Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: REAC • **Updated:** 18 Nov 2026
https://mockattempt.com/courses/reactjs-complete-developer-training

REAC • FRONTEND & REACT ECOSYSTEM **BEGINNER TO JOB-READY REACT ENGINEER**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

Complete React.js Developer Course (with HTML5, CSS3, Tailwind, JS & TypeScript)

Zero missing prerequisites: learn HTML5, CSS3 Flexbox/Grid, Tailwind CSS, JavaScript ES6+, and TypeScript 5 right inside the course before mastering React, Hooks, Router, RHF + Zod, Redux Toolkit, TanStack Query, Next.js fundamentals, Vitest, and live Vercel deployment.

Duration: 10 Weeks • Weekend Live (Sat–Sun • 2h/Day) + UI Coding Lab (111 Hrs) • **Modules:** 10 • **Live Topics:** 231 • **Subtopics:** 1,848+ • **Labs & Tasks:** 462 • **Mocks / AI Vivas:** 231 / 231 • **Projects:** 10 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹3,999 (Net Tuition ₹3,389 + Included 18% GST ₹610) • **Split Option:** $21 Lock + 1 × $21 • **Effective ROI:** ~₹36/Guided Hr (~₹25/Live Topic)

Faculty: 100% Active Working IT Professionals • **Weekend-Only Live Batch (7:00 AM – 8:00 PM IST Flexible Slots):** Weekend-Only Live Online 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) • 11–13 Weeks Weekend Track)

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

Core Technologies & Engineering Stack: HTML5 & Semantic A11y • CSS3, Flexbox & Grid • Tailwind CSS • JavaScript ES6+ & Async/DOM • TypeScript 5 (Strict) • React 18/19 & Hooks • React Router & Axios • React Hook Form & Zod • Redux Toolkit & TanStack Query • Next.js App Router Basics • Vitest & React Testing Library • Git & Vercel Deployment

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Build responsive, accessible layouts from scratch using Semantic HTML5, CSS3 Flexbox/Grid, and Tailwind CSS.
2. Write clean, bug-free JavaScript ES6+ (`map`/`filter`/`reduce`, closures, `async/await`) and zero-`any` TypeScript 5 interfaces & generics.
3. Architect production React applications with Custom Hooks, React Router, React Hook Form + Zod, Redux Toolkit, and TanStack Query.
4. Optimize Core Web Vitals, understand Next.js SSR/RSC fundamentals, write Vitest + RTL tests, solve React machine-coding rounds, and deploy live on Vercel.
5. Crack LeetCode and HackerRank problems in JavaScript ES6+, TypeScript 5 & React with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
6. Automated Testing, Debugging & Quality Gates (HTML5 & Semantic A11y, CSS3, Flexbox & Grid, Tailwind CSS, JavaScript ES6+ & Async/DOM, TypeScript 5 (Strict)) — Automated HTML5 & Semantic A11y, CSS3, Flexbox & Grid, Tailwind CSS, JavaScript ES6+ & Async/DOM, TypeScript 5 (Strict) test suites • Linting & static checks • Mentor Git pull-request reviews — ensuring your code never breaks silently when real users hit edge cases.
7. 100% Topic-Wise Timed Mock Tests, Spoken MockAttempt AI Viva & Live Client Production — Verify every single Complete React.js Developer Course (with HTML5, CSS3, Tailwind, JS & TypeScript) topic through timed MCQ exams and spoken technical interviews before deploying your Live Client Project.
8. Translated live client functional and non-functional requirements into modular HTML5 & Semantic A11y, CSS3, Flexbox & Grid, Tailwind CSS, JavaScript ES6+ & Async/DOM, TypeScript 5 (Strict) production architecture.
9. Implemented core application modules, input validation, security hardening, and optimized data/state workflows.
10. Completed peer Git pull-request code reviews, automated verification suites, and edge-case debugging sprints.
11. Deployed the verified build to live client production with monitoring, documentation, and mentor sign-off.
12. Clear 100% topic-wise Timed MCQ Mock Tests and Spoken MockAttempt AI Technical Interviews across every module of Complete React.js Developer Course (with HTML5, CSS3, Tailwind, JS & TypeScript).

Eligibility & Prerequisites: No prior HTML, CSS, JavaScript, TypeScript, or React experience required — Stage 1 (Modules 1–3) teaches all foundations from scratch! • Basic computer & internet familiarity on a Windows, macOS, or Linux laptop (8 GB+ RAM recommended for smooth live coding labs). • 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. • Dedication to write original code in your GitHub portfolio and complete the mentor-guided Live Client Production Sprint.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (10 MODULES • 231 LIVE TOPICS • 1,848+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–3: Month 1 – Stage 1: Built-In HTML5, CSS3/Tailwind, JavaScript ES6+ & TypeScript 5 Foundations (Weeks 1–3) (79 Topics / 632 Subtopics) • Modules 4–6: Month 2 – Stage 2: Core React, Hooks, Validated Forms, Routing & REST APIs (Weeks 4–6) (66 Topics / 528 Subtopics) • Modules 7–10: Month 3 – Stage 3: Redux Toolkit, TanStack Query, Next.js Basics, Testing & E-Commerce Capstone (Weeks 7–10) (86 Topics / 688 Subtopics)

MODULE 1 OF 10 • STAGE 1 • 11 LIVE + LAB HOURS • 25 TOPICS (200 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Semantic HTML5, CSS3 Flexbox/Grid, Responsive UI & Tailwind CSS Foundations

Build every web & styling foundation required for modern React UI engineering: DOM/CSSOM rendering, semantic HTML5 landmarks, accessible forms, Flexbox, CSS Grid, mobile-first breakpoints, and Tailwind CSS utility architecture.

Objectives: ✓ Author semantic, accessible HTML5 markup that translates cleanly into valid React JSX without nesting/hydration bugs. • ✓ Build responsive 1D/2D layouts using CSS3 Flexbox, CSS Grid, and mobile-first Tailwind CSS utilities (including Dark Mode & focus-visible states).

1.1 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) • Production Pitfall Guard (Browser Rendering Architecture (DOM and CSSOM)) • Hands-On Verification (Browser Rendering Architecture (DOM and CSSOM)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.2: HTML5 Introduction and Document Declaration

1.2 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 • Production Pitfall Guard (HTML5 Introduction and Document Declaration) • Hands-On Verification (HTML5 Introduction and Document Declaration) • HTML5 Introduction and Document Declaration — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.3: HTML Document Structure and Head Metadata

1.3 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">` • Production Pitfall Guard (HTML Document Structure and Head Metadata) • Hands-On Verification (HTML Document Structure and Head Metadata) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.4: HTML Elements (Block, Inline and Void Elements)

1.4 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` • Production Pitfall Guard (HTML Elements (Block, Inline and Void Elements)) • Hands-On Verification (HTML Elements (Block, Inline and Void Elements)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.5: Text Formatting and Semantic Phrasing

1.5 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>` • Production Pitfall Guard (Text Formatting and Semantic Phrasing) • Hands-On Verification (Text Formatting and Semantic Phrasing) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.6: Links and Navigation Anchors

1.6 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:` • Production Pitfall Guard (Links and Navigation Anchors) • Hands-On Verification (Links and Navigation Anchors) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.7: Images and Responsive Image Attributes

1.7 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"` • Production Pitfall Guard (Images and Responsive Image Attributes) • Hands-On Verification (Images and Responsive Image Attributes) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.8: Lists (Ordered, Unordered and Description Lists)

1.8 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 • Production Pitfall Guard (Lists (Ordered, Unordered and Description Lists)) • Hands-On Verification (Lists (Ordered, Unordered and Description Lists)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.9: Accessible HTML Tables

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

Subtopics Covered: `<table>` • `<caption>` • `<thead>` • `<tbody>` • Production Pitfall Guard (Accessible HTML Tables) • Hands-On Verification (Accessible HTML Tables) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.10: Flexbox Container and Flex Items

1.10 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!) • Production Pitfall Guard (Flexbox Container and Flex Items) • Hands-On Verification (Flexbox Container and Flex Items) • Flexbox Container and Flex Items — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.11: Flex Direction (row, column and reverse)

1.11 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` • Production Pitfall Guard (Flex Direction (row, column and reverse)) • Hands-On Verification (Flex Direction (row, column and reverse)) • Flex Direction (row, column and reverse) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.12: Flexbox Alignment (justify-content, align-items, align-self)

1.12 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` • Production Pitfall Guard (Flexbox Alignment (justify-content, align-items, align-self)) • Hands-On Verification (Flexbox Alignment (justify-content, align-items, align-self)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.13: Flex Wrapping (flex-wrap and align-content)

1.13 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 • Production Pitfall Guard (Flex Wrapping (flex-wrap and align-content)) • Hands-On Verification (Flex Wrapping (flex-wrap and align-content)) • Flex Wrapping (flex-wrap and align-content) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.14: Flex Sizing (flex-grow, flex-shrink, flex-basis & min-width: 0)

1.14 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! • Production Pitfall Guard (Flex Sizing (flex-grow, flex-shrink, flex-basis & min-width: 0)) • Hands-On Verification (Flex Sizing (flex-grow, flex-shrink, flex-basis & min-width: 0)) • Flex Sizing (flex-grow, flex-shrink, flex-basis & min-width: 0) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.15: CSS Grid Container and Grid Items

1.15 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 • Production Pitfall Guard (CSS Grid Container and Grid Items) • Hands-On Verification (CSS Grid Container and Grid Items) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.16: Grid Structure (Rows, Columns, fr Unit and Gaps)

1.16 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` • Production Pitfall Guard (Grid Structure (Rows, Columns, fr Unit and Gaps)) • Hands-On Verification (Grid Structure (Rows, Columns, fr Unit and Gaps)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.17: Grid Templates (grid-template-columns, rows and grid-auto-rows)

1.17 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 • Production Pitfall Guard (Grid Templates (grid-template-columns, rows and grid-auto-rows)) • Hands-On Verification (Grid Templates (grid-template-columns, rows and grid-auto-rows)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.18: Tailwind CSS Utility-First Architecture

1.18 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 • Production Pitfall Guard (Tailwind CSS Utility-First Architecture) • Hands-On Verification (Tailwind CSS Utility-First Architecture) • Tailwind CSS Utility-First Architecture — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Tailwind CSS Utility-First Architecture • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

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

What Topic Is Next → Next Topic 1.19: Tailwind CSS Setup with Vite

1.19 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 • Production Pitfall Guard (Tailwind CSS Setup with Vite) • Hands-On Verification (Tailwind CSS Setup with Vite) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.20: Tailwind Spacing, Typography and Color Tokens

1.20 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` • Production Pitfall Guard (Tailwind Spacing, Typography and Color Tokens) • Hands-On Verification (Tailwind Spacing, Typography and Color Tokens) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.21: Tailwind Flexbox and CSS Grid Utilities

1.21 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` • Production Pitfall Guard (Tailwind Flexbox and CSS Grid Utilities) • Hands-On Verification (Tailwind Flexbox and CSS Grid Utilities) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.22: Tailwind Responsive Breakpoint Prefixes

1.22 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`) • Production Pitfall Guard (Tailwind Responsive Breakpoint Prefixes) • Hands-On Verification (Tailwind Responsive Breakpoint Prefixes) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.23: Tailwind Interactive State Variants

1.23 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` • Production Pitfall Guard (Tailwind Interactive State Variants) • Hands-On Verification (Tailwind Interactive State Variants) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.24: Tailwind Dark Mode Implementation

1.24 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` • Production Pitfall Guard (Tailwind Dark Mode Implementation) • Hands-On Verification (Tailwind Dark Mode Implementation) • Tailwind Dark Mode Implementation — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 1.25: Reusable UI Component Design Patterns

1.25 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`) • Production Pitfall Guard (Reusable UI Component Design Patterns) • Hands-On Verification (Reusable UI Component Design Patterns) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 → Module 1 Capstone (Project 1: Complete Responsive Multi-Page Business Website) → Next Topic 2.1: Array Traversal (for, for...of, entries and forEach)

Module 1 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 2 OF 10 • STAGE 1 • 11 LIVE + LAB HOURS • 27 TOPICS (216 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Modern JavaScript (ES6+), Closures, Event Loop, Async/Await & DOM Foundations

Master the exact JavaScript ES6+ engine skills that power every line of React code: immutable `map`/`filter`/`reduce`, destructuring/spread, ES modules, closures, Event Loop, Promises, `async/await`, `fetch` + `AbortController`, and DOM events.

Objectives: ✓ Transform arrays and nested state objects immutably using `map`, `filter`, `reduce`, `find`, `some`, `every`, destructuring, and spread. • ✓ Understand lexical closures, stale-closure prevention, browser Event Loop microtasks, `async/await` REST fetching, and DOM event delegation.

2.1 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!) • Production Pitfall Guard (Array Traversal (for, for...of, entries and forEach)) • Hands-On Verification (Array Traversal (for, for...of, entries and forEach)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.2: Array.prototype.map

2.2 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! • Production Pitfall Guard (Array.prototype.map) • Hands-On Verification (Array.prototype.map) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.3: Array.prototype.filter

2.3 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(...)` • Production Pitfall Guard (Array.prototype.filter) • Hands-On Verification (Array.prototype.filter) • Array.prototype.filter — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.4: Array.prototype.reduce

2.4 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` • `{}`) • Production Pitfall Guard (Array.prototype.reduce) • Hands-On Verification (Array.prototype.reduce) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.5: Array.prototype.find and findIndex

2.5 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`) • Production Pitfall Guard (Array.prototype.find and findIndex) • Hands-On Verification (Array.prototype.find and findIndex) • Array.prototype.find and findIndex — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.6: Array.prototype.some

2.6 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 `[]`) • Production Pitfall Guard (Array.prototype.some) • Hands-On Verification (Array.prototype.some) • Array.prototype.some — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for

Array.prototype.some • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

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 2.7: Array.prototype.every

2.7 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`!) • Production Pitfall Guard (Array.prototype.every) • Hands-On Verification (Array.prototype.every) • Array.prototype.every — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Array.prototype.every • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

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

What Topic Is Next → Next Topic 2.8: Array.prototype.sort and toSorted (Custom Comparators)

2.8 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)` • Production Pitfall Guard (Array.prototype.sort and toSorted (Custom Comparators)) • Hands-On Verification (Array.prototype.sort and toSorted (Custom Comparators)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.9: Array.prototype.slice (Non-Mutating Subarrays)

2.9 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 • Production Pitfall Guard (Array.prototype.slice (Non-Mutating Subarrays)) • Hands-On Verification (Array.prototype.slice (Non-Mutating Subarrays)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.10: Array.prototype.splice and toSpliced

2.10 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 • Production Pitfall Guard (Array.prototype.splice and toSpliced) • Hands-On Verification (Array.prototype.splice and toSpliced) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.11: Array Destructuring

2.11 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 `[• Production Pitfall Guard (Array Destructuring) • Hands-On Verification (Array Destructuring) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Array Destructuring

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

What Topic Is Next → Next Topic 2.12: JavaScript Objects (Creation, Dynamic Keys and Reference Semantics)

2.12 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)` • Production Pitfall Guard (JavaScript Objects (Creation, Dynamic Keys and Reference Semantics)) • Hands-On Verification (JavaScript Objects (Creation, Dynamic Keys and Reference Semantics)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.13: Execution Contexts (Global and Function)

2.13 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 • Production Pitfall Guard (Execution Contexts (Global and Function)) • Hands-On Verification (Execution Contexts (Global and Function)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.14: The JavaScript Call Stack

2.14 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 • Production Pitfall Guard (The JavaScript Call Stack) • Hands-On Verification (The JavaScript Call Stack) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.15: Lexical Environment and the Scope Chain

2.15 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 • Production Pitfall Guard (Lexical Environment and the Scope Chain) • Hands-On Verification (Lexical Environment and the Scope Chain) • Lexical Environment and the Scope Chain — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Lexical Environment • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

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

What Topic Is Next → Next Topic 2.16: Closures (Mechanism, Practical Uses and Memory Nuances)

2.16 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)` • Production Pitfall Guard (Closures (Mechanism, Practical Uses and Memory Nuances)) • Hands-On Verification (Closures (Mechanism, Practical Uses and Memory Nuances)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.17: The this Keyword (4 Binding Rules)

2.17 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!) • Production Pitfall Guard (The this Keyword (4 Binding Rules)) • Hands-On Verification (The this Keyword (4 Binding Rules)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.18: Explicit Context Control (call, apply and bind)

2.18 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) • Production Pitfall Guard (Explicit Context Control (call, apply and bind)) • Hands-On Verification (Explicit Context Control (call, apply and bind)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.19: Prototypes and the [[Prototype]] Chain

2.19 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 • Production Pitfall Guard (Prototypes and the [[Prototype]] Chain) • Hands-On Verification (Prototypes and the [[Prototype]] Chain) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.20: Prototypal Inheritance (Object.create and Class Sugar)

2.20 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 • Production Pitfall Guard (Prototypal Inheritance (Object.create and Class Sugar)) • Hands-On Verification (Prototypal Inheritance (Object.create and Class Sugar)) • Prototypal Inheritance (Object.create and Class Sugar) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.21: Browser Event Loop Architecture

2.21 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 • Production Pitfall Guard (Browser Event Loop Architecture) • Hands-On Verification (Browser Event Loop Architecture) • Browser Event Loop Architecture — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.22: Microtask Queue (Promises & queueMicrotask)

2.22 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 • Production Pitfall Guard (Microtask Queue (Promises & queueMicrotask)) • Hands-On Verification (Microtask Queue (Promises & queueMicrotask)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.23: Document Object Model (DOM) Tree Architecture

2.23 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 • Production Pitfall Guard (Document Object Model (DOM) Tree Architecture) • Hands-On Verification (Document Object Model (DOM) Tree Architecture) • Document Object Model (DOM) Tree Architecture — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.24: DOM Selection (getElementById, querySelector, querySelectorAll, closest)

2.24 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!) • Production Pitfall Guard (DOM Selection (getElementById, querySelector, querySelectorAll, closest)) • Hands-On Verification (DOM Selection (getElementById, querySelector, querySelectorAll, closest)) • DOM Selection (getElementById, querySelector, querySelectorAll, closest) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.25: DOM Manipulation (createElement, textContent, append, replaceChildren, remove)

2.25 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!) • Production Pitfall Guard (DOM Manipulation (createElement, textContent, append, replaceChildren, remove)) • Hands-On Verification (DOM Manipulation (createElement, textContent, append, replaceChildren, remove)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.26: HTML Attributes and dataset Manipulation

2.26 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`) • Production Pitfall Guard (HTML Attributes and dataset Manipulation) • Hands-On Verification (HTML Attributes and dataset Manipulation) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 2.27: CSS Class Manipulation with classList

2.27 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!) • Production Pitfall Guard (CSS Class Manipulation with classList) • Hands-On Verification (CSS Class Manipulation with classList) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 → Module 2 Capstone (Project 2: JavaScript Task Management Application (ES6+, DOM & LocalStorage)) → Next Topic 3.1: TypeScript Introduction (JavaScript vs TypeScript)

Module 2 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 3 OF 10 • STAGE 1 • 11 LIVE + LAB HOURS • 27 TOPICS (216 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: 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.

3.1 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!) • Production Pitfall Guard (TypeScript Introduction (JavaScript vs TypeScript)) • Hands-On Verification (TypeScript Introduction (JavaScript vs TypeScript)) • TypeScript Introduction (JavaScript vs TypeScript) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.2: TypeScript Environment Setup (Node.js, npm & Vite)

3.2 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 • Production Pitfall Guard (TypeScript Environment Setup (Node.js, npm & Vite)) • Hands-On Verification (TypeScript Environment Setup (Node.js, npm & Vite)) • TypeScript Environment Setup (Node.js, npm & Vite) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache &

route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.3: tsconfig.json and Strict Compiler Options

3.3 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` • Production Pitfall Guard (tsconfig.json and Strict Compiler Options) • Hands-On Verification (tsconfig.json and Strict Compiler Options) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.4: Basic Primitive Types (string, number, boolean, null, undefined, bigint, symbol)

3.4 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`) • Production Pitfall Guard (Basic Primitive Types (string, number, boolean, null, undefined, bigint, symbol)) • Hands-On Verification (Basic Primitive Types (string, number, boolean, null, undefined, bigint, symbol)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.5: Special Types (any, unknown, never and void)

3.5 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` • Production Pitfall Guard (Special Types (any, unknown, never and void)) • Hands-On Verification (Special Types (any, unknown, never and void)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.6: Typed Arrays and ReadonlyArray<T>

3.6 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! • Production Pitfall Guard (Typed Arrays and ReadonlyArray<T>) • Hands-On Verification (Typed Arrays and ReadonlyArray<T>) • Typed Arrays and ReadonlyArray<T> — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Typed Arrays • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

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

What Topic Is Next → Next Topic 3.7: Tuples and Labeled Tuples

3.7 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`)! • Production Pitfall Guard (Tuples and Labeled Tuples) • Hands-On Verification (Tuples and Labeled Tuples) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.8: Enums vs Const String Literal Unions

3.8 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 • Production Pitfall Guard (Enums vs Const String Literal Unions) • Hands-On Verification (Enums vs Const String Literal Unions) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.9: Type Inference

3.9 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 • Production Pitfall Guard (Type Inference) • Hands-On Verification (Type Inference) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.10: Explicit Type Annotations (When Required vs Redundant)

3.10 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` • Production Pitfall Guard (Explicit Type Annotations (When Required vs Redundant)) • Hands-On Verification (Explicit Type Annotations (When Required vs Redundant)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.11: Typed Functions, Optional/Default Params and Overloads

3.11 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 • Production Pitfall Guard (Typed Functions, Optional/Default Params and Overloads) • Hands-On Verification (Typed Functions, Optional/Default Params and Overloads) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.12: Object Types, Optional (?) and readonly Modifiers

3.12 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` • Production Pitfall Guard (Object Types, Optional (?) and readonly Modifiers) • Hands-On Verification (Object Types, Optional (?) and readonly Modifiers) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.13: Interfaces and Extends Inheritance

3.13 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 • Production Pitfall Guard (Interfaces and Extends Inheritance) • Hands-On Verification (Interfaces and Extends Inheritance) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.14: Type Aliases (type) vs Interfaces

3.14 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 • Production Pitfall Guard (Type Aliases (type) vs Interfaces) • Hands-On Verification (Type Aliases (type) vs Interfaces) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.15: Union Types (|) and Discriminated (Tagged) Unions

3.15 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 `?`! • Production Pitfall Guard (Union Types (|) and Discriminated (Tagged) Unions) • Hands-On Verification (Union Types (|) and Discriminated (Tagged) Unions) • Union Types (|) and Discriminated (Tagged) Unions — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Union Types (|) • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

Purpose: Eliminating impossible `{ isLoading: true, error: "Failed", data: [...] }` boolean-soup states in React and Angular!

What Topic Is Next → Next Topic 3.16: Intersection Types (&)

3.16 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`! • Production Pitfall Guard (Intersection Types (&)) • Hands-On Verification (Intersection Types (&)) • Intersection Types (&) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Intersection Types (&)

Purpose: Composing `EmployeeSearchFilter & PaginationParams` cleanly for API query hooks.

What Topic Is Next → Next Topic 3.17: Literal Types and Template Literal Types

3.17 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}` `` • Production Pitfall Guard (Literal Types and Template Literal Types) • Hands-On Verification (Literal Types and Template Literal Types) • Literal Types and Template Literal Types — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Literal Types • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

Purpose: Autocompleting valid design-system `<Button variant="primary" size="md" />` props in VS Code.

What Topic Is Next → Next Topic 3.18: Type Narrowing (typeof, instanceof, in and Custom Type Guards)

3.18 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` • Production Pitfall Guard (Type Narrowing (typeof, instanceof, in and Custom Type Guards)) • Hands-On Verification (Type Narrowing (typeof, instanceof, in and Custom Type Guards)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.19: Type Assertions (as) vs the satisfies Operator

3.19 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!) • Production Pitfall Guard (Type Assertions (as) vs the satisfies Operator) • Hands-On Verification (Type Assertions (as) vs the satisfies

Operator) • Type Assertions (as) vs the satisfies Operator — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.20: Generics (Functions, Interfaces and Classes)

3.20 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[] • Production Pitfall Guard (Generics (Functions, Interfaces and Classes)) • Hands-On Verification (Generics (Functions, Interfaces and Classes)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.21: Generic Constraints (extends and keyof)

3.21 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 }>` • Production Pitfall Guard (Generic Constraints (extends and keyof)) • Hands-On Verification (Generic Constraints (extends and keyof)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.22: Built-in Utility Types (Partial, Required, Pick, Omit, Record, Readonly, ReturnType)

3.22 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 • Production Pitfall Guard (Built-in Utility Types (Partial, Required, Pick, Omit, Record, Readonly, ReturnType)) • Hands-On Verification (Built-in Utility Types (Partial, Required, Pick, Omit, Record, Readonly, ReturnType)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.23: Advanced Types (Indexed Access, Mapped and Conditional Types)

3.23 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>`) • Production Pitfall Guard (Advanced Types (Indexed Access, Mapped and Conditional Types)) • Hands-On Verification (Advanced Types (Indexed Access, Mapped and Conditional Types)) • Advanced Types (Indexed Access, Mapped and Conditional Types) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.24: TypeScript Modules and Type-Only Imports (import type)

3.24 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`) • Production Pitfall Guard (TypeScript Modules and Type-Only Imports (import type)) • Hands-On Verification (TypeScript Modules and Type-Only Imports (import type)) • TypeScript Modules and Type-Only Imports (import type) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.25: TypeScript Classes, Abstract Classes and Parameter Properties

3.25 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` • Production Pitfall Guard (TypeScript Classes, Abstract Classes and Parameter Properties) • Hands-On Verification (TypeScript Classes, Abstract Classes and Parameter Properties) • TypeScript Classes, Abstract Classes and Parameter Properties — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.26: Using JavaScript Libraries & Ambient Declaration Files (.d.ts)

3.26 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) • Production Pitfall Guard (Using JavaScript Libraries & Ambient Declaration Files (.d.ts)) • Hands-On Verification (Using JavaScript Libraries & Ambient Declaration Files (.d.ts)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3.27: TypeScript Debugging, Source Maps and JS-to-TS Migration

3.27 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 • Production Pitfall Guard (TypeScript Debugging, Source Maps and JS-to-TS Migration) • Hands-On Verification (TypeScript Debugging, Source Maps and JS-to-TS Migration) • TypeScript Debugging, Source Maps and JS-to-TS Migration — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 3 Capstone (Week 9 Lab Build: Strictly Typed TypeScript 5 Domain SDK (`strict: true`)) → Next Topic 4.1: React Introduction and Declarative Component Architecture

Module 3 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 4 OF 10 • STAGE 2 • 11 LIVE + LAB HOURS • 20 TOPICS (160 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: 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.

4.1 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 • Production Pitfall Guard (React Introduction and Declarative Component Architecture) • Hands-On Verification (React Introduction and Declarative Component Architecture) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.2: Vite + React + TypeScript Project Setup

4.2 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! • Production Pitfall Guard (Vite + React + TypeScript Project Setup) • Hands-On Verification (Vite + React + TypeScript Project Setup) • Vite + React + TypeScript Project Setup — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.3: React Application Structure (src, public, assets & Entry Points)

4.3 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`) • Production Pitfall Guard (React Application Structure (src, public, assets & Entry Points)) • Hands-On Verification (React Application Structure (src, public, assets & Entry Points)) • React Application Structure (src, public, assets & Entry Points) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.4: JSX Syntax, Expressions and Automatic XSS Escaping

4.4 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!) • Production Pitfall Guard (JSX Syntax, Expressions and Automatic XSS Escaping) • Hands-On Verification (JSX Syntax, Expressions and Automatic XSS Escaping) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.5: Functional Components and Purity Rules

4.5 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! • Production Pitfall Guard (Functional Components and Purity Rules) • Hands-On Verification (Functional Components and Purity Rules) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Functional Components

Purpose: Keeping every component deterministic and fast to test.

What Topic Is Next → Next Topic 4.6: Component Composition

4.6 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 • Production Pitfall Guard (Component Composition) • Hands-On Verification (Component Composition) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Component Composition

Purpose: Keeping every component file `< 150` lines with a single clear responsibility.

What Topic Is Next → Next Topic 4.7: Props and Unidirectional Data Flow

4.7 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)` • Production Pitfall Guard (Props and Unidirectional Data Flow) • Hands-On Verification (Props and Unidirectional Data Flow) • Props and Unidirectional Data Flow — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.8: TypeScript Props Validation and Default Values

4.8 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` • Production Pitfall Guard (TypeScript Props Validation and Default Values) • Hands-On Verification (TypeScript Props Validation and Default Values) • TypeScript Props Validation and Default Values — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.9: The children Prop and Slot Composition

4.9 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! • Production Pitfall Guard (The children Prop and Slot Composition) • Hands-On Verification (The children Prop and Slot Composition) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.10: Conditional Rendering Patterns (Avoiding the 0 Trap!)

4.10 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 />}`)! • Production Pitfall Guard (Conditional Rendering Patterns (Avoiding the 0 Trap!)) • Hands-On Verification (Conditional Rendering Patterns (Avoiding the 0 Trap!)) • Conditional Rendering Patterns (Avoiding the 0 Trap!) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.11: Rendering Lists with .map()

4.11 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 • Production Pitfall Guard (Rendering Lists with .map()) • Hands-On Verification (Rendering Lists with .map()) • Rendering Lists with .map() — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.12: Unique Keys and React Reconciliation

4.12 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 • Production Pitfall Guard (Unique Keys and React Reconciliation) • Hands-On Verification (Unique Keys and React Reconciliation) • Unique Keys and React Reconciliation — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.13: React Synthetic Events and Typed Event Handlers

4.13 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)}` • Production Pitfall Guard (React Synthetic Events and Typed Event Handlers) • Hands-On Verification (React Synthetic Events and Typed Event Handlers) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for React Synthetic Events

Purpose: Writing strictly typed `handleSubmit` and `handleInputChange` handlers without `: any`.

What Topic Is Next → Next Topic 4.14: Component State vs Local Variables & Derived State

4.14 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`! • Production Pitfall Guard (Component State vs Local Variables & Derived State) • Hands-On Verification (Component State vs Local Variables & Derived State) • Component State vs Local Variables & Derived State — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.15: useState Hook Mastery

4.15 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!) • Production Pitfall Guard (useState Hook Mastery) • Hands-On Verification (useState Hook Mastery) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.16: Immutable State Updates & Functional Updaters

4.16 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)` • Production Pitfall Guard (Immutable State Updates & Functional Updaters) • Hands-On Verification (Immutable State Updates & Functional Updaters) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.17: Controlled Components (Single Source of Truth Inputs)

4.17 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 ?? ""}`!) • Production Pitfall Guard (Controlled Components (Single Source of Truth Inputs)) • Hands-On Verification (Controlled Components (Single Source of Truth Inputs)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.18: Uncontrolled Components and FormData / Ref Access

4.18 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!) • Production Pitfall Guard (Uncontrolled Components and FormData / Ref Access) • Hands-On Verification (Uncontrolled Components and FormData / Ref Access) • Uncontrolled Components and FormData / Ref Access — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.19: Component Lifecycle Concepts (Mount, Update and Unmount)

4.19 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}`! • Production Pitfall Guard (Component Lifecycle Concepts (Mount, Update and Unmount)) • Hands-On Verification (Component Lifecycle Concepts (Mount, Update and Unmount)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4.20: Styling React Apps, React DevTools & Feature-Based Project Structure

4.20 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 • Production Pitfall Guard (Styling React Apps, React DevTools & Feature-Based Project Structure) • Hands-On Verification (Styling React Apps, React DevTools & Feature-Based Project Structure) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 4 Capstone (Week 10 Lab Build: Modular Vite + React + TypeScript Interactive Catalog UI) → Next Topic 5.1: Rules of Hooks and Fiber Hook Linked-List Internals

Module 4 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 5 OF 10 • STAGE 2 • 11 LIVE + LAB HOURS • 22 TOPICS (176 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: 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`.

5.1 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! • Production Pitfall Guard (Rules of Hooks and Fiber Hook Linked-List Internals) • Hands-On Verification (Rules of Hooks and Fiber Hook Linked-List Internals) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.2: Deep-Dive useState Patterns

5.2 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 • Production Pitfall Guard (Deep-Dive useState Patterns) • Hands-On Verification (Deep-Dive useState Patterns) • Deep-Dive useState Patterns — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Deep-Dive useState Patterns

Purpose: Structuring clean modal, stepper, and filter states.

What Topic Is Next → Next Topic 5.3: useEffect (Synchronizing with External Systems)

5.3 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 • Production Pitfall Guard (useEffect (Synchronizing with External Systems)) • Hands-On Verification (useEffect (Synchronizing with External Systems)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.4: Effect Dependency Arrays and Stale Closure Prevention

5.4 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 • Production Pitfall Guard (Effect Dependency Arrays and Stale Closure Prevention) • Hands-On Verification (Effect Dependency Arrays and Stale Closure Prevention) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.5: Effect Cleanup Functions

5.5 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` • Production Pitfall Guard (Effect Cleanup Functions) • Hands-On Verification (Effect Cleanup Functions) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.6: useRef (DOM References and Non-Rendering Mutable Boxes)

5.6 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! • Production Pitfall Guard (useRef (DOM References and Non-Rendering Mutable Boxes)) • Hands-On Verification (useRef (DOM References and Non-Rendering Mutable Boxes)) • useRef (DOM References and Non-Rendering Mutable Boxes) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.7: useContext (Scoped Shared State Without Prop Drilling)

5.7 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>`! • Production Pitfall Guard (useContext (Scoped Shared State Without Prop Drilling)) • Hands-On Verification (useContext (Scoped Shared State Without Prop Drilling)) • useContext (Scoped Shared State Without Prop Drilling) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.8: useReducer (Predictable Event-Driven State Transitions)

5.8 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! • Production Pitfall Guard (useReducer (Predictable Event-Driven State Transitions)) • Hands-On Verification (useReducer (Predictable Event-Driven State Transitions)) • useReducer (Predictable Event-Driven State Transitions) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.9: useMemo (Memoizing Expensive Derived Computations)

5.9 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 • Production Pitfall Guard (useMemo (Memoizing Expensive Derived Computations)) • Hands-On Verification (useMemo (Memoizing Expensive Derived Computations)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.10: useCallback (Referential Stability for Functions)

5.10 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! • Production Pitfall Guard (useCallback (Referential Stability for Functions)) • Hands-On Verification (useCallback (Referential Stability for Functions)) • useCallback (Referential Stability for Functions) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.11: Custom Hooks (Extracting Reusable Stateful Logic)

5.11 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 • Production Pitfall Guard (Custom Hooks (Extracting Reusable Stateful Logic)) • Hands-On Verification (Custom Hooks (Extracting Reusable Stateful Logic)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.12: Hook Composition (Combining Custom Hooks)

5.12 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`) • Production Pitfall Guard (Hook Composition (Combining Custom Hooks)) • Hands-On Verification (Hook Composition (Combining Custom Hooks)) • Hook Composition (Combining Custom Hooks) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Hook Composition (Combining Custom Hooks) • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

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 5.13: Controlled Forms Architecture in React

5.13 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 • Production Pitfall Guard (Controlled Forms Architecture in React) • Hands-On Verification (Controlled Forms Architecture in React) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.14: Client-Side Form Validation Logic & UX Patterns

5.14 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! • Production Pitfall Guard (Client-Side Form Validation Logic & UX Patterns) • Hands-On Verification (Client-Side Form Validation Logic & UX Patterns) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.15: React Hook Form (Uncontrolled-Performance Typed Forms)

5.15 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)` • Production Pitfall Guard (React Hook Form (Uncontrolled-Performance Typed Forms)) • Hands-On Verification (React Hook Form (Uncontrolled-Performance Typed Forms)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.16: Zod Schema Validation & Type Inference

5.16 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!) • Production Pitfall Guard (Zod Schema Validation & Type Inference) • Hands-On Verification (Zod Schema Validation & Type Inference) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.17: Error Boundaries (Graceful Render Crash Recovery)

5.17 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) • Production Pitfall Guard (Error Boundaries (Graceful Render Crash Recovery)) • Hands-On Verification (Error Boundaries (Graceful Render Crash Recovery)) • Error Boundaries (Graceful Render Crash Recovery) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Error Boundaries (Graceful Render Crash Recovery) • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

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 5.18: Code Splitting with React.lazy

5.18 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 • Production Pitfall Guard (Code Splitting with React.lazy) • Hands-On Verification (Code Splitting with React.lazy) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.19: React Suspense Boundaries

5.19 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 • Production Pitfall Guard (React Suspense Boundaries) • Hands-On Verification (React Suspense Boundaries) • React Suspense Boundaries — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for React Suspense Boundaries • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

Purpose: Placing smooth layout-matched `<DashboardSkeleton />` Suspense boundaries around lazy route outlets.

What Topic Is Next → Next Topic 5.20: React Fragments (<>...<\/> and Keyed <Fragment>)

5.20 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 • Production Pitfall Guard (React Fragments (<>...<\/> and Keyed <Fragment>)) • Hands-On Verification (React Fragments (<>...<\/> and Keyed <Fragment>)) • React Fragments (<>...<\/> and Keyed <Fragment>) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for React Fragments (<>...<\/> and Keyed <Fragment>) • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

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 5.21: React Portals (createPortal for Modals, Drawers & Toasts)

5.21 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! • Production Pitfall Guard (React Portals (createPortal for Modals, Drawers & Toasts)) • Hands-On Verification (React Portals (createPortal for Modals, Drawers & Toasts)) • React Portals (createPortal for Modals, Drawers & Toasts) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 5.22: Ref Forwarding & Context Performance Patterns

5.22 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! • Production Pitfall Guard (Ref Forwarding & Context Performance Patterns) • Hands-On Verification (Ref Forwarding & Context Performance Patterns) • Ref Forwarding & Context Performance Patterns — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Ref Forwarding • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

Purpose: Ensuring `React Hook Form`'s `{...register("email")}` ref attaches cleanly to your custom design-system `<TextField>` component!

What Topic Is Next → Module 5 Capstone (Week 11 Lab Build: Custom Hooks & Zod-Validated Multi-Step Onboarding Form) → Next Topic 6.1: Client-Side SPA Routing with React Router

Module 5 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 6 OF 10 • STAGE 2 • 11 LIVE + LAB HOURS • 24 TOPICS (192 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: 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.

6.1 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!) • Production Pitfall Guard (Client-Side SPA Routing with React Router) • Hands-On Verification (Client-Side SPA Routing with React Router) • Client-Side SPA Routing with React Router — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Client-Side SPA Routing with React Router • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

Purpose: Instant sub-frame route transitions with persistent shell navigation and audio/state continuity.

What Topic Is Next → Next Topic 6.2: BrowserRouter and Router Setup

6.2 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>` • Production Pitfall Guard (BrowserRouter and Router Setup) • Hands-On Verification (BrowserRouter and Router Setup) • BrowserRouter and Router Setup — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for BrowserRouter

Purpose: Structuring all application routes in a single declarative `AppRouter.tsx` module.

What Topic Is Next → Next Topic 6.3: Route Definitions and Catch-All 404 Pages

6.3 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 • Production Pitfall Guard (Route Definitions and Catch-All 404 Pages) • Hands-On Verification (Route Definitions and Catch-All 404 Pages) • Route Definitions and Catch-All 404 Pages — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Route Definitions

Purpose: Giving users a helpful 404 recovery screen when visiting mistyped URLs.

What Topic Is Next → Next Topic 6.4: Dynamic Route Parameters (useParams)

6.4 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 • Production Pitfall Guard (Dynamic Route Parameters (useParams)) • Hands-On Verification (Dynamic Route Parameters (useParams)) • Dynamic Route Parameters (useParams) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 6.5: Nested Routing and Persistent <Outlet /> Layouts

6.5 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! • Production Pitfall Guard (Nested Routing and Persistent <Outlet /> Layouts) • Hands-On Verification (Nested Routing and Persistent <Outlet /> Layouts) • Nested Routing and Persistent <Outlet /> Layouts — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 6.6: Navigation (<Link>, <NavLink> and useNavigate)

6.6 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 • Production Pitfall Guard (Navigation (<Link>, <NavLink> and useNavigate)) • Hands-On Verification (Navigation (<Link>, <NavLink> and useNavigate)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 6.7: Protected Routes (Authentication Guards)

6.7 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 />` • Production Pitfall Guard (Protected Routes (Authentication Guards)) • Hands-On Verification (Protected Routes (Authentication Guards)) • Protected Routes (Authentication Guards) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Protected Routes (Authentication Guards) • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

Purpose: Sending deep-linked users (`/employees/42` -> `/login` -> right back to `/employees/42` after login!).

What Topic Is Next → Next Topic 6.8: Role-Based Route Guards (Authorization)

6.8 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!) • Production Pitfall Guard (Role-Based Route Guards (Authorization)) • Hands-On Verification (Role-Based Route Guards (Authorization)) • Role-Based Route Guards (Authorization) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Role-Based Route Guards

(Authorization) • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

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 6.9: Route-Based Lazy Code Splitting

6.9 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 • Production Pitfall Guard (Route-Based Lazy Code Splitting) • Hands-On Verification (Route-Based Lazy Code Splitting) • Route-Based Lazy Code Splitting — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Route-Based Lazy Code Splitting • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

Purpose: Cutting initial First Load JS by 40–60% on multi-module portals.

What Topic Is Next → Next Topic 6.10: REST API Integration Architecture in React

6.10 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! • Production Pitfall Guard (REST API Integration Architecture in React) • Hands-On Verification (REST API Integration Architecture in React) • REST API Integration Architecture in React — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for REST API Integration Architecture in React • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

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 6.11: Fetch API Integration with Cleanup in React

6.11 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()`!) • Production Pitfall Guard (Fetch API Integration with Cleanup in React) • Hands-On Verification (Fetch API Integration with Cleanup in React) • Fetch API Integration with Cleanup in React — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 6.12: Axios HTTP Client Configuration

6.12 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 • Production Pitfall Guard (Axios HTTP Client Configuration) • Hands-On Verification (Axios HTTP Client Configuration) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 6.13: Typed GET Requests and Query Parameters

6.13 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 • Production Pitfall Guard (Typed GET Requests and Query Parameters) • Hands-On Verification (Typed GET Requests and Query Parameters) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Typed GET Requests

Purpose: Fetching paginated, filtered employee directories cleanly with typed responses.

What Topic Is Next → Next Topic 6.14: Typed POST Requests (Creating Resources)

6.14 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` • Production Pitfall Guard (Typed POST Requests (Creating Resources)) • Hands-On Verification (Typed POST Requests (Creating Resources)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 6.15: PUT and PATCH Requests (Updating Resources)

6.15 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 • Production Pitfall Guard (PUT and PATCH Requests (Updating Resources)) • Hands-On Verification (PUT and PATCH Requests (Updating Resources)) • PUT and PATCH Requests (Updating Resources) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 6.16: DELETE Requests and Accessible Confirmation Modals

6.16 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 • Production Pitfall Guard (DELETE Requests and Accessible Confirmation Modals) • Hands-On Verification (DELETE Requests and Accessible Confirmation Modals) • DELETE Requests and Accessible Confirmation Modals — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 6.17: HTTP Headers (Authorization Bearer, Accept & Idempotency)

6.17 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` • Production Pitfall Guard (HTTP Headers (Authorization Bearer, Accept & Idempotency)) • Hands-On Verification (HTTP Headers (Authorization Bearer, Accept & Idempotency)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 6.18: Axios Request and Response Interceptors

6.18 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!) • Production Pitfall Guard (Axios Request and Response Interceptors) • Hands-On Verification (Axios Request and Response Interceptors) • Axios Request and Response Interceptors — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 6.19: Graceful API Error Handling (axios.isAxiosError & ProblemDetail)

6.19 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 • Production Pitfall Guard (Graceful API Error Handling (axios.isAxiosError & ProblemDetail)) • Hands-On Verification (Graceful API Error Handling (axios.isAxiosError & ProblemDetail)) • Graceful API Error Handling (axios.isAxiosError & ProblemDetail) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Graceful API Error Handling (axios.isAxiosError & Pr... • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

Purpose: Turning raw HTTP failures into clear, actionable human messages (`"Email already registered"` on the email input!).

What Topic Is Next → Next Topic 6.20: Loading States (Layout-Preserving Skeletons vs Spinners)

6.20 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 • Production Pitfall Guard (Loading States (Layout-Preserving Skeletons vs Spinners)) • Hands-On Verification (Loading States (Layout-Preserving Skeletons vs Spinners)) • Loading States (Layout-Preserving Skeletons vs Spinners) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 6.21: Empty States (Zero-Data vs Zero-Search-Matches UX)

6.21 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!) • Production Pitfall Guard (Empty States (Zero-Data vs Zero-Search-Matches UX)) • Hands-On Verification (Empty States (Zero-Data vs Zero-Search-Matches UX)) • Empty States (Zero-Data vs Zero-Search-Matches UX) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Empty States (Zero-Data vs Zero-Search-Matches UX) • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

Purpose: Giving users a 1-click `"Reset Search & Filters"` button whenever a filter combination returns 0 rows.

What Topic Is Next → Next Topic 6.22: API-Driven Pagination and useSearchParams Sync

6.22 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! • Production Pitfall Guard (API-Driven Pagination and useSearchParams Sync) • Hands-On Verification (API-Driven Pagination and useSearchParams Sync) • API-Driven Pagination and useSearchParams Sync — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for API-Driven Pagination • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

Purpose: Sharing `/employees?dept=Finance&page=3` over Slack so a colleague opens the exact same filtered page.

What Topic Is Next → Next Topic 6.23: Debounced Search, Multi-Faceted Filtering & Multipart File Uploads

6.23 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 • Production Pitfall Guard (Debounced Search, Multi-Faceted Filtering & Multipart File Uploads) • Hands-On Verification (Debounced Search, Multi-Faceted Filtering & Multipart File Uploads) • Debounced Search, Multi-Faceted Filtering & Multipart File Uploads — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 6.24: Authentication Flows, JWT Storage Security & Vite Environment Variables

6.24 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!) • Production Pitfall Guard (Authentication Flows, JWT Storage Security & Vite

Environment Variables) • Hands-On Verification (Authentication Flows, JWT Storage Security & Vite Environment Variables) • Authentication Flows, JWT Storage Security & Vite Environment Variables — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 6 Capstone (Project 3: React & TypeScript Employee Management Dashboard) → Next Topic 7.1: React Rendering Phases (Render vs Commit)

Module 6 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 7 OF 10 • STAGE 3 • 11 LIVE + LAB HOURS • 21 TOPICS (168 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: 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.

7.1 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` • Production Pitfall Guard (React Rendering Phases (Render vs Commit)) • Hands-On Verification (React Rendering Phases (Render vs Commit)) • React Rendering Phases (Render vs Commit) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for React Rendering Phases (Render vs Commit) • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

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 7.2: Virtual DOM Diffing and Fiber Reconciliation

7.2 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 • Production Pitfall Guard (Virtual DOM Diffing and Fiber Reconciliation) • Hands-On Verification (Virtual DOM Diffing and Fiber Reconciliation) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.3: State Taxonomy: Local, Shared, Global, URL and Server State

7.3 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 • Production Pitfall Guard (State Taxonomy: Local, Shared, Global, URL and Server State) • Hands-On Verification (State Taxonomy: Local, Shared, Global, URL and Server State) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.4: React Context API (`createContext` and `useContext`)

7.4 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 • Production Pitfall Guard (React Context API (`createContext` and `useContext`)) • Hands-On Verification (React Context API (`createContext` and `useContext`)) • React Context API (`createContext` and `useContext`) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.5: Eliminating Prop Drilling with Composition and Context

7.5 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 • Production Pitfall Guard (Eliminating Prop Drilling with Composition and Context) • Hands-On Verification (Eliminating Prop Drilling with Composition and Context) • Eliminating Prop Drilling with Composition and Context — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Eliminating Prop Drilling with Composition • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

Purpose: Passing `<UserMenu />` as `children` into `<TopHeader>` so intermediate layout wrappers never need `user` props at all.

What Topic Is Next → Next Topic 7.6: Redux Architecture Fundamentals

7.6 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` • Production Pitfall Guard (Redux Architecture Fundamentals) • Hands-On Verification (Redux Architecture Fundamentals) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.7: Redux Toolkit (`@reduxjs/toolkit`) Setup

7.7 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}>` • Production Pitfall Guard (Redux Toolkit (`@reduxjs/toolkit`) Setup) • Hands-On Verification (Redux Toolkit (`@reduxjs/toolkit`) Setup) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.8: Redux Store Configuration and Reducer Composition

7.8 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: { ... } })` • Production Pitfall Guard (Redux Store Configuration and Reducer Composition) • Hands-On Verification (Redux Store Configuration and Reducer Composition) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.9: Creating Feature Slices with `createSlice`

7.9 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 • Production Pitfall Guard (Creating Feature Slices with `createSlice`) • Hands-On Verification (Creating Feature Slices with `createSlice`) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.10: Actions, Action Creators and Pure Reducers

7.10 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 • Production Pitfall Guard (Actions, Action Creators and Pure Reducers) • Hands-On Verification (Actions, Action Creators and Pure Reducers) • Actions, Action Creators and Pure Reducers — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.11: Memoized Selectors (`createSelector` / Reselect)

7.11 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 • Production Pitfall Guard (Memoized Selectors (`createSelector` / Reselect)) • Hands-On Verification (Memoized Selectors (`createSelector` / Reselect)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.12: Async Redux Workflows (`createAsyncThunk`)

7.12 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` • Production Pitfall Guard (Async Redux Workflows (`createAsyncThunk`)) • Hands-On Verification (Async Redux Workflows (`createAsyncThunk`)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.13: Redux Middleware and Listener Middleware

7.13 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 • Production Pitfall Guard (Redux Middleware and Listener Middleware) • Hands-On Verification (Redux Middleware and Listener Middleware) • Redux Middleware and Listener Middleware — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.14: RTK Query Fundamentals (`createApi` and `fetchBaseQuery`)

7.14 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 • Production Pitfall Guard (RTK Query Fundamentals (`createApi` and `fetchBaseQuery`)) • Hands-On Verification (RTK Query Fundamentals (`createApi` and `fetchBaseQuery`)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.15: Server State vs Client State Architecture

7.15 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 • Production Pitfall Guard (Server State vs Client State Architecture) • Hands-On Verification (Server State vs Client State Architecture) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.16: TanStack Query (`@tanstack/react-query`) Fundamentals

7.16 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 • Production Pitfall Guard (TanStack Query (`@tanstack/react-query`) Fundamentals) • Hands-On Verification (TanStack Query (`@tanstack/react-query`) Fundamentals) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.17: Mutations and Cache Invalidation (`useMutation`)

7.17 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'] })` • Production Pitfall Guard (Mutations and Cache Invalidation (`useMutation`)) • Hands-On Verification (Mutations and Cache Invalidation (`useMutation`)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.18: Optimistic UI Updates and Rollback

7.18 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` • Production Pitfall Guard (Optimistic UI Updates and Rollback) • Hands-On Verification (Optimistic UI Updates and Rollback) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.19: Lightweight Client State with Zustand Fundamentals

7.19 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 • Production Pitfall Guard (Lightweight Client State with Zustand Fundamentals) • Hands-On Verification (Lightweight Client State with Zustand Fundamentals) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 7.20: Safe Client State Persistence

7.20 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 • Production Pitfall Guard (Safe Client State Persistence) • Hands-On Verification (Safe Client State Persistence) • Safe Client State Persistence — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Safe Client State Persistence • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

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 7.21: Choosing the Right State Management Architecture

7.21 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 • Production Pitfall Guard (Choosing the Right State Management Architecture) • Hands-On Verification (Choosing the Right State Management Architecture) • Choosing the Right State Management Architecture — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Choosing the Right State Management Architecture • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

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 7 Capstone (Week 13 Lab Build: Redux Toolkit Multi-Slice Cart + TanStack Query Catalog) → Next Topic 8.1: Scalable Component Architecture (Presentational vs Feature Containers)

Module 7 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 8 OF 10 • STAGE 3 • 11 LIVE + LAB HOURS • 23 TOPICS (184 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: 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.

8.1 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`) • Production Pitfall Guard (Scalable Component Architecture (Presentational vs Feature Containers)) • Hands-On Verification (Scalable Component Architecture (Presentational vs Feature Containers)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.2: Feature-Based Folder Architecture

8.2 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 • Production Pitfall Guard (Feature-Based Folder Architecture) • Hands-On Verification (Feature-Based Folder Architecture) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.3: Design Systems and Tokenized UI Foundations

8.3 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 • Production Pitfall Guard (Design Systems and Tokenized UI Foundations) • Hands-On Verification (Design Systems and Tokenized UI Foundations) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.4: Working with Enterprise Component Libraries (Headless UI / Radix Concepts)

8.4 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 • Production Pitfall Guard (Working with Enterprise Component Libraries (Headless UI / Radix Concepts)) • Hands-On Verification (Working with Enterprise Component Libraries (Headless UI / Radix Concepts)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.5: Compound Components Pattern

8.5 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>`) • Production Pitfall Guard (Compound Components Pattern) • Hands-On Verification (Compound Components Pattern) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.6: Higher-Order Components (HOCs) — Legacy & Cross-Cutting Awareness

8.6 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 • Production Pitfall Guard (Higher-Order Components (HOCs) — Legacy & Cross-Cutting Awareness) • Hands-On Verification (Higher-Order Components (HOCs) — Legacy & Cross-Cutting Awareness) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.7: Render Props Pattern and Function-as-Child

8.7 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 • Production Pitfall Guard (Render Props Pattern and Function-as-Child) • Hands-On Verification (Render Props Pattern and Function-as-Child) • Render Props Pattern and Function-as-Child — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Render Props Pattern • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

Purpose: Typing generic `<DataTable<T> data={rows} renderRow={(row) => ...} />` and `<VirtualList items={...} renderItem={...} />` components.

What Topic Is Next → Next Topic 8.8: Advanced Custom Hooks (`useIntersectionObserver`, `useMediaQuery`, `useOnClickOutside`)

8.8 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 • Production Pitfall Guard (Advanced Custom Hooks (`useIntersectionObserver`, `useMediaQuery`, `useOnClickOutside`)) • Hands-On Verification (Advanced Custom Hooks (`useIntersectionObserver`, `useMediaQuery`, `useOnClickOutside`)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.9: Identifying Unnecessary Re-Renders and State Colocation

8.9 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) • Production Pitfall Guard (Identifying Unnecessary Re-Renders and State Colocation) • Hands-On Verification (Identifying Unnecessary Re-Renders and State Colocation) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.10: Component Memoization with `React.memo`

8.10 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` • Production Pitfall Guard (Component Memoization with `React.memo`) • Hands-On Verification (Component Memoization with `React.memo`) • Component Memoization with `React.memo` — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.11: Expensive Computation Caching with `useMemo`

8.11 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 • Production Pitfall Guard (Expensive Computation Caching with `useMemo`) • Hands-On Verification (Expensive Computation Caching with `useMemo`) • Expensive Computation Caching with `useMemo` — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.12: Stable Function References with `useCallback`

8.12 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 • Production Pitfall Guard (Stable Function References with `useCallback`) • Hands-On Verification (Stable Function References with `useCallback`) • Stable Function References with `useCallback` — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Stable Function References with `useCallback` • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

Purpose: Passing a stable `handleToggleSelect = useCallback((id: string) => ..., [])` using functional `setSelected(prev => ...)` to memoized table rows.

What Topic Is Next → Next Topic 8.13: Code Splitting and Dynamic `import()`

8.13 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')` • Production Pitfall Guard (Code Splitting and Dynamic `import()`) • Hands-On Verification (Code Splitting and Dynamic `import()`) • Code Splitting and Dynamic `import()` — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Code Splitting • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

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 8.14: Lazy Component Loading (`React.lazy`) and `<Suspense>`

8.14 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 • Production Pitfall Guard (Lazy Component Loading (`React.lazy`) and `<Suspense>`) • Hands-On Verification (Lazy Component Loading (`React.lazy`) and `<Suspense>`) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.15: Error Boundaries and Graceful Crash Recovery

8.15 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 • Production Pitfall Guard (Error Boundaries and Graceful Crash Recovery) • Hands-On Verification (Error Boundaries and Graceful Crash Recovery) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.16: React Portals (`createPortal`) for Modals, Drawers and Toasts

8.16 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 • Production Pitfall Guard (React Portals (`createPortal`) for Modals, Drawers and Toasts) • Hands-On Verification (React Portals (`createPortal`) for Modals, Drawers and Toasts) • React Portals (`createPortal`) for Modals, Drawers and Toasts — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.17: List Virtualization / Windowing (`@tanstack/react-virtual` Concepts)

8.17 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 • Production Pitfall Guard (List Virtualization / Windowing (`@tanstack/react-virtual` Concepts)) • Hands-On Verification (List Virtualization / Windowing (`@tanstack/react-virtual` Concepts)) • List Virtualization / Windowing (`@tanstack/react-virtual` Concepts) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.18: Debouncing High-Frequency User Input

8.18 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 • Production Pitfall Guard (Debouncing High-Frequency User Input) • Hands-On Verification (Debouncing High-Frequency User Input) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.19: Throttling Continuous Scroll, Resize and Pointer Events

8.19 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 • Production Pitfall Guard (Throttling Continuous Scroll, Resize and Pointer Events) • Hands-On Verification (Throttling Continuous Scroll, Resize and Pointer Events) • Throttling Continuous Scroll, Resize and Pointer Events — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Throttling Continuous Scroll • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

Purpose: Updating sticky scroll progress bars, window resize charts, or drag coordinates at steady 16ms/100ms intervals.

What Topic Is Next → Next Topic 8.20: Infinite Scrolling with `IntersectionObserver`

8.20 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 • Production Pitfall Guard (Infinite Scrolling with `IntersectionObserver`) • Hands-On Verification (Infinite Scrolling with `IntersectionObserver`) • Infinite Scrolling with `IntersectionObserver` — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Infinite Scrolling with `IntersectionObserver` • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

Purpose: Prefetching the next 20 products seamlessly 200px before the shopper reaches the bottom of the mobile feed.

What Topic Is Next → Next Topic 8.21: Client and API Search Optimization (Autocomplete, Highlighting & Caching)

8.21 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 • Production Pitfall Guard (Client and API Search Optimization (Autocomplete, Highlighting & Caching)) • Hands-On Verification (Client and API Search Optimization (Autocomplete, Highlighting & Caching)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.22: File Upload, Image Preview and Client Download Handling

8.22 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 • Production Pitfall Guard (File Upload, Image Preview and Client Download Handling) • Hands-On Verification (File Upload, Image Preview and Client Download Handling) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 8.23: Building Accessible Semantic Reusable UI Components

8.23 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 • Production Pitfall Guard (Building Accessible Semantic Reusable UI Components) • Hands-On Verification (Building Accessible Semantic Reusable UI Components) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler

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 8 Capstone (Week 14 Lab Build: Accessible Design-System Primitives & Virtualized Search Grid) → Next Topic 9.1: Browser and Frontend Security Threat Model

Module 8 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 9 OF 10 • STAGE 3 • 11 LIVE + LAB HOURS • 23 TOPICS (184 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: 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.

9.1 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 • Production Pitfall Guard (Browser and Frontend Security Threat Model) • Hands-On Verification (Browser and Frontend Security Threat Model) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.2: Cross-Site Scripting (XSS) Prevention in React

9.2 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 • Production Pitfall Guard (Cross-Site Scripting (XSS) Prevention in React) • Hands-On Verification (Cross-Site Scripting (XSS) Prevention in React) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.3: Cross-Site Request Forgery (CSRF) Fundamentals

9.3 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 • Production Pitfall Guard (Cross-Site Request Forgery (CSRF) Fundamentals) • Hands-On Verification (Cross-Site Request Forgery (CSRF) Fundamentals) • Cross-Site Request Forgery (CSRF) Fundamentals — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.4: Cross-Origin Resource Sharing (CORS) Deep Dive

9.4 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` • Production Pitfall Guard (Cross-Origin Resource Sharing (CORS) Deep Dive) • Hands-On Verification (Cross-Origin Resource Sharing (CORS) Deep Dive) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.5: Content Security Policy (CSP) and Security Headers

9.5 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'`) • Production Pitfall Guard (Content Security Policy (CSP) and Security Headers) • Hands-On Verification (Content Security Policy (CSP) and Security Headers) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.6: Secure Authentication and Token Storage Strategies

9.6 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 • Production Pitfall Guard (Secure Authentication and Token Storage Strategies) • Hands-On Verification (Secure Authentication and Token Storage Strategies) • Secure Authentication and Token Storage Strategies — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.7: Preventing Exposed Secrets in Frontend Bundles

9.7 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 • Production Pitfall Guard (Preventing Exposed Secrets in Frontend Bundles) • Hands-On Verification (Preventing Exposed Secrets in Frontend Bundles) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.8: Client-Side Input Validation vs Server Validation

9.8 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 • Production Pitfall Guard (Client-Side Input Validation vs Server Validation) • Hands-On Verification (Client-Side Input Validation vs Server Validation) • Client-Side Input Validation vs Server Validation — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.9: Measuring Web Page Performance (FCP, TTFB, TBT & Waterfall)

9.9 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) • Production Pitfall Guard (Measuring Web Page Performance (FCP, TTFB, TBT & Waterfall)) • Hands-On Verification (Measuring Web Page Performance (FCP, TTFB, TBT & Waterfall)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.10: Core Web Vitals: LCP, INP and CLS

9.10 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 • Production Pitfall Guard (Core Web Vitals: LCP, INP and CLS) • Hands-On Verification (Core Web Vitals: LCP, INP and CLS) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.11: Modern Image Optimization (`WebP`/`AVIF`, `srcset` & `<picture>`)

9.11 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 • Production Pitfall Guard (Modern Image Optimization (`WebP`/`AVIF`, `srcset` & `<picture>`)) • Hands-On Verification (Modern Image Optimization (`WebP`/`AVIF`, `srcset` & `<picture>`)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.12: Web Font Optimization and Preventing FOIT/CLS

9.12 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 • Production Pitfall Guard (Web Font Optimization and Preventing FOIT/CLS) • Hands-On Verification (Web Font Optimization and Preventing FOIT/CLS) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.13: Bundle Size Optimization and Tree-Shaking

9.13 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` • Production Pitfall Guard (Bundle Size Optimization and Tree-Shaking) • Hands-On Verification (Bundle Size Optimization and Tree-Shaking) • Bundle Size Optimization and Tree-Shaking — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Bundle Size Optimization • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

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 9.14: Code Splitting Initial JavaScript Payload

9.14 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`) • Production Pitfall Guard (Code Splitting Initial JavaScript Payload) • Hands-On Verification (Code Splitting Initial JavaScript Payload) • Code Splitting Initial JavaScript Payload — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.15: Browser Caching Strategies (`Cache-Control` & Content Hashing)

9.15 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` • Production Pitfall Guard (Browser Caching Strategies (`Cache-Control` & Content Hashing)) • Hands-On Verification (Browser Caching Strategies (`Cache-Control` & Content Hashing)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.16: Chrome DevTools Performance, Memory & Coverage Profiling

9.16 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) • Production Pitfall Guard (Chrome DevTools Performance, Memory & Coverage Profiling) • Hands-On Verification (Chrome DevTools Performance, Memory & Coverage Profiling) • Chrome DevTools Performance, Memory & Coverage Profiling — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.17: Lighthouse Audits (Performance, Accessibility, Best Practices, SEO)

9.17 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 • Production Pitfall Guard (Lighthouse Audits (Performance, Accessibility, Best Practices, SEO)) • Hands-On Verification (Lighthouse Audits (Performance, Accessibility, Best Practices, SEO)) • Lighthouse Audits (Performance, Accessibility, Best Practices, SEO) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Lighthouse Audits (Performance, Accessibility, Best... • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

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 9.18: Cross-Browser and Multi-Device Compatibility Testing

9.18 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) • Production Pitfall Guard (Cross-Browser and Multi-Device Compatibility Testing) • Hands-On Verification (Cross-Browser and Multi-Device Compatibility Testing) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.19: Unit Testing Fundamentals for Frontend Logic

9.19 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 • Production Pitfall Guard (Unit Testing Fundamentals for Frontend Logic) • Hands-On Verification (Unit Testing Fundamentals for Frontend Logic) • Unit Testing Fundamentals for Frontend Logic — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.20: Vitest / Jest Setup, Matchers and Lifecycle

9.20 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` • Production Pitfall Guard (Vitest / Jest Setup, Matchers and Lifecycle) • Hands-On Verification (Vitest / Jest Setup, Matchers and Lifecycle) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.21: Component Testing with React Testing Library (RTL)

9.21 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) • Production Pitfall Guard (Component Testing with React Testing Library (RTL)) • Hands-On Verification (Component Testing with React Testing Library (RTL)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.22: Simulating Realistic User Interactions (`@testing-library/user-event`)

9.22 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}')` • Production Pitfall Guard (Simulating Realistic User Interactions (`@testing-library/user-event`)) • Hands-On Verification (Simulating Realistic User Interactions (`@testing-library/user-event`)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9.23: Mocking REST APIs (`MSW` / `vi.fn`), Coverage & Playwright E2E Intro

9.23 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 • Production Pitfall Guard (Mocking REST APIs (`MSW` / `vi.fn`), Coverage & Playwright E2E Intro) • Hands-On Verification (Mocking REST APIs (`MSW` / `vi.fn`), Coverage & Playwright E2E Intro) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 9 Capstone (Week 15 Lab Build: Lighthouse 90+ Hardened App & Vitest + RTL Test Suite) → Next Topic 10.1: Next.js Architecture: CSR vs SSR vs SSG vs ISR

Module 9 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 10 OF 10 • STAGE 3 • 12 LIVE + LAB HOURS • 19 TOPICS (152 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Next.js App Router Basics, Production CI/CD, Vercel Deployment & E-Commerce Capstone

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.

10.1 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) • Production Pitfall Guard (Next.js Architecture: CSR vs SSR vs SSG vs ISR) • Hands-On Verification (Next.js Architecture: CSR vs SSR vs SSG vs ISR) • Next.js Architecture: CSR vs SSR vs SSG vs ISR — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Next.js Architecture • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

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 10.2: Next.js Project Setup (`create-next-app` with TypeScript & Tailwind)

10.2 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 • Production Pitfall Guard (Next.js Project Setup (`create-next-app` with TypeScript & Tailwind)) • Hands-On Verification (Next.js Project Setup (`create-next-app` with TypeScript & Tailwind)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10.3: File-System Routing with the Next.js App Router

10.3 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 • Production Pitfall Guard (File-System Routing with the Next.js App Router) • Hands-On Verification (File-System Routing with the Next.js App Router) • File-System Routing with the Next.js App Router — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10.4: Pages and Persistent Nested Layouts (`page.tsx` & `layout.tsx`)

10.4 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 • Production Pitfall Guard (Pages and Persistent Nested Layouts (`page.tsx` & `layout.tsx`)) • Hands-On Verification (Pages and Persistent Nested Layouts (`page.tsx` & `layout.tsx`)) • Pages and Persistent Nested Layouts (`page.tsx` & `layout.tsx`) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10.5: Dynamic Routes and Typed Route Params (`[slug]` & `[...catchAll]`)

10.5 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`) • Production Pitfall Guard (Dynamic Routes and Typed Route Params (`[slug]` & `[...catchAll]`)) • Hands-On Verification (Dynamic Routes and Typed Route Params (`[slug]` & `[...catchAll]`)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10.6: Navigation and Prefetching with `next/link` and `useRouter`

10.6 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 • Production Pitfall Guard (Navigation and Prefetching with `next/link` and `useRouter`) • Hands-On Verification (Navigation and Prefetching with `next/link` and `useRouter`) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Navigation

Purpose: Highlighting active navigation links via `usePathname()` inside a lightweight `<NavLinks>` Client Component.

What Topic Is Next → Next Topic 10.7: React Server Components (RSC) Fundamentals

10.7 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 • Production Pitfall Guard (React Server Components (RSC) Fundamentals) • Hands-On Verification (React Server Components (RSC) Fundamentals) • React Server Components (RSC) Fundamentals — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10.8: Client Components (`'use client'`) for Interactivity

10.8 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 • Production Pitfall Guard (Client Components (`'use client'`) for Interactivity) • Hands-On Verification (Client Components (`'use client'`) for Interactivity) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10.9: Server vs Client Component Composition Boundaries

10.9 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 • Production Pitfall Guard (Server vs Client Component Composition Boundaries) • Hands-On Verification (Server vs Client Component Composition Boundaries) • Server vs Client Component Composition Boundaries — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10.10: Server Data Fetching vs Client Interactive Fetching

10.10 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 • Production Pitfall Guard (Server Data Fetching vs Client Interactive Fetching) • Hands-On Verification (Server Data Fetching vs Client Interactive Fetching) • Server Data Fetching vs Client Interactive Fetching — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Server Data Fetching vs Client Interactive Fetching • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

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 10.11: Streaming Loading UI (`loading.tsx` and `<Suspense>`)

10.11 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 • Production Pitfall Guard (Streaming Loading UI (`loading.tsx` and `<Suspense>`)) • Hands-On Verification (Streaming Loading UI (`loading.tsx` and `<Suspense>`)) • Streaming Loading UI (`loading.tsx` and `<Suspense>`) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Streaming Loading UI (`loading.tsx` and `<Suspense>`... • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection

Purpose: Rendering product title/price immediately while slower customer reviews stream in inside their own `<Suspense>` boundary.

What Topic Is Next → Next Topic 10.12: Route Error Handling (`error.tsx` and `global-error.tsx`)

10.12 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 • Production Pitfall Guard (Route Error Handling (`error.tsx` and `global-error.tsx`)) • Hands-On Verification (Route Error Handling (`error.tsx` and `global-error.tsx`)) • Route Error Handling (`error.tsx` and `global-error.tsx`) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10.13: Static and Dynamic Metadata for Technical SEO

10.13 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` • Production Pitfall Guard (Static and Dynamic Metadata for Technical SEO) • Hands-On Verification (Static and Dynamic Metadata for Technical SEO) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10.14: Frontend Production Build Pipeline (`tsc -b && vite build`)

10.14 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/*`) • Production Pitfall Guard (Frontend Production Build Pipeline (`tsc -b && vite build`)) • Hands-On Verification (Frontend Production Build Pipeline (`tsc -b && vite build`)) • Frontend Production Build Pipeline (`tsc -b && vite build`) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler Execution & Memory Lifecycle for Frontend Production Build Pipeline (`tsc -b && vite... • Production Hardening & Security: DOM XSS escaping, AbortController race guards & RBAC route protection • Diagnostics, Profiling & Telemetry: React DevTools Flamegraph Profiler & Vite bundle analyzer

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 10.15: Vite Build Configuration (`vite.config.ts`, Aliases, Chunks & Sourcemaps)

10.15 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` • Production Pitfall Guard (Vite Build Configuration (`vite.config.ts`, Aliases, Chunks & Sourcemaps)) • Hands-On Verification (Vite Build Configuration (`vite.config.ts`, Aliases, Chunks & Sourcemaps)) • Vite Build Configuration (`vite.config.ts`, Aliases, Chunks & Sourcemaps) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10.16: Deploying Frontend Applications to Vercel & Netlify (with SPA Rewrites)

10.16 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` • Production Pitfall Guard (Deploying Frontend Applications to Vercel & Netlify (with SPA Rewrites)) • Hands-On Verification (Deploying Frontend Applications to Vercel & Netlify (with SPA Rewrites)) • Deploying Frontend Applications to Vercel & Netlify (with SPA Rewrites) — Interview Readiness • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10.17: Custom Domains, DNS (`A` / `CNAME`) and Automatic HTTPS/TLS

10.17 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 • Production Pitfall Guard (Custom Domains, DNS (`A` / `CNAME`) and Automatic HTTPS/TLS) • Hands-On Verification (Custom Domains, DNS (`A` / `CNAME`) and Automatic HTTPS/TLS) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10.18: Managing Production, Preview and Local Environment Variables

10.18 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 • Production Pitfall Guard (Managing Production, Preview and Local Environment Variables) • Hands-On Verification (Managing Production, Preview and Local Environment Variables) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10.19: GitHub Actions CI/CD Pipeline & Production Error Monitoring (Sentry Basics)

10.19 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` • Production Pitfall Guard (GitHub Actions CI/CD Pipeline & Production Error Monitoring (Sentry Basics)) • Hands-On Verification (GitHub Actions CI/CD Pipeline & Production Error Monitoring (Sentry Basics)) • Idiomatic REAC Implementation: pure functional hooks, TanStack Query server cache & route code-splitting • React 19 Fiber Reconciler & Concurrent Render Scheduler 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 10 Capstone → End-of-Course Live Client Project: Complete React.js Developer Course (with HTML5, CSS3, Tailwind, JS & TypeScript) — Enterprise Client Production Platform

Module 10 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...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: Complete React.js Developer Course (with HTML5, CSS3, Tailwind, JS & TypeScript) — Enterprise Client Production Platform • **Confirmed Role:** Complete React.js Developer Course (with HTML5, CSS3, Tailwind, JS & TypeScript) Engineering Project Intern **Domain & Production Infra:** Live Enterprise Frontend & React Ecosystem Client Engagement (HTML5 & Semantic A11y, CSS3, Flexbox & Grid, Tailwind CSS, JavaScript ES6+ & Async/DOM, TypeScript 5 (Strict) • Git CI/CD + Automated Testing + Live HTTPS Cloud Production)

- Translated live client functional and non-functional requirements into modular HTML5 & Semantic A11y, CSS3, Flexbox & Grid, Tailwind CSS, JavaScript ES6+ & Async/DOM, TypeScript 5 (Strict) 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: Complete React.js Developer Course (with HTML5, CSS3, Tailwind, JS & TypeScript) Developer, Associate Frontend & React Ecosystem Engineer, Software Development Engineer (SDE-1)

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹4.5 – ₹8.5 LPA • $55K–$85K Global **Mid-Level Product Engineer (1–3 Yrs):** ₹9.5 – ₹19.0 LPA • $85K–$130K Global **Senior / Multi-Stack Specialist (3–6+ Yrs):** ₹20.0 – ₹38.0+ LPA • $130K–$185K+ Global

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 95400 07917 • Web: https://mockattempt.com/courses/reactjs-complete-developer-training

Try 2 Free Topic-Wise Timed Topic Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.