Encapsulating `private readonly userSubject = new BehaviorSubject<User | null>(null)` while exposing `readonly user$ = this.userSubject.asObservable()`.

What Topic Is Next → Next Topic 19.17: Declarative Template Subscriptions with `AsyncPipe` (`| async`)

19.17 Declarative Template Subscriptions with `AsyncPipe` (`| async`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Auto-subscribe on init • Auto-markForCheck under `OnPush` change detection • Auto-unsubscribe on destroy • Single subscription aliasing `@if (vm$ | async • UI, Accessibility & Edge-Case Pitfall Guard (Declarative Template Subscriptions with `AsyncPipe` (`| async`)) • Hands-On Frontend Implementation & Verification (Declarative Template Subscriptions with `AsyncPipe` (`| async`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Declarative Template Subscriptions with `AsyncPipe`...

Purpose: Composing a single `vm$` view-model stream (`{ loading, data, error }`) consumed once via `@if (vm$ | async; as vm)` with zero manual `.subscribe()`!

What Topic Is Next → Next Topic 19.18: Signals and RxJS Interoperability (`toSignal` and `toObservable`)

19.18 Signals and RxJS Interoperability (`toSignal` and `toObservable`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@angular/core/rxjs-interop`: converting HTTP/Router Observables to synchronous template Signals via `toSignal(obs$ • { initialValue: [] })` • Debouncing a Signal via `toObservable(query)` • UI, Accessibility & Edge-Case Pitfall Guard (Signals and RxJS Interoperability (`toSignal` and `toObservable`)) • Hands-On Frontend Implementation & Verification (Signals and RxJS Interoperability (`toSignal` and `toObservable`)) • Signals and RxJS Interoperability (`toSignal` and `toObservable`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Signals

Purpose: Using `toObservable(this.searchQuery).pipe(debounceTime(300), switchMap(...))` -> `toSignal(...)` to get the best of RxJS async operators AND ergonomic Signal templates!

What Topic Is Next → Next Topic 19.19: Service-Based Reactive State Management (Signals / `BehaviorSubject` Store)

19.19 Service-Based Reactive State Management (Signals / `BehaviorSubject` Store) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When clean Signal/RxJS Store Services cover 85% of feature state needs without full Redux/NgRx ceremony: immutable `.update()` • Computed selectors • Async methods • UI, Accessibility & Edge-Case Pitfall Guard (Service-Based Reactive State Management (Signals / `BehaviorSubject` Store)) • Hands-On Frontend Implementation & Verification (Service-Based Reactive State Management (Signals / `BehaviorSubject` Store)) • Service-Based Reactive State Management (Signals / `BehaviorSubject` Store) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service-Based Reactive State Management (Signals / `...

Purpose: Building a clean 50-line `EmployeeStoreService` with Signals (`employees`, `loading`, `error`, `filteredEmployees`) for medium-sized enterprise modules.

What Topic Is Next → Next Topic 19.20: NgRx Fundamentals and Enterprise Redux Architecture in Angular

19.20 NgRx Fundamentals and Enterprise Redux Architecture in Angular [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@ngrx/store` • `@ngrx/effects` • `@ngrx/store-devtools` • When large banking/ERP Angular teams adopt NgRx (strict action audit log • UI, Accessibility & Edge-Case Pitfall Guard (NgRx Fundamentals and Enterprise Redux Architecture in Angular) • Hands-On Frontend Implementation & Verification (NgRx Fundamentals and Enterprise Redux Architecture in Angular) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for NgRx Fundamentals

Purpose: Working productively in enterprise Angular monorepos standardized on `@ngrx/store` + `@ngrx/effects` (or modern `@ngrx/signals` `signalStore`).

What Topic Is Next → Next Topic 19.21: NgRx Centralized Store and Feature State (`createFeature`)

19.21 NgRx Centralized Store and Feature State (`createFeature`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Defining typed `EmployeesState` (`ids` • `entities` • `loading` • `error`) • UI, Accessibility & Edge-Case Pitfall Guard (NgRx Centralized Store and Feature State (`createFeature`)) • Hands-On Frontend Implementation & Verification (NgRx Centralized Store and Feature State (`createFeature`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for NgRx Centralized Store

Purpose: Cutting NgRx boilerplate in half with modern `createFeature({ name: 'employees', reducer })`.

What Topic Is Next → Next Topic 19.22: NgRx Actions (`createAction` and `createActionGroup`)

19.22 NgRx Actions (`createAction` and `createActionGroup`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Good Action Hygiene (event naming `[Employees Page] Opened` vs `[Employees API] Load Success`) • Grouping related actions cleanly with `createActionGroup` + `props<T>()` / `emptyProps()` • UI, Accessibility & Edge-Case Pitfall Guard (NgRx Actions (`createAction` and `createActionGroup`)) • Hands-On Frontend Implementation & Verification (NgRx Actions (`createAction` and `createActionGroup`)) • NgRx Actions (`createAction` and `createActionGroup`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for NgRx Actions (`createAction` and `createActionGroup`... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Defining `EmployeesActions` (`opened`, `loadSuccess`, `loadFailure`, `deleted`) in 10 concise lines via `createActionGroup`.

What Topic Is Next → Next Topic 19.23: NgRx Pure Reducers (`createReducer` and `on`)

19.23 NgRx Pure Reducers (`createReducer` and `on`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable spread updates `on(EmployeesActions.loadSuccess • (state • { items }) => ({ ...state • Items • UI, Accessibility & Edge-Case Pitfall Guard (NgRx Pure Reducers (`createReducer` and `on`)) • Hands-On Frontend Implementation & Verification (NgRx Pure Reducers (`createReducer` and `on`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for NgRx Pure Reducers (`createReducer` and `on`)

Purpose: Unit-testing NgRx state transitions with plain synchronous `expect(reducer(initialState, action)).toEqual(...)` assertions.

What Topic Is Next → Next Topic 19.24: NgRx Side Effects (`createEffect` and `ofType`)

19.24 NgRx Side Effects (`createEffect` and `ofType`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Listening to `actions$.pipe(ofType(EmployeesActions.pageOpened) • SwitchMap(() => api.list().pipe(map(LoadSuccess) • CatchError(err => of(LoadFailure)))))` • `{ dispatch: false }` toast/router effects • UI, Accessibility & Edge-Case Pitfall Guard (NgRx Side Effects (`createEffect` and `ofType`)) • Hands-On Frontend Implementation & Verification (NgRx Side Effects (`createEffect` and `ofType`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for NgRx Side Effects (`createEffect` and `ofType`)

Purpose: Handling API requests, toast notifications, and post-login redirects cleanly inside functional `createEffect` streams.

What Topic Is Next → Next Topic 19.25: NgRx Memoized Selectors (`createSelector` & `store.selectSignal`)

19.25 NgRx Memoized Selectors (`createSelector` & `store.selectSignal`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Composing memoized derived selectors (`selectActiveEmployees` • `selectDepartmentCounts`) • Reading selectors as Signals in templates via `this.store.selectSignal(selectActiveEmployees)`! • UI, Accessibility & Edge-Case Pitfall Guard (NgRx Memoized Selectors (`createSelector` & `store.selectSignal`)) • Hands-On Frontend Implementation & Verification (NgRx Memoized Selectors (`createSelector` & `store.selectSignal`)) • NgRx Memoized Selectors (`createSelector` & `store.selectSignal`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for NgRx Memoized Selectors (`createSelector` & `store.s...

Purpose: Consuming NgRx state as synchronous Signals via `store.selectSignal(...)` with zero manual `AsyncPipe` boilerplate in templates!

What Topic Is Next → Module 19 Capstone (Week 19 Lab Build: Typed Angular Reactive `FormArray` + RxJS Live Search + NgRx Store) → Next Topic 20.1: Structuring Large Enterprise Angular Applications

Module 19 Hands-On Capstone Project (100 Marks): Week 19 Lab Build: Typed Angular Reactive `FormArray` + RxJS Live Search + NgRx Store — Implement a dynamic multi-item Invoice `FormArray` with custom sync/async validators, an RxJS `debounceTime` + `switchMap` search bar, and a typed NgRx feature slice (`createActionGroup`, `c...

MODULE 20 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 22 TOPICS (176 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 20: Angular Advanced Development and Angular Material

Scale large enterprise Angular applications with Standalone domain architecture, Angular Material & CDK (`MatTable`, `MatPaginator`, `MatSort`, `MatDialog`, `MatSidenav`, `MatMenu`, `MatAutocomplete`, `BreakpointObserver`), `ChangeDetectionStrategy.OnPush`, dynamic component rendering, unit/HTTP testing (`TestBed`, `HttpTestingController`), production build budgets, and build Major Project 5: Enterprise Admin Dashboard using Angular.

Objectives: ✓ Organize large Angular codebases into `core/`, `shared/`, and lazy-loaded `features/` while understanding legacy `FeatureModule` boundaries. • ✓ Build polished enterprise admin interfaces rapidly using Angular Material 3 theming, `MatTable`, `MatPaginator`, `MatSort`, `MatDialog`, `MatSidenav`, `MatAutocomplete`, and CDK `BreakpointObserver`. • ✓ Optimize Angular rendering performance with `ChangeDetectionStrategy.OnPush`, Signals, `AsyncPipe`, and lazy feature chunks. • ✓ Write Angular unit and HTTP service tests using `TestBed` and `HttpTestingController`, verify `ng build` bundle budgets, and deliver **Project 5: Enterprise Admin Dashboard using Angular**.

20.1 Structuring Large Enterprise Angular Applications [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Clean layering: `src/app/core/` (auth • Guards • Interceptors • Layout shell) • UI, Accessibility & Edge-Case Pitfall Guard (Structuring Large Enterprise Angular Applications) • Hands-On Frontend Implementation & Verification (Structuring Large Enterprise Angular Applications) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Structuring Large Enterprise Angular Applications

Purpose: Keeping 40+ enterprise screens maintainable with strict one-way dependency rules (`features -> shared/core`, never `shared -> features`!).

What Topic Is Next → Next Topic 20.2: Legacy Feature Modules (`NgModule`) Awareness for Enterprise Maintenance

20.2 Legacy Feature Modules (`NgModule`) Awareness for Enterprise Maintenance [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reading `CommonModule` • `SharedModule` • `CoreModule` • `RouterModule.forRoot` vs `forChild` in pre-v17 Angular codebases • UI, Accessibility & Edge-Case Pitfall Guard (Legacy Feature Modules (`NgModule`) Awareness for Enterprise Maintenance) • Hands-On Frontend Implementation & Verification (Legacy Feature Modules (`NgModule`) Awareness for Enterprise Maintenance) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Legacy Feature Modules (`NgModule`) Awareness for En...

Purpose: Confidently onboarding onto an existing bank/insurance Angular codebase that mixes older `NgModule` features with newer Standalone screens.

What Topic Is Next → Next Topic 20.3: Modern Standalone Application Architecture at Scale

20.3 Modern Standalone Application Architecture at Scale [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Route-level providers (`providers: [FeatureStore]` inside `Routes`) • Granular component `imports` arrays keeping lazy chunks lean • UI, Accessibility & Edge-Case Pitfall Guard (Modern Standalone Application Architecture at Scale) • Hands-On Frontend Implementation & Verification (Modern Standalone Application Architecture at Scale) • Modern Standalone Application Architecture at Scale — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Modern Standalone Application Architecture at Scale • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Scoping feature state/effects providers directly to `{ path: 'analytics', providers: [...], loadChildren: ... }` so analytics code loads ONLY when visited.

What Topic Is Next → Next Topic 20.4: Designing Reusable Shared UI Components in Angular

20.4 Designing Reusable Shared UI Components in Angular [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Stateless `<app-page-header>` • `<app-stat-kpi-card>` • `<app-empty-state>` • `<app-status-chip>` • UI, Accessibility & Edge-Case Pitfall Guard (Designing Reusable Shared UI Components in Angular) • Hands-On Frontend Implementation & Verification (Designing Reusable Shared UI Components in Angular) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Designing Reusable Shared UI Components in Angular

Purpose: Using multi-slot `<ng-content select="[header-actions]">` so every admin screen shares identical spacing, breadcrumbs, and typography.

What Topic Is Next → Next Topic 20.5: Angular Material Setup and Material 3 Custom Theming

20.5 Angular Material Setup and Material 3 Custom Theming [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ng add @angular/material` • Material 3 `mat.theme(...)` SCSS tokens • Typography & density scales (`density: -1` compact enterprise admin density!) • Dark/light theme support • UI, Accessibility & Edge-Case Pitfall Guard (Angular Material Setup and Material 3 Custom Theming) • Hands-On Frontend Implementation & Verification (Angular Material Setup and Material 3 Custom Theming) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Angular Material Setup

Purpose: Applying compact `density: -1` on enterprise admin tables and forms so operators see 35% more actionable rows without clutter.

What Topic Is Next → Next Topic 20.6: Angular Material Form Controls (`MatFormField`, `MatInput`, `MatSelect`, `MatDatepicker`)

20.6 Angular Material Form Controls (`MatFormField`, `MatInput`, `MatSelect`, `MatDatepicker`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<mat-form-field appearance="outline">` • `<mat-label>` • `<input matInput>` • `<mat-error>` • UI, Accessibility & Edge-Case Pitfall Guard (Angular Material Form Controls (`MatFormField`, `MatInput`, `MatSelect`, `MatDatepicker`)) • Hands-On Frontend Implementation & Verification (Angular Material Form Controls (`MatFormField`, `MatInput`, `MatSelect`, `MatDatepicker`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Angular Material Form Controls (`MatFormField`, `Mat...

Purpose: Building accessible, consistent enterprise Create/Edit forms where `<mat-error>` syncs automatically with Reactive Form touched/invalid states.

What Topic Is Next → Next Topic 20.7: Enterprise Data Tables with `MatTable` and `MatSort`

20.7 Enterprise Data Tables with `MatTable` and `MatSort` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `mat-table [dataSource]` • `ng-container matColumnDef` • `mat-header-cell` • `mat-cell` • UI, Accessibility & Edge-Case Pitfall Guard (Enterprise Data Tables with `MatTable` and `MatSort`) • Hands-On Frontend Implementation & Verification (Enterprise Data Tables with `MatTable` and `MatSort`) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Enterprise Data Tables with `MatTable`

Purpose: Rendering sortable, sticky-header enterprise inventory/user/order tables with formatted action menus and zero-results `*matNoDataRow`.

What Topic Is Next → Next Topic 20.8: Confirmation and Form Modals with `MatDialog`

20.8 Confirmation and Form Modals with `MatDialog` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `inject(MatDialog).open<ConfirmDialogComponent • ConfirmData • Boolean>(...)` • `MAT_DIALOG_DATA` • UI, Accessibility & Edge-Case Pitfall Guard (Confirmation and Form Modals with `MatDialog`) • Hands-On Frontend Implementation & Verification (Confirmation and Form Modals with `MatDialog`) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Confirmation

Purpose: Opening a reusable typed `<app-confirm-dialog>` returning `Observable<boolean>` (`filter(Boolean)`, `switchMap(() => api.delete(id))`) cleanly!

What Topic Is Next → Next Topic 20.9: Navigation Chrome with `MatSidenav`, `MatToolbar` and `MatMenu`

20.9 Navigation Chrome with `MatSidenav`, `MatToolbar` and `MatMenu` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<mat-sidenav-container>` • Desktop `mode="side"` vs mobile drawer `mode="over"` • Top `<mat-toolbar>` • Accessible row overflow actions `<mat-menu #rowMenu="matMenu">` • UI, Accessibility & Edge-Case Pitfall Guard (Navigation Chrome with `MatSidenav`, `MatToolbar` and `MatMenu`) • Hands-On Frontend Implementation & Verification (Navigation Chrome with `MatSidenav`, `MatToolbar` and `MatMenu`) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Navigation Chrome with `MatSidenav`

Purpose: Switching `<mat-sidenav>` automatically from persistent `side` rail on `>= 1024px` desktop to collapsible `over` drawer on tablets/phones.

What Topic Is Next → Next Topic 20.10: Client and Server-Side Pagination with `MatPaginator`

20.10 Client and Server-Side Pagination with `MatPaginator` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<mat-paginator [length]="totalCount()" [pageSize]="pageSize()" [pageSizeOptions]="[10 • 50]" (page)="onPageChange($event)">` • Server-driven `PageEvent` vs client `MatTableDataSource` • UI, Accessibility & Edge-Case Pitfall Guard (Client and Server-Side Pagination with `MatPaginator`) • Hands-On Frontend Implementation & Verification (Client and Server-Side Pagination with `MatPaginator`) • Client and Server-Side Pagination with `MatPaginator` — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Client

Purpose: Driving server-paginated REST endpoints (`?page=&limit=`) from `MatPaginator` `(page)` events while resetting `pageIndex = 0` whenever filters change!

What Topic Is Next → Next Topic 20.11: Typeahead Selection with `MatAutocomplete`

20.11 Typeahead Selection with `MatAutocomplete` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<input matInput [formControl]="assigneeCtrl" [matAutocomplete]="auto">` • `<mat-autocomplete #auto="matAutocomplete" [displayWith]="displayFn">` • Async RxJS filtered options • UI, Accessibility & Edge-Case Pitfall Guard (Typeahead Selection with `MatAutocomplete`) • Hands-On Frontend Implementation & Verification (Typeahead Selection with `MatAutocomplete`) • Typeahead Selection with `MatAutocomplete` — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Typeahead Selection with `MatAutocomplete`

Purpose: Selecting an Assignee or Product SKU from 5,000 server records via debounced `MatAutocomplete` with `[displayWith]` label formatting.

What Topic Is Next → Next Topic 20.12: Responsive Layouts with Angular CDK (`BreakpointObserver`)

20.12 Responsive Layouts with Angular CDK (`BreakpointObserver`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@angular/cdk/layout` `BreakpointObserver` • Matching `(max-width: 1023.98px)` into a Signal via `toSignal` • Angular CDK Overlay/Clipboard/DragDrop awareness • UI, Accessibility & Edge-Case Pitfall Guard (Responsive Layouts with Angular CDK (`BreakpointObserver`)) • Hands-On Frontend Implementation & Verification (Responsive Layouts with Angular CDK (`BreakpointObserver`)) • Responsive Layouts with Angular CDK (`BreakpointObserver`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Responsive Layouts with Angular CDK (`BreakpointObse...

Purpose: Adapting `MatSidenav` mode and table column visibility reactively via `BreakpointObserver` without manual `window.resize` listeners.

What Topic Is Next → Next Topic 20.13: Advanced Component Interaction (`viewChild`, `contentChildren` & Service Bus)

20.13 Advanced Component Interaction (`viewChild`, `contentChildren` & Service Bus) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Signal queries `viewChild.required(MatPaginator)` • `contentChildren()` • `@HostBinding` / `@HostListener` • Coordinating sibling drawers/tables cleanly • UI, Accessibility & Edge-Case Pitfall Guard (Advanced Component Interaction (`viewChild`, `contentChildren` & Service Bus)) • Hands-On Frontend Implementation & Verification (Advanced Component Interaction (`viewChild`, `contentChildren` & Service Bus)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Advanced Component Interaction (`viewChild`, `conten...

Purpose: Querying `readonly paginator = viewChild.required(MatPaginator)` with modern Signal queries instead of nullable `@ViewChild(...) !` assertions.

What Topic Is Next → Next Topic 20.14: Dynamic Component Rendering (`NgComponentOutlet` & `ViewContainerRef`)

20.14 Dynamic Component Rendering (`NgComponentOutlet` & `ViewContainerRef`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Declarative `*ngComponentOutlet="widget.component • Inputs: widget.inputs"` for configurable dashboard widgets vs imperative `ViewContainerRef.createComponent` • UI, Accessibility & Edge-Case Pitfall Guard (Dynamic Component Rendering (`NgComponentOutlet` & `ViewContainerRef`)) • Hands-On Frontend Implementation & Verification (Dynamic Component Rendering (`NgComponentOutlet` & `ViewContainerRef`)) • Dynamic Component Rendering (`NgComponentOutlet` & `ViewContainerRef`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Component Rendering (`NgComponentOutlet` & `... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Rendering user-customizable dashboard KPI/chart widgets dynamically from a role configuration array via `*ngComponentOutlet`!

What Topic Is Next → Next Topic 20.15: Angular Change Detection Internals (Zone.js vs Zoneless Signals)

20.15 Angular Change Detection Internals (Zone.js vs Zoneless Signals) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How `Default` change detection checks the whole component tree top-down after every async browser event • `ExpressionChangedAfterItHasBeenCheckedError` root cause & fix • Modern Signal-driven direction • UI, Accessibility & Edge-Case Pitfall Guard (Angular Change Detection Internals (Zone.js vs Zoneless Signals)) • Hands-On Frontend Implementation & Verification (Angular Change Detection Internals (Zone.js vs Zoneless Signals)) • Angular Change Detection Internals (Zone.js vs Zoneless Signals) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Angular Change Detection Internals (Zone.js vs Zonel...

Purpose: Fixing `NG0100: ExpressionChangedAfterItHasBeenCheckedError` cleanly by deriving state via `computed()` instead of mutating parent state inside `ngAfterViewInit`!

What Topic Is Next → Next Topic 20.16: Performance Optimization with `ChangeDetectionStrategy.OnPush`

20.16 Performance Optimization with `ChangeDetectionStrategy.OnPush` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When an `OnPush` component re-renders ONLY: 1) `@Input` reference changes • 2) component/child DOM event fires • 3) `| async` emits • Or 4) a read `signal()` updates! • UI, Accessibility & Edge-Case Pitfall Guard (Performance Optimization with `ChangeDetectionStrategy.OnPush`) • Hands-On Frontend Implementation & Verification (Performance Optimization with `ChangeDetectionStrategy.OnPush`) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Performance Optimization with `ChangeDetectionStrate...

Purpose: Adding `changeDetection: ChangeDetectionStrategy.OnPush` across all admin cards, rows, and pages so typing in a filter skips checking 95% of the tree!

What Topic Is Next → Next Topic 20.17: Lazy Loading Feature Areas and Deferrable Views (`@defer`)

20.17 Lazy Loading Feature Areas and Deferrable Views (`@defer`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Route-level `loadChildren` + Angular template `@defer (on viewport)` / `@placeholder` / `@loading` lazy-loading below-the-fold heavy analytics charts! • UI, Accessibility & Edge-Case Pitfall Guard (Lazy Loading Feature Areas and Deferrable Views (`@defer`)) • Hands-On Frontend Implementation & Verification

(Lazy Loading Feature Areas and Deferrable Views (`@defer`)) • Lazy Loading Feature Areas and Deferrable Views (`@defer`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Lazy Loading Feature Areas • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Wrapping a heavy Revenue Chart component in `@defer (on viewport)` with `@placeholder` so chart JS downloads ONLY when scrolled into view!

What Topic Is Next → Next Topic 20.18: Angular Component Unit Testing (`TestBed` and `ComponentFixture`)

20.18 Angular Component Unit Testing (`TestBed` and `ComponentFixture`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `TestBed.configureTestingModule({ imports: [StandaloneComp] })` • `fixture.componentRef.setInput(...)` • `fixture.detectChanges()` • Querying accessible DOM & verifying emitted `@Output`s • UI, Accessibility & Edge-Case Pitfall Guard (Angular Component Unit Testing (`TestBed` and `ComponentFixture`)) • Hands-On Frontend Implementation & Verification (Angular Component Unit Testing (`TestBed` and `ComponentFixture`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Angular Component Unit Testing (`TestBed` and `Compo...

Purpose: Writing fast, isolated unit tests for Standalone Components, Reactive Form validation rules, and custom Pipes.

What Topic Is Next → Next Topic 20.19: Testing Angular HTTP Services & Interceptors (`provideHttpClientTesting`)

20.19 Testing Angular HTTP Services & Interceptors (`provideHttpClientTesting`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `provideHttpClient()` + `provideHttpClientTesting()` • `inject(HttpTestingController)` • `httpMock.expectOne('/api/users')` • Asserting HTTP method/headers (`Authorization`) • UI, Accessibility & Edge-Case Pitfall Guard (Testing Angular HTTP Services & Interceptors (`provideHttpClientTesting`)) • Hands-On Frontend Implementation & Verification (Testing Angular HTTP Services & Interceptors (`provideHttpClientTesting`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Testing Angular HTTP Services

Purpose: Proving `EmployeeApiService` sends exact query params and `authInterceptor` attaches `Bearer <token>` without hitting any live backend!

What Topic Is Next → Next Topic 20.20: Angular Production Build Optimization & Bundle Budgets

20.20 Angular Production Build Optimization & Bundle Budgets [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ng build --configuration production` (esbuild + AOT template compilation + minification + dead-code elimination) • `angular.json` `budgets` (`initial` warning/error KB thresholds) • UI, Accessibility & Edge-Case Pitfall Guard (Angular Production Build Optimization & Bundle Budgets) • Hands-On Frontend Implementation & Verification (Angular Production Build Optimization & Bundle Budgets) • Angular Production Build Optimization & Bundle Budgets — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Angular Production Build Optimization • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Enforcing `budgets` in `angular.json` so CI immediately flags any PR that accidentally balloons the initial JS bundle!

What Topic Is Next → Next Topic 20.21: Deploying Angular SPAs (Vercel / Netlify Rewrite Rules)

20.21 Deploying Angular SPAs (Vercel / Netlify Rewrite Rules) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Output directory `dist/<project>/browser` • Configuring SPA fallback rewrite `/* -> /index.html (200)` so refreshing `/admin/users` deep links never returns host `404`! • UI, Accessibility & Edge-Case Pitfall Guard (Deploying Angular SPAs (Vercel / Netlify Rewrite Rules)) • Hands-On Frontend Implementation & Verification (Deploying Angular SPAs (Vercel / Netlify Rewrite Rules)) • Deploying Angular SPAs (Vercel / Netlify Rewrite Rules) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Deploying Angular SPAs (Vercel / Netlify Rewrite Rul... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Adding `vercel.json` / `public/_redirects` SPA rewrite rules so every client-side Angular route works on direct browser refresh.

What Topic Is Next → Next Topic 20.22: Major Project 5 Integration: Enterprise Admin Dashboard in Angular

20.22 Major Project 5 Integration: Enterprise Admin Dashboard in Angular [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Delivering complete Angular Admin Dashboard (Auth + Guards + Interceptors • KPI Overview • Users/Products/Orders `MatTable` CRUD + Search/Filter/Pagination • Reactive Forms • UI, Accessibility & Edge-Case Pitfall Guard (Major Project 5 Integration: Enterprise Admin Dashboard in Angular) • Hands-On Frontend Implementation & Verification (Major Project 5 Integration: Enterprise Admin Dashboard in Angular) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Major Project 5 Integration

Purpose: Demonstrating side-by-side proficiency in both React (Project 4) and Enterprise Angular (Project 5) in your GitHub portfolio!

What Topic Is Next → Module 20 Capstone (Project 5: Enterprise Admin Dashboard using Angular & Angular Material) → Next Topic 21.1: Purpose of Distributed Version Control Systems (DVCS)

Module 20 Hands-On Capstone Project (100 Marks): Project 5: Enterprise Admin Dashboard using Angular & Angular Material — Develop a modular, role-protected Enterprise Back-Office Admin Dashboard using modern Standalone Angular, TypeScript 5, Signals, RxJS, Reactive Forms (`FormGroup` & `FormArray`), Functional...

MODULE 21 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 22 TOPICS (176 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 21: Git, GitHub and Frontend Team Collaboration

Consolidate professional daily Git & GitHub engineering habits across every frontend repository: branching strategies, conflict resolution, safe rebasing, `stash`, `reset` vs `revert`, Pull Request code reviews, Conventional Commits, `.gitignore` hygiene, portfolio `README.md` standards, and Agile sprint collaboration.

Objectives: ✓ Work fluently from the terminal with `git init`, `status`, `diff`, `add -p`, `commit`, `log --oneline --graph`, `branch`, `switch`, `merge`, `rebase`, `stash`, `reset`, and `revert`. • ✓ Resolve multi-developer HTML/CSS/TypeScript/package-lock merge conflicts calmly without losing teammate changes. • ✓ Execute professional GitHub Pull Request workflows: small scoped feature branches, PR descriptions with screenshots, code review checklists, and branch protection. • ✓ Maintain spotless `.gitignore` files, Conventional Commits, recruiter-grade `README.md` documentation, and Agile Jira/Linear ticket estimation & standups.

21.1 Purpose of Distributed Version Control Systems (DVCS) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why Git snapshots content-addressed trees (SHA-1/SHA-256) • Working Directory vs Staging Index vs Local `.git` Repository vs Remote `origin` • UI, Accessibility & Edge-Case Pitfall Guard (Purpose of Distributed Version Control Systems (DVCS)) • Hands-On Frontend Implementation & Verification (Purpose of Distributed Version Control Systems (DVCS)) • Purpose of Distributed Version Control Systems (DVCS) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Purpose of Distributed Version Control Systems (DVCS... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Tracing exact regression commits (`git blame` / `git bisect`) and collaborating across distributed frontend/backend squads.

What Topic Is Next → Next Topic 21.2: Git Installation, Identity, SSH Keys and `core.autocrlf`

21.2 Git Installation, Identity, SSH Keys and `core.autocrlf` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring `user.name` • `user.email` • Ed25519 SSH keys (`ssh-keygen -t ed25519`) • Windows/Linux line-ending safety (`core.autocrlf` • UI, Accessibility & Edge-Case Pitfall Guard (Git Installation, Identity, SSH Keys and `core.autocrlf`) • Hands-On Frontend Implementation & Verification (Git Installation, Identity, SSH Keys and `core.autocrlf`) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Git Installation

Purpose: Preventing noisy 500-line CRLF-vs-LF diffs when Windows and macOS/Linux teammates collaborate on the same React/Angular repo!

What Topic Is Next → Next Topic 21.3: Local and Remote Repository Lifecycle

21.3 Local and Remote Repository Lifecycle [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Initializing local repos vs cloning existing team monorepos • `origin` vs `upstream` • Tracking branches (`origin/main`) • UI, Accessibility & Edge-Case Pitfall Guard (Local and Remote Repository Lifecycle) • Hands-On Frontend Implementation & Verification (Local and Remote Repository Lifecycle) • Local and Remote Repository Lifecycle — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Local

Purpose: Understanding why `origin/main` is your local cached pointer updated by `git fetch` until merged/rebased.

What Topic Is Next → Next Topic 21.4: Initializing Repositories cleanly (`git init`)

21.4 Initializing Repositories cleanly (`git init`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git init -b main` • Creating `.gitignore` • `.env.example` BEFORE the very first `git add` so `node_modules` or `.env` never touch history • UI, Accessibility & Edge-Case Pitfall Guard (Initializing Repositories cleanly (`git init`)) • Hands-On Frontend Implementation & Verification (Initializing Repositories cleanly (`git init`)) • Initializing Repositories cleanly (`git init`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Initializing Repositories cleanly (`git init`)

Purpose: Starting greenfield Vite/Angular/Next.js repos cleanly on `main` with zero tracked build artifacts.

What Topic Is Next → Next Topic 21.5: Selective Staging (`git add`, `git add -p` & `git restore --staged`)

21.5 Selective Staging (`git add`, `git add -p` & `git restore --staged`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reviewing `git diff` before staging • Hunk-level interactive staging `git add -p` to split unrelated changes into atomic commits • Unstaging mistakes safely • UI, Accessibility & Edge-Case Pitfall Guard (Selective Staging (`git add`, `git add -p` & `git restore --staged`)) • Hands-On Frontend Implementation & Verification (Selective Staging (`git add`, `git add -p` & `git restore --staged`)) • Selective Staging (`git add`, `git add -p` & `git restore --staged`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Selective Staging (`git add`, `git add -p` & `git re...

Purpose: Using `git add -p` to leave temporary `console.log` debug lines out of a clean bug-fix commit!

What Topic Is Next → Next Topic 21.6: Atomic Commits (`git commit`) and Commit Hygiene

21.6 Atomic Commits (`git commit`) and Commit Hygiene [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: One logical change per commit • Imperative subject (`feat(cart): apply coupon discount`) • Explaining *why* in the commit body • `--amend` on unpushed local commits only • UI, Accessibility & Edge-Case Pitfall Guard (Atomic Commits (`git commit`) and Commit Hygiene) • Hands-On Frontend Implementation & Verification (Atomic Commits (`git commit`) and Commit Hygiene) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Atomic Commits (`git commit`)

Purpose: Making code reviews 3x faster and rollbacks surgical when each commit compiles and passes tests independently.

What Topic Is Next → Next Topic 21.7: Inspecting History (`git log`, `git show`, `git diff`, `git blame`)

21.7 Inspecting History (`git log`, `git show`, `git diff`, `git blame`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git log --oneline --graph --decorate` • Filtering by file `git log -p -- src/cartSlice.ts` • `git show <sha>` • `git blame` context lookup • UI, Accessibility & Edge-Case Pitfall Guard (Inspecting History (`git log`, `git show`, `git diff`, `git blame`)) • Hands-On Frontend Implementation & Verification (Inspecting History (`git log`, `git show`, `git diff`, `git blame`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Inspecting History (`git log`, `git show`, `git diff...

Purpose: Checking `git log -p -- src/services/api.ts` to understand why a teammate added a custom timeout header 3 months ago before deleting it.

What Topic Is Next → Next Topic 21.8: Branching (`git branch`, `git switch -c`)

21.8 Branching (`git branch`, `git switch -c`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lightweight pointer mechanics of Git branches • Naming conventions (`feat/wishlist-drawer` • `fix/safari-modal-scroll` • `chore/ci-pipeline`) • UI, Accessibility & Edge-Case Pitfall Guard (Branching (`git branch`, `git switch -c`)) • Hands-On Frontend Implementation & Verification (Branching (`git branch`, `git switch -c`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Branching (`git branch`, `git switch -c`)

Purpose: Creating short-lived (< 2 day) feature branches off fresh `main` for every ticket.

What Topic Is Next → Next Topic 21.9: Merging and Resolving Merge Conflicts Calmly

21.9 Merging and Resolving Merge Conflicts Calmly [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fast-forward vs 3-way merge commits • Reading `<<<<<<< HEAD` • `=======` • `>>>>>>> feat/...` conflict markers • UI, Accessibility & Edge-Case Pitfall Guard (Merging and Resolving Merge Conflicts Calmly) • Hands-On Frontend Implementation & Verification (Merging and Resolving Merge Conflicts Calmly) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Merging

Purpose: Resolving conflicting edits in `AppRoutes.tsx` and `package.json` cleanly, then running `npm test && npm run build` before `git commit`.

What Topic Is Next → Next Topic 21.10: Rebasing Fundamentals (`git rebase main`) and Golden Rule

21.10 Rebasing Fundamentals (`git rebase main`) and Golden Rule [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Replaying your local feature commits linearly on top of latest `origin/main` • `git rebase --continue` / `--abort` • Golden Rule: NEVER rebase shared public `main`! • UI, Accessibility & Edge-Case Pitfall Guard (Rebasing Fundamentals (`git rebase main`) and Golden Rule) • Hands-On Frontend Implementation & Verification (Rebasing Fundamentals (`git rebase main`) and Golden Rule) • Rebasing Fundamentals (`git rebase main`) and Golden Rule — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Rebasing Fundamentals (`git rebase main`)

Purpose: Updating your local `feat/search` branch onto latest `origin/main` before opening a Pull Request for a crisp, linear diff.

What Topic Is Next → Next Topic 21.11: Saving Work-in-Progress with `git stash`

21.11 Saving Work-in-Progress with `git stash` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git stash push -u -m "wip: checkout address form"` • `git stash list` • `git stash pop` vs `apply` • `git stash drop` • UI, Accessibility & Edge-Case Pitfall Guard (Saving Work-in-Progress with `git stash`) • Hands-On Frontend Implementation & Verification (Saving Work-in-Progress with `git stash`) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Saving Work-in-Progress with `git stash`

Purpose: Pausing unfinished feature edits (`-u` including untracked new files!) for 10 minutes to fix an urgent production P0 hotfix on `main`.

What Topic Is Next → Next Topic 21.12: Undoing Mistakes Safely (`git restore`, `git reset` vs `git revert` & `reflog`)

21.12 Undoing Mistakes Safely (`git restore`, `git reset` vs `git revert` & `reflog`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git restore <file>` • `--soft` vs `--mixed` vs `--hard` `git reset` (local unpushed only!) • Safe public rollback commit `git revert <sha>` • Emergency `git reflog` recovery • UI, Accessibility & Edge-Case Pitfall Guard (Undoing Mistakes Safely (`git restore`, `git reset` vs `git revert` & `reflog`)) • Hands-On Frontend Implementation & Verification (Undoing Mistakes Safely (`git restore`, `git reset` vs `git revert` & `reflog`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Undoing Mistakes Safely (`git restore`, `git reset`...

Purpose: Reverting a broken merged PR on `main` via `git revert <merge-sha>` (creates a safe new inverse commit without rewriting shared history!).

What Topic Is Next → Next Topic 21.13: Managing Remotes (`git remote`)

21.13 Managing Remotes (`git remote`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Adding/updating SSH/HTTPS remotes (`git remote add origin` • `set-url`) • Fork workflow (`origin` fork + `upstream` main repo) • UI, Accessibility & Edge-Case Pitfall Guard (Managing Remotes (`git remote`)) • Hands-On Frontend Implementation & Verification (Managing Remotes (`git remote`)) • Managing Remotes (`git remote`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Managing Remotes (`git remote`)

Purpose: Syncing an open-source or organization fork cleanly from `upstream/main`.

What Topic Is Next → Next Topic 21.14: Pushing & Pulling Safely (`git push --force-with-lease`, `git pull --ff-only`)

21.14 Pushing & Pulling Safely (`git push --force-with-lease`, `git pull --ff-only`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Publishing new branches `git push -u origin feat/...` • Why `--force-with-lease` protects against overwriting collaborator commits vs dangerous `--force` • Avoiding messy accidental merge bubbles on pull • UI, Accessibility & Edge-Case Pitfall Guard (Pushing & Pulling Safely (`git push --force-with-lease`, `git pull --ff-only`)) • Hands-On Frontend Implementation & Verification (Pushing & Pulling Safely (`git push --force-with-lease`, `git pull --ff-only`)) • Pushing & Pulling Safely (`git push --force-with-lease`, `git pull --ff-only`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pushing

Purpose: Using `--force-with-lease` (never bare `-f`) when updating your own rebased PR branch.

What Topic Is Next → Next Topic 21.15: Cloning (`git clone`) and Fetching (`git fetch --prune`)

21.15 Cloning (`git clone`) and Fetching (`git fetch --prune`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git fetch --prune` updating remote refs • Cleaning deleted remote branch pointers safely before inspecting `git log HEAD..origin/main` • UI, Accessibility & Edge-Case Pitfall Guard (Cloning (`git clone`) and Fetching (`git fetch --prune`)) • Hands-On Frontend Implementation & Verification (Cloning (`git clone`) and Fetching (`git fetch --prune`)) • Cloning (`git clone`) and Fetching (`git fetch --prune`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cloning (`git clone`) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Fetching + pruning stale remote branches weekly so `git branch -a` stays clean and accurate.

What Topic Is Next → Next Topic 21.16: GitHub Repository Configuration & Branch Protection

21.16 GitHub Repository Configuration & Branch Protection [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Protecting `main` (require PR review + require passing GitHub Actions status checks `build` & `test` + block force pushes) • Issues • Labels • GitHub Releases • UI, Accessibility & Edge-Case Pitfall Guard (GitHub Repository Configuration & Branch Protection) • Hands-On Frontend Implementation & Verification (GitHub Repository Configuration & Branch Protection) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for GitHub Repository Configuration

Purpose: Ensuring zero broken TypeScript builds or failing Vitest tests can ever merge into `main`!

What Topic Is Next → Next Topic 21.17: Authoring High-Signal Pull Requests (PRs)

21.17 Authoring High-Signal Pull Requests (PRs) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Keeping PRs < 300–400 lines • Structured PR template (Problem/Ticket • Solution Summary • Desktop + Mobile Screenshots/GIF • UI, Accessibility & Edge-Case Pitfall Guard (Authoring High-Signal Pull Requests (PRs)) • Hands-On Frontend Implementation & Verification (Authoring High-Signal Pull Requests (PRs)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Authoring High-Signal Pull Requests (PRs)

Purpose: Attaching desktop + 375px mobile screenshots to every UI Pull Request so reviewers catch visual regressions in 30 seconds!

What Topic Is Next → Next Topic 21.18: Team Branching Strategies (GitHub Flow & Trunk-Based Development)

21.18 Team Branching Strategies (GitHub Flow & Trunk-Based Development) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: GitHub Flow (`main` always deployable + short-lived feature branches + PR + auto preview deploy) vs release branches (`release/v1.2`) in regulated enterprises • UI, Accessibility & Edge-Case Pitfall Guard (Team Branching Strategies (GitHub Flow & Trunk-Based Development)) • Hands-On Frontend Implementation & Verification (Team Branching Strategies (GitHub Flow & Trunk-Based Development)) • Team Branching Strategies (GitHub Flow & Trunk-Based Development) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Team Branching Strategies (GitHub Flow & Trunk-Based... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Using Vercel/Netlify automatic PR Preview URLs on every feature branch so QA and designers test live links before merge!

What Topic Is Next → Next Topic 21.19: Frontend Code Review Practices (Giving & Receiving Reviews)

21.19 Frontend Code Review Practices (Giving & Receiving Reviews) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Frontend PR checklist: strict TypeScript (no `any`!) • Keyboard/ARIA accessibility • Loading/empty/error states • XSS/secret safety • UI, Accessibility & Edge-Case Pitfall Guard (Frontend Code Review Practices (Giving & Receiving Reviews)) • Hands-On Frontend Implementation & Verification (Frontend Code Review Practices (Giving & Receiving Reviews)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Frontend Code Review Practices (Giving & Receiving R...

Purpose: Catching missing `AbortController`/stale closures, `index` keys, or unescaped URLs during peer PR review before customers ever see them.

What Topic Is Next → Next Topic 21.20: Frontend `.gitignore` Hygiene (`node_modules`, `dist`, `.angular`, `.next`, `.env*`)

21.20 Frontend `.gitignore` Hygiene (`node_modules`, `dist`, `.angular`, `.next`, `.env*`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Ignoring `node_modules/` • `dist/` • `.angular/cache` • `.next/` • UI, Accessibility & Edge-Case Pitfall Guard (Frontend `.gitignore` Hygiene (`node_modules`, `dist`, `.angular`, `.next`, `.env*`)) • Hands-On Frontend Implementation & Verification (Frontend `.gitignore` Hygiene (`node_modules`, `dist`, `.angular`, `.next`, `.env*`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Frontend `.gitignore` Hygiene (`node_modules`, `dist...

Purpose: Committing `package-lock.json` so `npm ci` installs identical dependency versions across every developer laptop and CI runner!

What Topic Is Next → Next Topic 21.21: Writing Recruiter-Grade Project `README.md` Documentation

21.21 Writing Recruiter-Grade Project `README.md` Documentation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Live Demo URL + Demo Credentials at very top • Problem Statement • Architecture & State diagram • Tech Stack badges • UI, Accessibility & Edge-Case Pitfall Guard (Writing Recruiter-Grade Project `README.md` Documentation) • Hands-On Frontend Implementation & Verification (Writing Recruiter-Grade Project `README.md` Documentation) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Writing Recruiter-Grade Project `README.md` Document...

Purpose: Turning your GitHub repositories into compelling engineering portfolios that hiring managers can evaluate in 60 seconds (instead of default Vite boilerplate README!).

What Topic Is Next → Next Topic 21.22: Conventional Commits & Agile Frontend Collaboration (Sprints, Standups, DoD)

21.22 Conventional Commits & Agile Frontend Collaboration (Sprints, Standups, DoD) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Conventional Commits (`feat` • `fix` • `perf` • `refactor` • UI, Accessibility & Edge-Case Pitfall Guard (Conventional Commits & Agile Frontend Collaboration (Sprints, Standups, DoD)) • Hands-On Frontend Implementation & Verification (Conventional Commits & Agile Frontend Collaboration (Sprints, Standups, DoD)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Conventional Commits

Purpose: Estimating frontend user stories realistically by accounting for loading/empty/error states, mobile breakpoints, accessibility, and tests — not just happy-path CSS!

What Topic Is Next → Module 21 Capstone (Week 21 Lab Build: Multi-Branch PR Collaboration, Conflict Resolution & Portfolio Audit) → Next Topic 22.1: Frontend Testing Strategy (The Testing Trophy)

Module 21 Hands-On Capstone Project (100 Marks): Week 21 Lab Build: Multi-Branch PR Collaboration, Conflict Resolution & Portfolio Audit — Execute a full GitHub Flow sprint (feature branches, interactive staging, rebase/merge conflict resolution, peer PR code review, `.gitignore` audit, and recruiter-grade `README.md` upgrades...

MODULE 22 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 23 TOPICS (184 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 22: Frontend Testing, Accessibility, SEO and Deployment

Take frontend applications to production standard: full Testing Trophy verification (Vitest/RTL, Angular `TestBed`, Playwright E2E critical user journeys, MSW API mocking), WCAG 2.2 AA accessibility (semantic HTML, ARIA, keyboard focus, screen readers, contrast), Technical SEO (OpenGraph, canonicals, JSON-LD), Vite production bundling, Vercel/Netlify deployment with custom domains/HTTPS, GitHub Actions CI/CD, and Sentry error monitoring.

Objectives: ✓ Automate unit, component (React RTL + Angular `TestBed`), API-mocked integration (`MSW`), and critical browser E2E (`Playwright`) tests. • ✓ Audit and remediate WCAG 2.2 AA accessibility across semantic landmarks, ARIA roles/live regions, keyboard focus traps, screen-reader labels, and 4.5:1 color contrast. • ✓ Implement Technical SEO (`<title>`, `meta description`, canonicals, OpenGraph, `robots.txt`, `sitemap.xml`, `JSON-LD` Product schema, crawlable `<a href>` links) and Lighthouse tuning. • ✓ Configure Vite production builds, deploy live SPAs/SSR apps to Vercel & Netlify with HTTPS/env vars/SPA rewrites, wire GitHub Actions CI (`lint -> typecheck -> test -> build`), and understand Sentry browser error monitoring.

22.1 Frontend Testing Strategy (The Testing Trophy) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Balancing ESLint + Strict TypeScript (static base) -> Pure Unit Tests (utilities/reducers) -> Component Integration Tests (70% sweet spot!) -> Focused E2E Critical Flows (Auth + Checkout) • UI, Accessibility & Edge-Case Pitfall Guard (Frontend Testing Strategy (The Testing Trophy)) • Hands-On Frontend Implementation & Verification (Frontend Testing Strategy (The Testing Trophy)) • Frontend Testing Strategy (The Testing Trophy) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Frontend Testing Strategy (The Testing Trophy) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Getting maximum bug-catching confidence per minute of CI runtime without maintaining 200 brittle E2E tests.

What Topic Is Next → Next Topic 22.2: UI Component Testing Best Practices (Behavior over Implementation)

22.2 UI Component Testing Best Practices (Behavior over Implementation) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Testing all 4 UI states (Loading skeleton • Empty state • Error retry • Populated interaction) • UI, Accessibility & Edge-Case Pitfall Guard (UI Component Testing Best Practices (Behavior over Implementation)) • Hands-On Frontend Implementation & Verification (UI Component Testing Best Practices (Behavior over Implementation)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for UI Component Testing Best Practices (Behavior over I...

Purpose: Creating a single `renderWithProviders(<CartDrawer />, { preloadedState })` test utility used across every feature test file.

What Topic Is Next → Next Topic 22.3: React Component Testing Deep Dive (Vitest + RTL)

22.3 React Component Testing Deep Dive (Vitest + RTL) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Testing multi-step wizards • Modal focus & close • Debounced search filtering • Redux cart interactions • UI, Accessibility & Edge-Case Pitfall Guard (React Component Testing Deep Dive (Vitest + RTL)) • Hands-On Frontend Implementation & Verification (React Component Testing Deep Dive (Vitest + RTL)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Component Testing Deep Dive (Vitest + RTL)

Purpose: Asserting that adding 2 items, applying a valid coupon, and removing 1 item updates the rendered grand total accurately.

What Topic Is Next → Next Topic 22.4: Angular Component & Reactive Form Testing (`TestBed`)

22.4 Angular Component & Reactive Form Testing (`TestBed`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Testing Standalone Angular components with `MatTable` • Reactive Form invalid -> valid submit emission • Guard functional tests (`TestBed.runInInjectionContext`) • UI, Accessibility & Edge-Case Pitfall Guard (Angular Component & Reactive Form Testing (`TestBed`)) • Hands-On Frontend Implementation & Verification (Angular Component & Reactive Form Testing (`TestBed`)) • Angular Component & Reactive Form Testing (`TestBed`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Angular Component

Purpose: Testing `authGuard` and `roleGuard` in 8 lines via `TestBed.runInInjectionContext(() => authGuard(route, state))`!

What Topic Is Next → Next Topic 22.5: End-to-End Browser Testing with Playwright (`@playwright/test`)

22.5 End-to-End Browser Testing with Playwright (`@playwright/test`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `playwright.config.ts` (`webServer` auto-start) • Resilient locators (`page.getByRole` • `page.getByLabel`) • Auto-waiting `await expect(locator).toBeVisible()` • UI, Accessibility & Edge-Case Pitfall Guard (End-to-End Browser Testing with Playwright (`@playwright/test`)) • Hands-On Frontend Implementation & Verification (End-to-End Browser Testing with Playwright (`@playwright/test`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for End-to-End Browser Testing with Playwright (`@playwr...

Purpose: Automating the full revenue-critical user journey in real Chromium/WebKit browsers: Browse -> Search -> Add to Cart -> Login -> Checkout -> Order Confirmation!

What Topic Is Next → Next Topic 22.6: Realistic Network Mocking (`MSW` & Playwright `page.route`)

22.6 Realistic Network Mocking (`MSW` & Playwright `page.route`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Intercepting REST calls deterministically with Mock Service Worker (`msw/node` in Vitest) • `await page.route('**/api/orders' • ...)` in Playwright to test `500` errors & slow 3G latency • UI, Accessibility & Edge-Case Pitfall Guard (Realistic Network Mocking (`MSW` & Playwright `page.route`)) • Hands-On Frontend Implementation & Verification (Realistic Network Mocking (`MSW` & Playwright `page.route`)) • Realistic Network Mocking (`MSW` & Playwright

`page.route`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Realistic Network Mocking (`MSW` & Playwright `page....

Purpose: Simulating a `503 Payment Gateway Timeout` in Playwright to prove your checkout page displays a safe retry alert without clearing the user's address form!

What Topic Is Next → Next Topic 22.7: Web Accessibility (WCAG 2.2 Level AA) POUR Principles

22.7 Web Accessibility (WCAG 2.2 Level AA) POUR Principles [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Perceivable • Operable • Understandable • Robust (POUR) • UI, Accessibility & Edge-Case Pitfall Guard (Web Accessibility (WCAG 2.2 Level AA) POUR Principles) • Hands-On Frontend Implementation & Verification (Web Accessibility (WCAG 2.2 Level AA) POUR Principles) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Web Accessibility (WCAG 2.2 Level AA) POUR Principle...

Purpose: Catching ~50% of violations automatically with `@axe-core/playwright` / Lighthouse and the remaining 50% with a 5-minute keyboard + screen-reader walkthrough.

What Topic Is Next → Next Topic 22.8: Semantic HTML as the First Rule of Accessibility

22.8 Semantic HTML as the First Rule of Accessibility [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Native `<button>` • `<a href>` • `<input>` • `<select>` • UI, Accessibility & Edge-Case Pitfall Guard (Semantic HTML as the First Rule of Accessibility) • Hands-On Frontend Implementation & Verification (Semantic HTML as the First Rule of Accessibility) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Semantic HTML as the First Rule of Accessibility

Purpose: Replacing `<div class="card-btn" onClick={...}>` with `<button type="button">` to get `Tab`, `Enter`, `Space`, and `disabled` semantics for free!

What Topic Is Next → Next Topic 22.9: WAI-ARIA Roles, Labels, Descriptions and Live Regions

22.9 WAI-ARIA Roles, Labels, Descriptions and Live Regions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Icon-only button `aria-label="Remove Wireless Mouse from cart"` • `aria-labelledby` • `aria-describedby` field errors • `aria-expanded` • UI, Accessibility & Edge-Case Pitfall Guard (WAI-ARIA Roles, Labels, Descriptions and Live Regions) • Hands-On Frontend Implementation & Verification (WAI-ARIA Roles, Labels, Descriptions and Live Regions) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for WAI-ARIA Roles

Purpose: Giving every icon-only close/trash/cart `<button>` a specific contextual `aria-label` and announcing "Added to cart (3 items)" via an `aria-live="polite"` region.

What Topic Is Next → Next Topic 22.10: Keyboard Navigation, Visible Focus Indicators & Modal Focus Traps

22.10 Keyboard Navigation, Visible Focus Indicators & Modal Focus Traps [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Logical DOM `Tab` / `Shift+Tab` order (never positive `tabindex="3"`!) • `:focus-visible` high-contrast ring (`focus-visible:ring-2`) • Modal focus trapping & trigger focus restoration • Skip-to-content link • UI, Accessibility & Edge-Case Pitfall Guard (Keyboard Navigation, Visible Focus Indicators & Modal Focus Traps) • Hands-On Frontend Implementation & Verification (Keyboard Navigation, Visible Focus Indicators & Modal Focus Traps) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Keyboard Navigation

Purpose: Completing an entire product search -> filter -> cart drawer -> checkout flow using ONLY the keyboard without ever losing a visible focus ring!

What Topic Is Next → Next Topic 22.11: Screen Reader Testing Awareness (NVDA / VoiceOver & Accessibility Tree)

22.11 Screen Reader Testing Awareness (NVDA / VoiceOver & Accessibility Tree) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inspecting Chrome DevTools -> Elements -> Accessibility Tree (computed accessible name & role) • Testing landmark rotor navigation • Form error announcements in NVDA (Windows) / VoiceOver (macOS) • UI, Accessibility & Edge-Case Pitfall Guard (Screen Reader Testing Awareness (NVDA / VoiceOver & Accessibility Tree)) • Hands-On Frontend Implementation & Verification (Screen Reader Testing Awareness (NVDA / VoiceOver & Accessibility Tree)) • Screen Reader Testing Awareness (NVDA / VoiceOver & Accessibility Tree) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Screen Reader Testing Awareness (NVDA / VoiceOver &...

Purpose: Verifying in Chrome's Accessibility pane that every form control computes a clear Accessible Name + Error Description.

What Topic Is Next → Next Topic 22.12: Color Contrast Verification & Non-Color State Indicators

22.12 Color Contrast Verification & Non-Color State Indicators [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: WCAG AA contrast ratios (`>= 4.5:1` normal text • `>= 3:1` large text & UI borders/icons) • Pairing red/green status colors with icons & text labels for color-blind users • `prefers-reduced-motion` • UI, Accessibility & Edge-Case Pitfall Guard (Color Contrast Verification & Non-Color State Indicators) • Hands-On Frontend Implementation & Verification (Color Contrast Verification & Non-Color State Indicators) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Color Contrast Verification

Purpose: Using `text-slate-600` (`7:1` on white) instead of faint `text-slate-300` (`1.6:1` unreadable!) on muted subtitles and pairing error borders with warning icons + text.

What Topic Is Next → Next Topic 22.13: Multi-Viewport Responsive QA (`320px` to `1920px` & Touch Targets)

22.13 Multi-Viewport Responsive QA (`320px` to `1920px` & Touch Targets) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Testing `320px`/`360px` small Android • `390px` iPhone • `768px` iPad portrait • `1024px` laptop • UI, Accessibility & Edge-Case Pitfall Guard (Multi-Viewport Responsive QA (`320px` to `1920px` & Touch Targets)) • Hands-On Frontend Implementation & Verification (Multi-Viewport Responsive QA (`320px` to `1920px` & Touch Targets)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-

Ready Career Program Production Runtime Execution & Memory Lifecycle for Multi-Viewport Responsive QA (`320px` to `1920px` &...

Purpose: Ensuring mobile `<input className="text-base sm:text-sm">` uses `16px` on phones so iOS Safari doesn't jarringly zoom the viewport on input focus!

What Topic Is Next → Next Topic 22.14: Technical SEO Fundamentals for Frontend Engineers

22.14 Technical SEO Fundamentals for Frontend Engineers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Googlebot crawls • Renders JS • Indexes pages • Robots directives (`robots.txt` • UI, Accessibility & Edge-Case Pitfall Guard (Technical SEO Fundamentals for Frontend Engineers) • Hands-On Frontend Implementation & Verification (Technical SEO Fundamentals for Frontend Engineers) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Technical SEO Fundamentals for Frontend Engineers

Purpose: Keeping public `/`, `/products`, and `/products/:slug` indexable while marking `/cart`, `/checkout`, and `/admin` `noindex`.

What Topic Is Next → Next Topic 22.15: SEO Metadata, Canonical URLs, OpenGraph & JSON-LD Structured Data

22.15 SEO Metadata, Canonical URLs, OpenGraph & JSON-LD Structured Data [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unique `<title>` (50–60 chars) & `<meta name="description">` (140–158 chars) • `<link rel="canonical">` preventing `?utm_source=` duplicate indexing • OpenGraph/Twitter cards • Schema.org `Product` `application/ld+json` • UI, Accessibility & Edge-Case Pitfall Guard (SEO Metadata, Canonical URLs, OpenGraph & JSON-LD Structured Data) • Hands-On Frontend Implementation & Verification (SEO Metadata, Canonical URLs, OpenGraph & JSON-LD Structured Data) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SEO Metadata

Purpose: Adding `Product` + `AggregateRating` `JSON-LD` structured data on product pages for rich price/rating search snippets and clean WhatsApp/LinkedIn link previews.

What Topic Is Next → Next Topic 22.16: Crawlable Link Architecture & Heading Hierarchy

22.16 Crawlable Link Architecture & Heading Hierarchy [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why search crawlers follow real `<a href="/products/shoes">` links (NOT `<div onClick={() => navigate(...)}>`!) • Single `<h1>` per page with logical `<h2>` -> `<h3>` outline • Descriptive image `alt` • UI, Accessibility & Edge-Case Pitfall Guard (Crawlable Link Architecture & Heading Hierarchy) • Hands-On Frontend Implementation & Verification (Crawlable Link Architecture & Heading Hierarchy) • Crawlable Link Architecture & Heading Hierarchy — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Crawlable Link Architecture

Purpose: Using `<Link to="/products/shoes">` (renders `<a href>`) on every product card title and pagination page so crawlers AND middle-click "Open in new tab" users both work!

What Topic Is Next → Next Topic 22.17: Lighthouse & Core Web Vitals Production Tuning

22.17 Lighthouse & Core Web Vitals Production Tuning [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Eliminating render-blocking assets • Preconnecting API origins (`<link rel="preconnect" href="...">`) • Sizing images/skeletons for `CLS = 0` • Keeping initial JS lean • UI, Accessibility & Edge-Case Pitfall Guard (Lighthouse & Core Web Vitals Production Tuning) • Hands-On Frontend Implementation & Verification (Lighthouse & Core Web Vitals Production Tuning) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Lighthouse

Purpose: Adding `<link rel="preconnect" href="https://api.yourdomain.com">` to shave 150–300ms DNS+TLS handshake off the first API fetch!

What Topic Is Next → Next Topic 22.18: Frontend Production Build Pipeline (`tsc -b && vite build`)

22.18 Frontend Production Build Pipeline (`tsc -b && vite build`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Difference between Vite dev server (native ESM) vs Rollup production build (`tsc -b` strict typecheck -> tree-shaking -> minification -> CSS extraction -> content-hashed `/dist/assets/*`) • UI, Accessibility & Edge-Case Pitfall Guard (Frontend Production Build Pipeline (`tsc -b && vite build`)) • Hands-On Frontend Implementation & Verification (Frontend Production Build Pipeline (`tsc -b && vite build`)) • Frontend Production Build Pipeline (`tsc -b && vite build`) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Frontend Production Build Pipeline (`tsc -b && vite... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Always running `npm run build` locally before pushing because `vite` dev server transpiles TS with esbuild without failing on cross-file TypeScript type errors!

What Topic Is Next → Next Topic 22.19: Vite Build Configuration (`vite.config.ts`, Aliases, Chunks & Sourcemaps)

22.19 Vite Build Configuration (`vite.config.ts`, Aliases, Chunks & Sourcemaps) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Path alias `@/* -> ./src/*` (synced in `tsconfig` + `vite.config.ts`) • `build.sourcemap: 'hidden'` for Sentry without exposing source code publicly • Chunk size warnings & `manualChunks` • UI, Accessibility & Edge-Case Pitfall Guard (Vite Build Configuration (`vite.config.ts`, Aliases, Chunks & Sourcemaps)) • Hands-On Frontend Implementation & Verification (Vite Build Configuration (`vite.config.ts`, Aliases, Chunks & Sourcemaps)) • Vite Build Configuration (`vite.config.ts`, Aliases, Chunks & Sourcemaps) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Vite Build Configuration (`vite.config.ts`, Aliases,...

Purpose: Configuring `@/` path aliases cleanly so imports read `@/features/cart/cartSlice` instead of `../../../../cart/cartSlice`.

What Topic Is Next → Next Topic 22.20: Deploying Frontend Applications to Vercel & Netlify (with SPA Rewrites)

22.20 Deploying Frontend Applications to Vercel & Netlify (with SPA Rewrites) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Connecting GitHub repo for automatic `main` production deploys + PR Preview deployments • Configuring `vercel.json` (`rewrites: [{ "source": "/(.*)" • "destination": "/index.html" }]`) & `public/_redirects` • UI, Accessibility & Edge-Case Pitfall Guard (Deploying Frontend Applications to Vercel & Netlify (with SPA Rewrites)) • Hands-On Frontend Implementation & Verification (Deploying Frontend Applications to Vercel & Netlify (with SPA Rewrites)) • Deploying Frontend

Applications to Vercel & Netlify (with SPA Rewrites) — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Deploying Frontend Applications to Vercel

Purpose: Ensuring direct navigation or F5 refresh on `https://your-app.vercel.app/products/laptops` serves `index.html` (200) instead of a 404 page!

What Topic Is Next → Next Topic 22.21: Custom Domains, DNS (`A` / `CNAME`) and Automatic HTTPS/TLS

22.21 Custom Domains, DNS (`A` / `CNAME`) and Automatic HTTPS/TLS [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Apex domain `A` record vs subdomain `CNAME` (`shop.yourdomain.com`) • Let's Encrypt automated TLS provisioning • HTTP -> HTTPS redirect & HSTS • Updating backend CORS origin allowlist • UI, Accessibility & Edge-Case Pitfall Guard (Custom Domains, DNS (`A` / `CNAME`) and Automatic HTTPS/TLS) • Hands-On Frontend Implementation & Verification (Custom Domains, DNS (`A` / `CNAME`) and Automatic HTTPS/TLS) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Custom Domains

Purpose: Remembering to add your live `https://shop.yourdomain.com` origin to the backend API's CORS allowlist when moving off `localhost:5173`!

What Topic Is Next → Next Topic 22.22: Managing Production, Preview and Local Environment Variables

22.22 Managing Production, Preview and Local Environment Variables [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring separate `Development` • `Preview` (staging API) • `Production` environment variables in Vercel/Netlify dashboards • Why build-time env changes require triggering a new build • UI, Accessibility & Edge-Case Pitfall Guard (Managing Production, Preview and Local Environment Variables) • Hands-On Frontend Implementation & Verification (Managing Production, Preview and Local Environment Variables) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Managing Production

Purpose: Pointing PR Preview deploys at a safe Staging/Sandbox API while `main` Production deploys talk to the Production API.

What Topic Is Next → Next Topic 22.23: GitHub Actions CI/CD Pipeline & Production Error Monitoring (Sentry Basics)

22.23 GitHub Actions CI/CD Pipeline & Production Error Monitoring (Sentry Basics) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Authoring `.github/workflows/ci.yml` (`actions/checkout` • `actions/setup-node` with `cache: 'npm'` • `npm ci` • `npm run lint` • UI, Accessibility & Edge-Case Pitfall Guard (GitHub Actions CI/CD Pipeline & Production Error Monitoring (Sentry Basics)) • Hands-On Frontend Implementation & Verification (GitHub Actions CI/CD Pipeline & Production Error Monitoring (Sentry Basics)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for GitHub Actions CI/CD Pipeline

Purpose: Running automated lint + TypeScript + Vitest + production build verification on every Pull Request AND capturing uncaught production browser exceptions with stack traces!

What Topic Is Next → Module 22 Capstone (Week 22 Lab Build: Playwright E2E, WCAG 2.2 AA Audit & GitHub Actions Live CI/CD) → Next Topic 23.1: HTML5 Semantics, Forms & Accessibility Interview Deep Dive

Module 22 Hands-On Capstone Project (100 Marks): Week 22 Lab Build: Playwright E2E, WCAG 2.2 AA Audit & GitHub Actions Live CI/CD — Add Playwright E2E critical-flow tests, remediate WCAG keyboard/ARIA/contrast issues, add OpenGraph + JSON-LD SEO tags, wire `.github/workflows/ci.yml`, and deploy live to Vercel/Netlify wit...

MODULE 23 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 20 TOPICS (160 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 23: Frontend Coding Challenges and Technical Interview Preparation

Intensive placement preparation across all 20 Frontend Technical Interview domains, 60–100 JavaScript/TypeScript DSA coding problems, and timed 45–60 minute DOM & React Machine-Coding rounds (Autocomplete, Sortable Data Table, Pagination, Accessible Modal, Cart, Multi-Step Form, Infinite Scroll, Custom Dropdown, API Dashboard).

Objectives: ✓ Answer conceptual, output-prediction, debugging, and architecture questions crisply across HTML5, CSS3, JS internals, TypeScript, React, Redux/Query, Angular, RxJS, Security, Performance, Testing, and Git. • ✓ Solve timed JavaScript/TypeScript DSA coding questions (Arrays, Strings, Map/Set, Two Pointers, Sliding Window, Closures/Debounce/Promise utilities, Stack/Queue, Recursion) while explaining Big-O trade-offs. • ✓ Complete timed 60-minute Frontend Machine-Coding rounds from a blank screen with clean state design, keyboard accessibility, edge-case handling, and responsive CSS. • ✓ Present a recruiter-ready frontend developer resume, GitHub profile, and live portfolio walkthrough.

23.1 HTML5 Semantics, Forms & Accessibility Interview Deep Dive [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<!DOCTYPE>` • DOM/CSSOM/defer/async scripts • Semantic landmarks • `<button>` vs `<a>` • UI, Accessibility & Edge-Case Pitfall Guard (HTML5 Semantics, Forms & Accessibility Interview Deep Dive) • Hands-On Frontend Implementation & Verification (HTML5 Semantics, Forms & Accessibility Interview Deep Dive) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTML5 Semantics

Purpose: Explaining `<script defer>` vs `<script async>` execution order timelines with complete precision.

What Topic Is Next → Next Topic 23.2: CSS3 Cascade, Specificity, Stacking Context & Styling Architecture Viva

23.2 CSS3 Cascade, Specificity, Stacking Context & Styling Architecture Viva [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `box-sizing` • Specificity `(ID • Class • Type)` & `@layer` • UI, Accessibility & Edge-Case Pitfall Guard (CSS3 Cascade, Specificity, Stacking Context & Styling Architecture Viva) • Hands-On Frontend Implementation & Verification (CSS3 Cascade, Specificity, Stacking Context & Styling Architecture Viva) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSS3 Cascade

Purpose: Explaining why `z-index: 9999` inside a parent with `transform: translateY(0)` still renders behind a sibling header (`z-index: 10`)!

What Topic Is Next → Next Topic 23.3: Flexbox, CSS Grid & Responsive Layout Machine-Coding Drills

23.3 Flexbox, CSS Grid & Responsive Layout Machine-Coding Drills [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `flex: 1` (`1 1 0%`) vs `auto` • `min-width: 0` flex truncation fix • `repeat(auto-fit • Minmax(260px • UI, Accessibility & Edge-Case Pitfall Guard (Flexbox, CSS Grid & Responsive Layout Machine-Coding Drills) • Hands-On Frontend Implementation & Verification (Flexbox, CSS Grid & Responsive Layout Machine-Coding Drills) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Flexbox

Purpose: Machine-coding a responsive navbar + collapsible mobile drawer + fluid card grid in 20 minutes cleanly.

What Topic Is Next → Next Topic 23.4: JavaScript Execution Context, Hoisting, Closures & `this` Output Rounds

23.4 JavaScript Execution Context, Hoisting, Closures & `this` Output Rounds [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Creation vs Execution phase • TDZ (`let`/`const` vs `var`) • Lexical closures • `var` vs `let` loop timer output • UI, Accessibility & Edge-Case Pitfall Guard (JavaScript Execution Context, Hoisting, Closures & `this` Output Rounds) • Hands-On Frontend Implementation & Verification (JavaScript Execution Context, Hoisting, Closures & `this` Output Rounds) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JavaScript Execution Context

Purpose: Predicting exact console output on tricky closure/`this` snippets AND implementing `memoize(fn)` / `curry(fn)` live.

What Topic Is Next → Next Topic 23.5: JavaScript Event Loop, Microtasks, Prototypes & Memory Leaks Viva

23.5 JavaScript Event Loop, Microtasks, Prototypes & Memory Leaks Viva [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Call Stack -> Microtask Queue (`Promise.then` • `queueMicrotask`) -> Render -> Macrotask Queue (`setTimeout`) • Prototypal inheritance (`Object.create` • `class` sugar) • UI, Accessibility & Edge-Case Pitfall Guard (JavaScript Event Loop, Microtasks, Prototypes & Memory Leaks Viva) • Hands-On Frontend Implementation & Verification (JavaScript Event Loop, Microtasks, Prototypes & Memory Leaks Viva) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JavaScript Event Loop

Purpose: Walking line-by-line through mixed `sync + Promise.then + async/await + setTimeout(0)` execution order.

What Topic Is Next → Next Topic 23.6: Asynchronous JS & Custom Promise Combinator Coding Round

23.6 Asynchronous JS & Custom Promise Combinator Coding Round [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing `promiseAll` • `promiseAllSettled` • `promiseRace` • `promiseAny` • UI, Accessibility & Edge-Case Pitfall Guard (Asynchronous JS & Custom Promise Combinator Coding Round) • Hands-On Frontend Implementation & Verification (Asynchronous JS & Custom Promise Combinator Coding Round) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Asynchronous JS

Purpose: Writing `promiseAll` (preserving input index order!) and `retryWithBackoff` cleanly in frontend coding interviews.

What Topic Is Next → Next Topic 23.7: JavaScript Array & String DSA Problem-Solving Sprint

23.7 JavaScript Array & String DSA Problem-Solving Sprint [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Two Sum • Sliding Window Longest Unique Substring • Merge Intervals • Group Anagrams • UI, Accessibility & Edge-Case Pitfall Guard (JavaScript Array & String DSA Problem-Solving Sprint) • Hands-On Frontend Implementation & Verification (JavaScript Array & String DSA Problem-Solving Sprint) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JavaScript Array

Purpose: Implementing both classic Big-O array/string algorithms AND polyfills (`myReduce`, `flattenDeep`, `chunk`, `groupBy`) asked specifically in JS interviews!

What Topic Is Next → Next Topic 23.8: Deep Object Manipulation, Cloning & Equality Coding Round

23.8 Deep Object Manipulation, Cloning & Equality Coding Round [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing `deepClone` (handling nested arrays/Dates/circular `WeakMap` refs!) • `deepEqual(a • B)` • `get(obj • UI, Accessibility & Edge-Case Pitfall Guard (Deep Object Manipulation, Cloning & Equality Coding Round) • Hands-On Frontend Implementation & Verification (Deep Object Manipulation, Cloning & Equality Coding Round) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Deep Object Manipulation

Purpose: Solving `deepClone` with circular reference `WeakMap` protection and `deepEqual` — two of the top 10 frontend JS coding questions!

What Topic Is Next → Next Topic 23.9: DOM Manipulation, Event Delegation & Vanilla JS Machine Coding

23.9 DOM Manipulation, Event Delegation & Vanilla JS Machine Coding [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Capturing vs Target vs Bubbling • `e.target.closest('[data-action]')` • `DocumentFragment` • `debounce`/`throttle` • UI, Accessibility & Edge-Case Pitfall Guard (DOM Manipulation, Event Delegation & Vanilla JS Machine Coding) • Hands-On Frontend Implementation & Verification (DOM Manipulation, Event Delegation & Vanilla JS Machine Coding) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DOM Manipulation

Purpose: Acing Round-1 Vanilla JS DOM machine-coding interviews where React/Angular libraries are intentionally disallowed!

What Topic Is Next → Next Topic 23.10: TypeScript Strict Typing, Generics & Type Narrowing Viva

23.10 TypeScript Strict Typing, Generics & Type Narrowing Viva [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `unknown` vs `any` vs `never` • `interface` vs `type` • Discriminated unions • Custom type predicates (`x is User`) • UI, Accessibility & Edge-Case Pitfall Guard (TypeScript Strict Typing, Generics & Type Narrowing Viva) • Hands-On Frontend Implementation & Verification (TypeScript Strict Typing, Generics & Type Narrowing Viva) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TypeScript Strict Typing

Purpose: Typing a generic `ApiResponse<T>` discriminated union and `pick<T, K extends keyof T>(obj: T, keys: K[]): Pick<T, K>` live.

What Topic Is Next → Next Topic 23.11: React Hooks, Stale Closures & Rendering Internals Viva

23.11 React Hooks, Stale Closures & Rendering Internals Viva [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Rules of Hooks (fixed call-order array on Fiber node!) • Stale closures in `useEffect`/`setInterval` • `useEffect` vs `useLayoutEffect` • `useRef` • UI, Accessibility & Edge-Case Pitfall Guard (React Hooks, Stale Closures & Rendering Internals Viva) • Hands-On Frontend Implementation & Verification (React Hooks, Stale Closures & Rendering Internals Viva) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Hooks

Purpose: Fixing a broken stopwatch/polling hook suffering from stale `count` closure in 60 seconds during a live React debugging round!

What Topic Is Next → Next Topic 23.12: React State Architecture, Redux Toolkit & TanStack Query Viva

23.12 React State Architecture, Redux Toolkit & TanStack Query Viva [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Context vs Redux Toolkit vs Zustand vs TanStack Query trade-offs • Immer slices • `createSelector` memoization • Query keys • UI, Accessibility & Edge-Case Pitfall Guard (React State Architecture, Redux Toolkit & TanStack Query Viva) • Hands-On Frontend Implementation & Verification (React State Architecture, Redux Toolkit & TanStack Query Viva) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React State Architecture

Purpose: Designing the complete state architecture for a real-time food-delivery or e-commerce frontend in a 20-minute system design discussion.

What Topic Is Next → Next Topic 23.13: React Machine-Coding Round 1: Search Autocomplete, Sortable Data Table & Pagination

23.13 React Machine-Coding Round 1: Search Autocomplete, Sortable Data Table & Pagination [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Timed 60-min machine-coding execution: debounced API Autocomplete (keyboard Up/Down/Enter/Esc + caching + abort) • Sortable/Filterable Paginated Data Table • UI, Accessibility & Edge-Case Pitfall Guard (React Machine-Coding Round 1: Search Autocomplete, Sortable Data Table & Pagination) • Hands-On Frontend Implementation & Verification (React Machine-Coding Round 1: Search Autocomplete, Sortable Data Table & Pagination) • React Machine-Coding Round 1: Search Autocomplete, Sortable Data Table & Pagination — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Machine-Coding Round 1 • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Shipping working, accessible, typed React machine-coding solutions within a strict 60-minute interview window.

What Topic Is Next → Next Topic 23.14: React Machine-Coding Round 2: Accessible Modal, Multi-Step Form, Cart & Infinite Scroll

23.14 React Machine-Coding Round 2: Accessible Modal, Multi-Step Form, Cart & Infinite Scroll [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Timed 60-min machine-coding execution: Portal Accessible Modal (`Escape` + focus return) • Validated Multi-Step Wizard (persisted draft) • Shopping Cart • `IntersectionObserver` Infinite Scroll • UI, Accessibility & Edge-Case Pitfall Guard (React Machine-Coding Round 2: Accessible Modal, Multi-Step Form, Cart & Infinite Scroll) • Hands-On Frontend Implementation & Verification (React Machine-Coding Round 2: Accessible Modal, Multi-Step Form, Cart & Infinite Scroll) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for React Machine-Coding Round 2

Purpose: Handling state persistence across wizard Back/Next steps and per-step validation cleanly under interview pressure.

What Topic Is Next → Next Topic 23.15: Angular Architecture, Signals, DI & Change Detection Viva

23.15 Angular Architecture, Signals, DI & Change Detection Viva [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Standalone Components • Signals (`signal`/`computed`/`effect`) vs Zone.js • `OnPush` 4 trigger rules • Hierarchical DI (`root` vs route vs component) • UI, Accessibility & Edge-Case Pitfall Guard (Angular Architecture, Signals, DI & Change Detection Viva) • Hands-On Frontend Implementation & Verification (Angular Architecture, Signals, DI & Change Detection Viva) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Angular Architecture

Purpose: Answering enterprise Angular panel questions on `OnPush` change detection, Hierarchical Injectors, and Typed Reactive Forms.

What Topic Is Next → Next Topic 23.16: RxJS Operators, Subjects & Angular HTTP Interceptors Viva

23.16 RxJS Operators, Subjects & Angular HTTP Interceptors Viva [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cold vs Hot Observables • `Subject` vs `BehaviorSubject` vs `ReplaySubject` • `switchMap` vs `mergeMap` vs `concatMap` vs `exhaustMap` • `combineLatest` vs `forkJoin` • UI, Accessibility & Edge-Case Pitfall Guard (RxJS Operators, Subjects & Angular HTTP Interceptors Viva) • Hands-On Frontend Implementation & Verification (RxJS Operators, Subjects & Angular HTTP Interceptors Viva) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for RxJS Operators

Purpose: Drawing exact marble diagrams and choosing the right flattening/combination operator for any enterprise scenario.

What Topic Is Next → Next Topic 23.17: REST API Integration, Browser Storage & Frontend Security Viva

23.17 REST API Integration, Browser Storage & Frontend Security Viva [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Axios/HttpClient interceptors • Silent `401` token refresh queue (preventing 5 parallel refresh calls!) • `Cookie` (`HttpOnly`/`SameSite`) vs `localStorage` vs `sessionStorage` vs `IndexedDB` • XSS • UI, Accessibility & Edge-Case Pitfall Guard (REST API Integration, Browser Storage & Frontend Security Viva) • Hands-On Frontend Implementation & Verification (REST API Integration, Browser Storage & Frontend Security Viva) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for REST API Integration

Purpose: Explaining single-flight `refreshTokenPromise` locking when 4 parallel dashboard API requests all hit `401 Token Expired` simultaneously!

What Topic Is Next → Next Topic 23.18: Web Performance, Core Web Vitals & Frontend System Design Viva

23.18 Web Performance, Core Web Vitals & Frontend System Design Viva [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: LCP/INP/CLS optimization checklist • Code splitting • List virtualization • HTTP caching • UI, Accessibility & Edge-Case Pitfall Guard (Web Performance, Core Web Vitals & Frontend System Design Viva) • Hands-On Frontend Implementation & Verification (Web Performance, Core Web Vitals & Frontend System Design Viva) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Web Performance

Purpose: Structuring a 25-minute Frontend System Design answer using the **RADIO** framework (Requirements, Architecture, Data Model, Interface/API, Optimizations/A11y).

What Topic Is Next → Next Topic 23.19: Frontend Testing Strategy, Accessibility & SEO Interview Viva

23.19 Frontend Testing Strategy, Accessibility & SEO Interview Viva [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Vitest/RTL vs Playwright trade-offs • Accessible query priority (`getByRole`) • Mocking with MSW • WCAG keyboard/ARIA fixes • UI, Accessibility & Edge-Case Pitfall Guard (Frontend Testing Strategy, Accessibility & SEO Interview Viva) • Hands-On Frontend Implementation & Verification (Frontend Testing Strategy, Accessibility & SEO Interview Viva) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Frontend Testing Strategy

Purpose: Explaining how you tested your own Capstone project across unit, component, and Playwright E2E layers.

What Topic Is Next → Next Topic 23.20: Git Collaboration, Resume Polish & Portfolio Defense Walkthrough

23.20 Git Collaboration, Resume Polish & Portfolio Defense Walkthrough [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Rebase vs Merge • `reset` vs `revert` • Conflict resolution • CI/CD pipelines • UI, Accessibility & Edge-Case Pitfall Guard (Git Collaboration, Resume Polish & Portfolio Defense Walkthrough) • Hands-On Frontend Implementation & Verification (Git Collaboration, Resume Polish & Portfolio Defense Walkthrough) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Git Collaboration

Purpose: Delivering a crisp 5-minute live demo + code walkthrough of your React E-Commerce Capstone and Angular Admin Dashboard without fumbling.

What Topic Is Next → Module 23 Capstone (Week 23 Lab Build: 100-Challenge JS DSA + Timed Machine-Coding Interview Pack) → Next Topic 24.1: Capstone Architecture, Feature Blueprint & Design Tokens

Module 23 Hands-On Capstone Project (100 Marks): Week 23 Lab Build: 100-Challenge JS DSA + Timed Machine-Coding Interview Pack — Complete the curated JavaScript/TypeScript DSA repository (`Arrays` through `Promises`/`Closures`) plus timed 60-minute DOM & React Machine-Coding solutions (Autocomplete, Data Table, Modal,...

MODULE 24 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 15 TOPICS (120 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 24: Final Capstone Project and Placement Readiness

Architect, build, test, optimize, deploy, and defend Project 6 (Final Industry Capstone): a Complete Enterprise E-Commerce Frontend Application featuring Customer Storefront + Admin Management Portal using React, TypeScript, Redux Toolkit / TanStack Query, React Router, React Hook Form + Zod, Tailwind CSS, Vitest/RTL/Playwright, GitHub Actions CI, and live Vercel/Netlify HTTPS deployment.

Objectives: ✓ Deliver all 11 functional modules of an enterprise e-commerce frontend: Homepage, Auth, Product Listing, Product Details, Search Suggestions & Multi-Facet Filters, Cart, Wishlist, Multi-Step Checkout, Order History & Tracking, User Profile, and Protected Admin Product/Order Management. • ✓ Enforce production engineering standards across strict TypeScript, feature-based architecture, Redux Toolkit + Server Cache state, responsive `360px–1440px+` layouts, WCAG keyboard accessibility, and XSS/auth safety. • ✓ Verify critical user journeys with automated Vitest + React Testing Library + Playwright tests running inside GitHub Actions CI. • ✓ Deploy the live HTTPS application on Vercel or Netlify with SPA rewrites, custom SEO metadata, recruiter-grade `README.md`, and pass the Final Capstone Architecture & Code Defense Viva (>= 70%).

24.1 Capstone Architecture, Feature Blueprint & Design Tokens [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Setting up strict Vite + React + TypeScript + Tailwind tokens • `src/features/{catalog • Product • Cart • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Architecture, Feature Blueprint & Design Tokens) • Hands-On Frontend Implementation & Verification (Capstone Architecture, Feature Blueprint & Design Tokens) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Architecture

Purpose: Establishing clean feature boundaries, shared UI primitives, and route lazy-loading on Day 1 of the Final Capstone.

What Topic Is Next → Next Topic 24.2: Capstone Module 1: Conversion-Focused Homepage & Merchandising Sections

24.2 Capstone Module 1: Conversion-Focused Homepage & Merchandising Sections [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Eager LCP hero banner (`fetchpriority="high"`) • Featured category tiles • Flash-deal countdown/carousel • Trending products rail • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Module 1: Conversion-Focused Homepage & Merchandising Sections) • Hands-On Frontend Implementation & Verification (Capstone Module 1: Conversion-Focused Homepage & Merchandising Sections) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 1

Purpose: Delivering an instant, CLS-free storefront landing experience (`LCP < 2.5s`, `CLS < 0.05`) across mobile and desktop.

What Topic Is Next → Next Topic 24.3: Capstone Module 2: Authentication, Registration & Role Session Flow

24.3 Capstone Module 2: Authentication, Registration & Role Session Flow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Login & Registration with React Hook Form + Zod • Password visibility toggle • `returnUrl` safe post-login redirect • `CUSTOMER` vs `ADMIN` role guards • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Module 2: Authentication, Registration & Role Session Flow) • Hands-On Frontend Implementation & Verification (Capstone Module 2: Authentication, Registration & Role Session Flow) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 2

Purpose: Preserving guest cart items across login -> checkout redirect seamlessly without losing selected products!

What Topic Is Next → Next Topic 24.4: Capstone Module 3: Product Listing Page (Grid/List, Sort & URL Pagination)

24.4 Capstone Module 3: Product Listing Page (Grid/List, Sort & URL Pagination) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Responsive product card grid + compact list toggle • Sort (`Price Low–High` • `Price High–Low` • `Rating` • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Module 3: Product Listing Page (Grid/List, Sort & URL Pagination)) • Hands-On Frontend Implementation & Verification (Capstone Module 3: Product Listing Page (Grid/List, Sort & URL Pagination)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 3

Purpose: Making every filtered/sorted catalog URL bookmarkable and shareable via `useSearchParams`!

What Topic Is Next → Next Topic 24.5: Capstone Module 4: Product Details Page (Gallery, Variants, Specs & Reviews)

24.5 Capstone Module 4: Product Details Page (Gallery, Variants, Specs & Reviews) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Interactive thumbnail/zoom gallery • Color/size/storage variant selector updating price & stock availability • Sticky mobile Add-to-Cart bar • Specs/reviews accessible `<Tabs>` • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Module 4: Product Details Page (Gallery, Variants, Specs & Reviews)) • Hands-On Frontend Implementation & Verification (Capstone Module 4: Product Details Page (Gallery, Variants, Specs & Reviews)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 4

Purpose: Disabling "Add to Cart" cleanly with "Out of Stock" badge when a specific Size/Color SKU combination has `stock === 0`.

What Topic Is Next → Next Topic 24.6: Capstone Module 5: Instant Search Suggestions & Multi-Facet Filter Drawer

24.6 Capstone Module 5: Instant Search Suggestions & Multi-Facet Filter Drawer [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Debounced header search bar (`300ms` • `AbortController` • Keyboard Up/Down/Enter/Escape combobox • Recent searches) + desktop sidebar / mobile slide-over Filter Drawer (category • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Module 5: Instant Search Suggestions & Multi-Facet Filter Drawer) • Hands-On Frontend Implementation & Verification (Capstone Module 5: Instant Search Suggestions & Multi-Facet Filter Drawer) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 5

Purpose: Providing removable active-filter chips (`Electronics ×`, `4★ & above ×`, `Clear All`) above catalog results.

What Topic Is Next → Next Topic 24.7: Capstone Module 6: Shopping Cart Drawer & Full Cart Page (`cartSlice`)

24.7 Capstone Module 6: Shopping Cart Drawer & Full Cart Page (`cartSlice`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Redux Toolkit `cartSlice` + validated `localStorage` persistence • Slide-over quick Cart Drawer + `/cart` page • Quantity stepper (`1..maxStock`) • Coupon code (`SAVE10`/`FIRST500`) • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Module 6: Shopping Cart Drawer & Full Cart Page (`cartSlice`)) • Hands-On Frontend Implementation & Verification (Capstone Module 6: Shopping Cart Drawer & Full Cart Page (`cartSlice`)) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 6

Purpose: Deriving all pricing totals via memoized `createSelector` and guarding quantity steppers against stock limits.

What Topic Is Next → Next Topic 24.8: Capstone Module 7: Wishlist & Move-to-Cart Workflow

24.8 Capstone Module 7: Wishlist & Move-to-Cart Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Heart toggle with optimistic UI feedback on product cards & PDP • `/wishlist` saved items grid • One-click "Move to Cart" atomic action • Empty wishlist discovery CTA • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Module 7: Wishlist & Move-to-Cart Workflow) • Hands-On Frontend Implementation & Verification (Capstone Module 7: Wishlist & Move-to-Cart Workflow) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 7

Purpose: Letting shoppers save comparison items and move them straight into the active cart with a polite toast confirmation.

What Topic Is Next → Next Topic 24.9: Capstone Module 8: Multi-Step Validated Checkout Flow

24.9 Capstone Module 8: Multi-Step Validated Checkout Flow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 3-step Checkout (`1. Shipping Address` with PIN/Phone Zod validation -> `2. Delivery & Payment Method` UPI/Card/COD UI -> `3. Order Review & Place Order`) • Idempotent double-click prevention • Order Confirmation receipt • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Module 8: Multi-Step Validated Checkout Flow) • Hands-On Frontend Implementation & Verification (Capstone Module 8: Multi-Step Validated Checkout Flow) • Capstone Module 8: Multi-Step Validated Checkout Flow — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 8

Purpose: Redirecting empty carts away from `/checkout` automatically and preventing duplicate order submissions while `isPlacingOrder` is pending.

What Topic Is Next → Next Topic 24.10: Capstone Module 9: Order History, Invoice Summary & Status Timeline

24.10 Capstone Module 9: Order History, Invoice Summary & Status Timeline [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `/orders` filterable list + `/orders/:orderId` detail screen with visual stepper (`Placed -> Packed -> Shipped -> Out for Delivery -> Delivered`) • Itemized receipt • Printable/downloadable summary • Cancel Order modal (when `Placed`) • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Module 9: Order History, Invoice Summary & Status Timeline) • Hands-On Frontend Implementation & Verification (Capstone Module 9: Order History, Invoice Summary & Status Timeline) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 9

Purpose: Giving customers transparent post-purchase order status tracking, itemized GST breakdown, and safe cancellation before shipment.

What Topic Is Next → Next Topic 24.11: Capstone Module 10: Customer Profile & Saved Address Book

24.11 Capstone Module 10: Customer Profile & Saved Address Book [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `/profile` account tabs: edit personal details • Manage multiple saved shipping addresses (Add/Edit/Delete/Set Default) • Security preferences • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Module 10: Customer Profile & Saved Address Book) • Hands-On Frontend Implementation & Verification (Capstone Module 10: Customer Profile & Saved Address Book) • Capstone Module 10: Customer Profile & Saved Address Book — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 10

Purpose: Pre-selecting the user's default saved shipping address automatically in Step 1 of Checkout!

What Topic Is Next → Next Topic 24.12: Capstone Module 11: Role-Protected Admin Product & Order Dashboard

24.12 Capstone Module 11: Role-Protected Admin Product & Order Dashboard [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `/admin` role-guarded layout: KPI summary cards (Revenue • Orders • Low Stock) • Admin Product CRUD (`Create`/`Edit`/`Delete` modal/form with image preview & validation) • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Module 11: Role-Protected Admin Product & Order Dashboard) • Hands-On Frontend Implementation & Verification (Capstone Module 11: Role-Protected Admin Product & Order Dashboard) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 11

Purpose: Demonstrating both B2C Storefront UX AND B2B Back-Office Admin CRUD/RBAC inside one cohesive flagship project!

What Topic Is Next → Next Topic 24.13: Capstone Milestone 12: Multi-Device Responsiveness, Accessibility & Performance Polish

24.13 Capstone Milestone 12: Multi-Device Responsiveness, Accessibility & Performance Polish [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Full audit across `360px` mobile • `768px` tablet • `1440px` desktop • Keyboard Tab/Escape/ARIA verification • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Milestone 12: Multi-Device Responsiveness, Accessibility & Performance Polish) • Hands-On Frontend Implementation & Verification (Capstone Milestone 12: Multi-Device Responsiveness, Accessibility & Performance Polish) • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Milestone 12

Purpose: Ensuring recruiters opening your live link on an iPhone, Android phone, or MacBook experience zero horizontal overflow or layout shifts.

What Topic Is Next → Next Topic 24.14: Capstone Milestone 13: Automated Unit, Component & Playwright E2E Test Suite

24.14 Capstone Milestone 13: Automated Unit, Component & Playwright E2E Test Suite [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unit tests for `cartSlice`/pricing/coupons • RTL component tests for Auth/ProductCard/Checkout validation • Playwright E2E test for Search -> Add to Cart -> Checkout -> Order Confirmation • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Milestone 13: Automated Unit, Component & Playwright E2E Test Suite) • Hands-On Frontend Implementation & Verification (Capstone Milestone 13: Automated Unit, Component & Playwright E2E Test Suite) • Capstone Milestone 13: Automated Unit, Component & Playwright E2E Test Suite — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Milestone 13

Purpose: Proving production engineering discipline with green GitHub Actions CI badges on your Capstone `README.md`.

What Topic Is Next → Next Topic 24.15: Capstone Milestone 14 & 15: Live Cloud Deployment, Portfolio README & Technical Viva Defense

24.15 Capstone Milestone 14 & 15: Live Cloud Deployment, Portfolio README & Technical Viva Defense [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Deploying live HTTPS URL on Vercel/Netlify (with SPA rewrites + demo credentials button) • Comprehensive `README.md` with screenshots & architecture diagram • 30-minute live code defense viva • UI, Accessibility & Edge-Case Pitfall Guard (Capstone Milestone 14 & 15: Live Cloud Deployment, Portfolio README & Technical Viva Defense) • Hands-On Frontend Implementation & Verification (Capstone Milestone 14 & 15: Live Cloud Deployment, Portfolio README & Technical Viva Defense) • Capstone Milestone 14 & 15: Live Cloud Deployment, Portfolio README & Technical Viva Defense — Technical & Machine-Coding Interview Readiness • Idiomatic FDCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Frontend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Milestone 14

Purpose: Adding a one-click "Fill Demo Customer / Demo Admin Credentials" button on `/login` so busy hiring managers explore your entire app in 10 seconds!

What Topic Is Next → Module 24 Capstone → End-of-Course Live Client Project: Frontend Developer – Industry-Ready Career Program — Enterprise Client Production Platform

Module 24 Hands-On Capstone Project (100 Marks): Project 6 (Final Industry Capstone): Complete Enterprise E-Commerce Frontend Platform (Storefront + Admin) — Design, engineer, test, optimize, and deploy a flagship, recruiter-ready Enterprise E-Commerce Frontend Platform combining a high-conversion Customer Storefront AND a Role-Protected Admin Op...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: Frontend Developer – Industry-Ready Career Program — Enterprise Client Production Platform • **Confirmed Role:** Frontend Developer – Industry-Ready Career Program Engineering Project Intern

Domain & Production Infra: Live Enterprise Frontend React, TypeScript & Angular Engineering Client Engagement (HTML5 & Semantic Accessibility, CSS3, Flexbox & CSS Grid, Responsive Mobile-First UI, Sass / SCSS & Bootstrap 5, Tailwind CSS • Git CI/CD + Automated Testing + Live HTTPS Cloud Production)

- Translated live client functional and non-functional requirements into modular HTML5 & Semantic Accessibility, CSS3, Flexbox & CSS Grid, Responsive Mobile-First UI, Sass / SCSS & Bootstrap 5, Tailwind CSS production architecture.
- Implemented core application modules, input validation, security hardening, and optimized data/state workflows.
- Completed peer Git pull-request code reviews, automated verification suites, and edge-case debugging sprints.
- Deployed the verified build to live client production with monitoring, documentation, and mentor sign-off.

Target Job Roles: Frontend Developer – Industry-Ready Career Program Developer, Associate Frontend React, TypeScript & Angular Engineering Engineer, Software Development Engineer (SDE-1)

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹4.5 LPA – ₹8.5 LPA **Mid-Level Product Engineer (1–3 Yrs):** ₹9.5 LPA – ₹19.0 LPA **Senior / Multi-Stack Specialist (3–6+ Yrs):** ₹20.0 LPA – ₹38.0+ LPA

Optional Worthwhile Global Certifications (100% Exam-Success Mentorship Available): **OpenJS Node.js Application Developer (JSNAD):** ₹12.0 LPA – ₹27.0 LPA (Hands-On Coding Exam) • **AWS Certified Solutions Architect – Associate (SAA-C03):** ₹11.0 LPA – ₹26.0 LPA (#1 High-ROI Cloud Credential)

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 95400 07917 • Web: https://mockattempt.com/training/frontend-developer-career-program

Try 2 Free Topic-Wise Timed Topic Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.