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

ANGU • ENTERPRISE FRONTEND & TYPESCRIPT SPA **BEGINNER TO ENTERPRISE ANGULAR ENGINEER**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript)

Master enterprise frontend development from scratch: includes built-in HTML5, CSS3/SCSS, JavaScript ES6+, and Strict TypeScript 5 foundations followed by modern Standalone Angular, Signals, Services/DI, Router Guards, HttpClient Interceptors, Reactive Forms, RxJS, NgRx, Angular Material, and TestBed testing.

Duration: 10 Weeks • Weekend Live (Sat–Sun • 2h/Day) + Coding Lab (111 Hrs) • **Modules:** 10 • **Live Topics:** 179 • **Subtopics:** 1,432+ • **Labs & Tasks:** 358 • **Mocks / AI Vivas:** 179 / 179 • **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 (TypeScript 5, JavaScript ES6+ & Angular RxJS): Live trainer support to crack LeetCode & HackerRank problems, optimize complexity, and handle edge cases.**

Core Technologies & Engineering Stack: HTML5 & Accessible Forms • CSS3, Flexbox, Grid & SCSS • JavaScript ES6+ & Async/DOM • TypeScript 5 & Decorators • Angular (Standalone + Signals) • Services, DI & Router Guards • HttpClient & Interceptors • Typed Reactive Forms & FormArray • RxJS Operators & AsyncPipe • NgRx Store & Signal Store • Angular Material 3 & CDK • Angular TestBed & Deployment

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Author semantic HTML5, responsive CSS3/SCSS layouts, clean ES6+ async logic, and strict TypeScript 5 classes/interfaces/generics.
2. Build modular Standalone Angular applications using Signals (`signal`/`computed`/`effect`), `@if/@for/@switch`, custom Pipes, and `OnPush` change detection.
3. Implement `@Injectable` services, Hierarchical DI, lazy-loaded routes with `CanActivateFn` guards, and `HttpClient` with functional JWT/error interceptors.
4. Master Strictly Typed Reactive Forms (`FormArray`), RxJS flattening/combination streams (`switchMap`, `exhaustMap`, `combineLatest`), NgRx, Angular Material 3 (`MatTable`, `MatDialog`), and `TestBed` unit specs.
5. Crack LeetCode and HackerRank problems in TypeScript 5, JavaScript ES6+ & Angular RxJS with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
6. Automated Testing, Debugging & Quality Gates (HTML5 & Accessible Forms, CSS3, Flexbox, Grid & SCSS, JavaScript ES6+ & Async/DOM, TypeScript 5 & Decorators, Angular (Standalone + Signals)) — Automated HTML5 & Accessible Forms, CSS3, Flexbox, Grid & SCSS, JavaScript ES6+ & Async/DOM, TypeScript 5 & Decorators, Angular (Standalone + Signals) 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 Angular Developer Course (with HTML5, CSS3/SCSS, 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 & Accessible Forms, CSS3, Flexbox, Grid & SCSS, JavaScript ES6+ & Async/DOM, TypeScript 5 & Decorators, Angular (Standalone + Signals) 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 Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript).

Eligibility & Prerequisites: No prior HTML, CSS, JavaScript, TypeScript, or Angular experience required — Stage 1 (Modules 1–3) builds every foundation step-by-step! • 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 • 179 LIVE TOPICS • 1,432+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–3: Month 1 – Stage 1: Built-In HTML5, CSS3/SCSS, JavaScript ES6+ & Strict TypeScript Foundations (Weeks 1–3) (77 Topics / 616 Subtopics) • Modules 4–6: Month 2 – Stage 2: Standalone Angular, Signals, Dependency Injection, Router & HttpClient (Weeks 4–6) (45 Topics / 360 Subtopics) • Modules 7–10: Month 3 – Stage 3: Reactive Forms, RxJS, NgRx, Angular Material, Testing & Admin Capstone (Weeks 7–10) (57 Topics / 456 Subtopics)

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

Module 1: Semantic HTML5, Accessible Forms, CSS3 Flexbox/Grid & Sass/SCSS Foundations

Establish clean enterprise markup and stylesheet engineering for Angular templates: semantic HTML5 landmarks, accessible tables & forms, CSS3 cascade/specificity, Flexbox, CSS Grid, responsive breakpoints, and modular Sass/SCSS.

Objectives: ✓ Write semantic, accessible HTML5 templates and forms ready for Angular component bindings. • ✓ Design responsive enterprise layouts using Flexbox, CSS Grid, CSS custom properties, and modular `.scss` architecture.

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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Browser Rendering Architecture (DOM and CSSOM)

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

What Topic Is Next → Next Topic 1.2: 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for HTML5 Introduction

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

What Topic Is Next → Next Topic 1.3: 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for HTML Document Structure

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

What Topic Is Next → Next Topic 1.4: 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for HTML Elements (Block, Inline and Void Elements)

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

What Topic Is Next → Next Topic 1.5: 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Text Formatting

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

What Topic Is Next → Next Topic 1.6: 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Links

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

What Topic Is Next → Next Topic 1.7: 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Images

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

What Topic Is Next → Next Topic 1.8: 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Lists (Ordered, Unordered and Description Lists)

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

What Topic Is Next → Next Topic 1.9: 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Accessible HTML Tables

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

What Topic Is Next → Next Topic 1.10: Pseudo-Classes (:hover, :focus-visible, :active, :nth-child, :is, :where, :has)

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

Subtopics Covered: Interactive `:hover` • Keyboard-only `:focus-visible` (superior to `:focus`!) • Structural `:nth-child(even)` • Zero-specificity `:where()` • Production Pitfall Guard (Pseudo-Classes (:hover, :focus-visible, :active, :nth-child, :is, :where, :has)) • Hands-On Verification (Pseudo-Classes (:hover, :focus-visible, :active, :nth-child, :is, :where, :has)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Pseudo-Classes (:hover, :focus-visible, :active, :nt...

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

What Topic Is Next → Next Topic 1.11: Pseudo-Elements (::before, ::after, ::placeholder, ::selection)

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

Subtopics Covered: Decorative `::before`/`::after` (`content: ""`) • Custom badges/dividers • `::first-letter` • `::placeholder` • Production Pitfall Guard (Pseudo-Elements (::before, ::after, ::placeholder, ::selection)) • Hands-On Verification (Pseudo-Elements (::before, ::after, ::placeholder, ::selection)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Pseudo-Elements (::before, ::after, ::placeholder, :...

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

What Topic Is Next → Next Topic 1.12: The Cascade, Inheritance and Specificity

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

Subtopics Covered: Origin & layer order • `(Inline • Class/Attr/Pseudo-class • Element/Pseudo-element)` tuple scoring • Production Pitfall Guard (The Cascade, Inheritance and Specificity) • Hands-On Verification (The Cascade, Inheritance and Specificity) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for The Cascade

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

What Topic Is Next → Next Topic 1.13: CSS Units (px, %, em, rem, vh, dvh, vw, ch)

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

Subtopics Covered: `rem` (root-relative — respects user browser font-size accessibility zoom!) • `em` (compound component scaling) • `ch` (`60ch–70ch` optimal reading measure!) • `100dvh` mobile dynamic viewport height • Production Pitfall Guard (CSS Units (px, %, em, rem, vh, dvh, vw, ch)) • Hands-On Verification (CSS Units (px, %, em, rem, vh, dvh, vw, ch)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for CSS Units (px, %, em, rem, vh, dvh, vw, ch)

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

What Topic Is Next → Next Topic 1.14: CSS Color Systems (HEX, RGB/RGBA, HSL and Modern Color Syntax)

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

Subtopics Covered: HEX (`#0f172a`) • Modern space-separated `rgb(15 23 42 / 0.8)` • Intuitive `hsl(222 47% 11%)` hue/saturation/lightness palettes • WCAG 4.5:1 text contrast ratio • Production Pitfall Guard (CSS Color Systems (HEX, RGB/RGBA, HSL and Modern Color Syntax)) • Hands-On Verification (CSS Color Systems (HEX, RGB/RGBA, HSL and Modern Color Syntax)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for CSS Color Systems (HEX, RGB/RGBA, HSL and Modern Col...

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

What Topic Is Next → Next Topic 1.15: Backgrounds and Linear/Radial Gradients

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

Subtopics Covered: `background-color` • `background-image` • `linear-gradient` / `radial-gradient` • `background-size: cover | contain` • Production Pitfall Guard (Backgrounds and Linear/Radial Gradients) • Hands-On Verification (Backgrounds and Linear/Radial Gradients) • Idiomatic ANGU Implementation: clean modular

architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Backgrounds

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

What Topic Is Next → Next Topic 1.16: Flexbox Container and Flex Items

1.16 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Flexbox Container

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

What Topic Is Next → Next Topic 1.17: Flex Direction (row, column and reverse)

1.17 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Flex Direction (row, column and reverse)

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

What Topic Is Next → Next Topic 1.18: Flexbox Alignment (justify-content, align-items, align-self)

1.18 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Flexbox Alignment (justify-content, align-items, ali...

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

What Topic Is Next → Next Topic 1.19: Flex Wrapping (flex-wrap and align-content)

1.19 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Flex Wrapping (flex-wrap and align-content)

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

What Topic Is Next → Next Topic 1.20: Flex Sizing (flex-grow, flex-shrink, flex-basis & min-width: 0)

1.20 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Flex Sizing (flex-grow, flex-shrink, flex-basis & mi...

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

What Topic Is Next → Next Topic 1.21: CSS Grid Container and Grid Items

1.21 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for CSS Grid Container

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

What Topic Is Next → Next Topic 1.22: Grid Structure (Rows, Columns, fr Unit and Gaps)

1.22 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Grid Structure (Rows, Columns, fr Unit and Gaps)

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

What Topic Is Next → Next Topic 1.23: Grid Templates (grid-template-columns, rows and grid-auto-rows)

1.23 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Grid Templates (grid-template-columns, rows and

grid...

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

What Topic Is Next → Next Topic 1.24: Grid Placement (Line Numbers, Spans and Named grid-template-areas)

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

Subtopics Covered: `grid-column: 1 / -1` full-bleed span • `span 2` • Visual ASCII-map `grid-template-areas` (`"header header" "sidebar main" "footer footer"`) • Production Pitfall Guard (Grid Placement (Line Numbers, Spans and Named grid-template-areas)) • Hands-On Verification (Grid Placement (Line Numbers, Spans and Named grid-template-areas)) • Grid Placement (Line Numbers, Spans and Named grid-template-areas) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Grid Placement (Line Numbers, Spans and Named grid-t...

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

What Topic Is Next → Next Topic 1.25: Intrinsic Grid Functions (repeat, minmax, auto-fit vs auto-fill)

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

Subtopics Covered: Zero-media-query responsive grid `repeat(auto-fit • Minmax(min(100% • 17rem) • 1fr))` • Production Pitfall Guard (Intrinsic Grid Functions (repeat, minmax, auto-fit vs auto-fill)) • Hands-On Verification (Intrinsic Grid Functions (repeat, minmax, auto-fit vs auto-fill)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Intrinsic Grid Functions (repeat, minmax, auto-fit v...

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

What Topic Is Next → Module 1 Capstone (Week 3 Lab Build: Mobile-First Landing Page & Sass/Tailwind Component Kit) → Next Topic 2.1: Array Traversal (for, for...of, entries and forEach)

Module 1 Hands-On Capstone Project (100 Marks): Week 3 Lab Build: Mobile-First Landing Page & Sass/Tailwind Component Kit — Translate a Figma design spec into a fluid mobile-first landing page (`360px` to `1440px+`) alongside reusable Bootstrap 5 and Tailwind CSS utility components.

MODULE 2 OF 10 • STAGE 1 • 11 LIVE + LAB HOURS • 25 TOPICS (200 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Modern JavaScript (ES6+), Closures, `this`, Event Loop, Promises & DOM Foundations

Master modern JavaScript ES6+ fundamentals essential for TypeScript and RxJS: immutable array/object operations, ES modules, closures, `this` binding & prototypes, Event Loop microtasks, Promises, `async/await`, `fetch`, and DOM events.

Objectives: ✓ Work fluently with ES6+ array methods, objects, `Map`/`Set`, destructuring, spread, and ES modules. • ✓ Explain closures, `this` context, browser Event Loop execution order, Promises, `async/await`, and `LocalStorage`.

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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Array Traversal (for, for...of, entries and forEach)

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Array.prototype.map

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Array.prototype.filter

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Array.prototype.reduce

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Array.prototype.find

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Array.prototype.some • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Array.prototype.every • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Array.prototype.sort

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Array.prototype.slice (Non-Mutating Subarrays)

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Array.prototype.splice

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

What Topic Is Next → Next Topic 2.11: Execution Contexts (Global and Function)

2.11 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Execution Contexts (Global and Function)

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

What Topic Is Next → Next Topic 2.12: The JavaScript Call Stack

2.12 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for The JavaScript Call Stack

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

What Topic Is Next → Next Topic 2.13: Lexical Environment and the Scope Chain

2.13 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Lexical Environment • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

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

What Topic Is Next → Next Topic 2.14: Closures (Mechanism, Practical Uses and Memory Nuances)

2.14 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Closures (Mechanism, Practical Uses and Memory Nuanc...

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

What Topic Is Next → Next Topic 2.15: The this Keyword (4 Binding Rules)

2.15 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for The this Keyword (4 Binding Rules)

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

What Topic Is Next → Next Topic 2.16: Explicit Context Control (call, apply and bind)

2.16 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Explicit Context Control (call, apply and bind)

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

What Topic Is Next → Next Topic 2.17: Prototypes and the [[Prototype]] Chain

2.17 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Prototypes

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

What Topic Is Next → Next Topic 2.18: Prototypal Inheritance (Object.create and Class Sugar)

2.18 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Prototypal Inheritance (Object.create and Class Suga...

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

What Topic Is Next → Next Topic 2.19: Browser Event Loop Architecture

2.19 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Browser Event Loop Architecture

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

What Topic Is Next → Next Topic 2.20: Microtask Queue (Promises & queueMicrotask)

2.20 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Microtask Queue (Promises & queueMicrotask)

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

What Topic Is Next → Next Topic 2.21: Document Object Model (DOM) Tree Architecture

2.21 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Document Object Model (DOM) Tree Architecture

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

What Topic Is Next → Next Topic 2.22: DOM Selection (getElementById, querySelector, querySelectorAll, closest)

2.22 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for DOM Selection (getElementById, querySelector, queryS...

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

What Topic Is Next → Next Topic 2.23: DOM Manipulation (createElement, textContent, append, replaceChildren, remove)

2.23 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for DOM Manipulation (createElement, textContent, append...

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

What Topic Is Next → Next Topic 2.24: HTML Attributes and dataset Manipulation

2.24 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for HTML Attributes

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

What Topic Is Next → Next Topic 2.25: CSS Class Manipulation with classList

2.25 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for CSS Class Manipulation with classList

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

What Topic Is Next → 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for TypeScript Introduction (JavaScript vs TypeScript)

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for TypeScript Environment Setup (Node.js, npm & Vite)

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for tsconfig.json

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Basic Primitive Types (string, number, boolean, null...

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Special Types (any, unknown, never and void)

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Typed Arrays • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Tuples

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Enums vs Const String Literal Unions

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Type Inference

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer

Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Explicit Type Annotations (When Required vs Redundan...

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Typed Functions

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Object Types

Purpose: Marking `readonly id: string` and `readonly createdAt: string` on domain models so UI code never overwrites identifiers.

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Interfaces

Purpose: Extending `ImportMetaEnv` in `vite-env.d.ts` (`readonly VITE_API_BASE_URL: string`) and composing React component `Props` interfaces.

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Type Aliases (type) vs Interfaces

Purpose: Using `interface` for object/prop shapes and `type` for unions, state machines, and utility transformations.

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Union Types (|) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Literal Types • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Type Narrowing (typeof, instanceof, in and Custom Ty...

Purpose: Filtering nullable arrays cleanly via `.filter((x): x is Product => x !== null)`!

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Type Assertions (as) vs the satisfies Operator

Purpose: Validating a route/theme config map with `satisfies Record<RouteKey, RouteMeta>` while keeping exact literal types intact!

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Generics (Functions, Interfaces and Classes)

Purpose: Writing ONE generic `fetchJson<T>(url)` and `PaginatedResponse<T>` shared across Employees, Products, and Orders!

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Generic Constraints (extends and keyof)

Purpose: Building a type-safe `<DataTable<Row> columns={...} />` where column `accessorKey` MUST be a valid `keyof Row`!

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Built-in Utility Types (Partial, Required, Pick, Omi...

Purpose: Deriving Create, Patch, and Card-Preview DTOs directly from the canonical `Employee` interface so changes stay 100% synchronized!

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Advanced Types (Indexed Access, Mapped and Condition...

Purpose: Generating `type FormErrors<Values> = Partial<Record<keyof Values, string>>` automatically for any form!

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for TypeScript Modules

Purpose: Using `import type` cleanly across React and Angular components so bundlers drop type-only imports immediately.

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for TypeScript Classes

Purpose: Writing concise Angular services (`constructor(private readonly http: HttpClient) {}`) and repository adapters.

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Using JavaScript Libraries

Purpose: Typing `import.meta.env.VITE_API_URL` and third-party scripts cleanly without `@ts-ignore`!

What Topic Is Next → Next Topic 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for TypeScript Debugging

Purpose: Migrating Month 2's JavaScript Task Management Application cleanly into 100% strict TypeScript!

What Topic Is Next → Module 3 Capstone (Week 9 Lab Build: Strictly Typed TypeScript 5 Domain SDK (`strict: true`)) → Next Topic 4.1: Introduction to Angular and Enterprise Ecosystem

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 • 21 TOPICS (168 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Angular Fundamentals and Architecture

Enter enterprise Angular development: understand Angular's batteries-included architecture vs React, master Angular CLI, strict TypeScript decorators, Standalone Components, template bindings, Signals (`signal`, `computed`, `effect`), modern `@if`/`@for`/`@switch` control flow, `@Input`/`@Output`, Pipes, Lifecycle Hooks, and View Encapsulation.

Objectives: ✓ Contrast Angular's opinionated enterprise framework (built-in DI, Router, HttpClient, Reactive Forms, RxJS, Signals) with React's library ecosystem. • ✓ Scaffold strict Angular projects with Angular CLI (`ng new`) and build modular Standalone Components (`standalone: true`). • ✓ Master Angular template interpolation, property/event/two-way `[(ngModel)]` bindings, `@Input()`/`input()`, `@Output()`/`output()`, and reactive Signals (`signal`, `computed`, `effect`). • ✓ Use modern built-in `@if`, `@for` (with mandatory `track`), and `@switch` blocks alongside classic `*ngIf`/`*ngFor`, attribute directives, built-in & custom `@Pipe`s, and component lifecycle hooks.

4.1 Introduction to Angular and Enterprise Ecosystem [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why banking • Insurance • Healthcare • Large enterprise teams standardize on Angular • Production Pitfall Guard (Introduction to Angular and Enterprise Ecosystem) • Hands-On Verification (Introduction to Angular and Enterprise Ecosystem) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Introduction to Angular

Purpose: Positioning yourself for both React Product roles AND Enterprise Angular Frontend roles in interviews.

What Topic Is Next → Next Topic 4.2: Angular vs React: Architectural Comparison

4.2 Angular vs React: Architectural Comparison [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Framework (opinionated CLI • Router • Forms • HTTP included) vs UI Library (compose Vite + Router + RTK/Query + RHF) • Production Pitfall Guard (Angular vs React: Architectural Comparison) • Hands-On Verification (Angular vs React: Architectural Comparison) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Angular vs React

Purpose: Answering the classic interview question "Since you know both React and Angular, what are the real engineering trade-offs between them?" objectively.

What Topic Is Next → Next Topic 4.3: Node, npm and Angular CLI (`@angular/cli`) Setup

4.3 Node, npm and Angular CLI (`@angular/cli`) Setup [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Verifying Node LTS • Installing/running `@angular/cli` (`ng version` • `ng generate component/service/pipe/guard` • `ng serve` • Production Pitfall Guard (Node, npm and Angular CLI (`@angular/cli`) Setup) • Hands-On Verification (Node, npm and Angular CLI (`@angular/cli`) Setup) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Node

Purpose: Using `ng g c features/employees/employee-card` to scaffold consistent, spec-ready components across large teams.

What Topic Is Next → Next Topic 4.4: Creating an Angular Workspace (`ng new`)

4.4 Creating an Angular Workspace (`ng new`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Scaffolding with Standalone defaults • SCSS/CSS • Routing • Strict TypeScript + `strictTemplates` compiler checks • Production Pitfall Guard (Creating an Angular Workspace (`ng new`)) • Hands-On Verification (Creating an Angular Workspace (`ng new`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Creating an Angular Workspace (`ng new`)

Purpose: Enabling `strictTemplates: true` in `tsconfig.json` so template typos and wrong `@Input` types fail immediately at compile time!

What Topic Is Next → Next Topic 4.5: Angular Project Structure (`angular.json`, `main.ts`, `app.config.ts`)

4.5 Angular Project Structure (`angular.json`, `main.ts`, `app.config.ts`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `angular.json` build/architect targets • `src/main.ts` (`bootstrapApplication(AppComponent • AppConfig)`) • `app.config.ts` providers (`provideRouter` • Production Pitfall Guard (Angular Project Structure (`angular.json`, `main.ts`, `app.config.ts`)) • Hands-On Verification (Angular Project Structure (`angular.json`, `main.ts`, `app.config.ts`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular

Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Angular Project Structure (`angular.json`, `main.ts`...

Purpose: Registering application-wide router, HTTP, and animation providers cleanly inside `app.config.ts`.

What Topic Is Next → Next Topic 4.6: TypeScript for Angular (Decorators, Classes & Access Modifiers)

4.6 TypeScript for Angular (Decorators, Classes & Access Modifiers) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@Component` • `@Injectable` • `@Directive` • `@Pipe` metadata decorators • Production Pitfall Guard (TypeScript for Angular (Decorators, Classes & Access Modifiers)) • Hands-On Verification (TypeScript for Angular (Decorators, Classes & Access Modifiers)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for TypeScript for Angular (Decorators, Classes & Access...

Purpose: Marking template-bound signals/methods `protected readonly` so they are accessible to the component's own HTML template without leaking as public API.

What Topic Is Next → Next Topic 4.7: Angular Component Architecture (`@Component`)

4.7 Angular Component Architecture (`@Component`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `selector` (`app-...` kebab-case prefix) • `imports` • `templateUrl` vs inline `template` • `styleUrl` • Production Pitfall Guard (Angular Component Architecture (`@Component`)) • Hands-On Verification (Angular Component Architecture (`@Component`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Angular Component Architecture (`@Component`)

Purpose: Keeping complex enterprise forms/tables cleanly separated between declarative `.component.html` markup and typed `.component.ts` view logic.

What Topic Is Next → Next Topic 4.8: Standalone Components vs Legacy `NgModule` Awareness

4.8 Standalone Components vs Legacy `NgModule` Awareness [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Standalone Components import exactly what they use directly (`imports: [CommonModule • RouterLink]`) vs how legacy `NgModule` (`declarations` • `imports` • `exports`) workspaces work • Production Pitfall Guard (Standalone Components vs Legacy `NgModule` Awareness) • Hands-On Verification (Standalone Components vs Legacy `NgModule` Awareness) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Standalone Components vs Legacy `NgModule` Awareness

Purpose: Writing modern Standalone Angular code for greenfield apps while understanding `SharedModule`/`FeatureModule` when maintaining existing bank apps.

What Topic Is Next → Next Topic 4.9: Angular Templates and Expression Syntax

4.9 Angular Templates and Expression Syntax [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Template expressions vs statements • Safe navigation `user?.department?.name` • Nullish coalescing `??` • Template reference variables `#searchInput` • Production Pitfall Guard (Angular Templates and Expression Syntax) • Hands-On Verification (Angular Templates and Expression Syntax) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Angular Templates

Purpose: Referencing `#searchBox` directly in templates (`(keyup.enter)="onSearch(searchBox.value)"`) cleanly.

What Topic Is Next → Next Topic 4.10: Interpolation (`{{ }}`) and Property Binding (`[prop]`)

4.10 Interpolation (`{{ }}`) and Property Binding (`[prop]`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Auto-escaped text interpolation `{{ user.name }}` • DOM property binding `[disabled]="isSubmitting()"` • Attribute binding `[attr.aria-label]` • Class `[class.active]` & style `[style.width.%]` bindings • Production Pitfall Guard (Interpolation (`{{ }}`) and Property Binding (`[prop]`)) • Hands-On Verification (Interpolation (`{{ }}`) and Property Binding (`[prop]`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Interpolation (`{{ }}`)

Purpose: Using `[disabled]="form.invalid"` (boolean DOM property!) rather than string interpolation `disabled="{{ form.invalid }}"` (which always disables!).

What Topic Is Next → Next Topic 4.11: Event Binding (`(event)`) and Key Filters

4.11 Event Binding (`(event)`) and Key Filters [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `(click)="onSave()"` • Typed `$event` (`MouseEvent` • `Event`) • Built-in Angular key pseudo-events `(keydown.enter)` • Production Pitfall Guard (Event Binding (`(event)`) and Key Filters) • Hands-On Verification (Event Binding (`(event)`) and Key Filters) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Event Binding (`(event)`)

Purpose: Closing modals on `(keydown.escape)="close()"` and submitting search on `(keydown.enter)="search()"` declaratively.

What Topic Is Next → Next Topic 4.12: Two-Way Data Binding (`[(ngModel)]` and Custom `[(value)]`)

4.12 Two-Way Data Binding (`[(ngModel)]` and Custom `[(value)]`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `FormsModule` `[(ngModel)]="filterText"` • Banana-in-a-box desugaring `[value]="x" (valueChange)="x = $event"` • Modern `model()` signal two-way binding • Production Pitfall Guard (Two-Way Data Binding (`[(ngModel)]` and Custom `[(value)]`)) • Hands-On Verification (Two-Way Data Binding (`[(ngModel)]` and Custom `[(value)]`)) • Two-Way Data Binding (`[(ngModel)]` and Custom `[(value)]`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Two-Way Data Binding (`[(ngModel)]` and Custom `[(va...

Purpose: Binding quick toolbar filter selects or custom `<app-rating [(rating)]="score">` controls cleanly.

What Topic Is Next → Next Topic 4.13: Parent–Child Component Communication (`@Input`/`input` & `@Output`/`output`)

4.13 Parent–Child Component Communication (`@Input`/`input` & `@Output`/`output`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Passing data down via `@Input({ required: true })` / signal `input.required<Employee>()` • Emitting typed events up via `@Output() deleted = new EventEmitter<string>()` / `output<string>()` • Production Pitfall Guard (Parent–Child Component Communication (`@Input`/`input` & `@Output`/`output`)) • Hands-On Verification (Parent–Child Component Communication (`@Input`/`input` & `@Output`/`output`)) • Parent–Child Component Communication (`@Input`/`input` & `@Output`/`output`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Parent–Child Component Communication (`@Input`/`inpu... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Passing `employee` into `<app-employee-row [employee]="emp" (remove)="deleteEmployee($event)">` with compile-time required input safety.

What Topic Is Next → Next Topic 4.14: Angular Signals (`signal`, `computed` and `effect`)

4.14 Angular Signals (`signal`, `computed` and `effect`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fine-grained reactivity in modern Angular: writable `signal(0)` (`.set()` • `.update()`) • Derived memoized `computed(() => ...)` • Side-effect `effect()` • Production Pitfall Guard (Angular Signals (`signal`, `computed` and `effect`)) • Hands-On Verification (Angular Signals (`signal`, `computed` and `effect`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Angular Signals (`signal`, `computed` and `effect`)

Purpose: Deriving filtered/sorted KPI totals via `computed()` so Angular knows *precisely* which template expressions changed without scanning unchanged components!

What Topic Is Next → Next Topic 4.15: Directives Overview (Component, Structural and Attribute Directives)

4.15 Directives Overview (Component, Structural and Attribute Directives) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Components are directives with templates • Structural directives reshape DOM (`*ngIf` • `*ngFor`) • Attribute directives customize element behavior/appearance (`ngClass` • Production Pitfall Guard (Directives Overview (Component, Structural and Attribute Directives)) • Hands-On Verification (Directives Overview (Component, Structural and Attribute Directives)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Directives Overview (Component, Structural and Attri...

Purpose: Creating a custom `[appHasRole]="'ADMIN'"` or `[appAutoFocus]` directive to encapsulate reusable DOM behavior cleanly.

What Topic Is Next → Next Topic 4.16: Modern Built-In Control Flow (`@if`, `@for`, `@empty`, `@switch`) & `*ngIf`/`*ngFor`

4.16 Modern Built-In Control Flow (`@if`, `@for`, `@empty`, `@switch`) & `*ngIf`/`*ngFor` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Zero-import `@if (user) { ... } @else { ... }` • `@for (item of items() • Track item.id) { ... } @empty { ... }` • `@switch (status)` • Production Pitfall Guard (Modern Built-In Control Flow (`@if`, `@for`, `@empty`, `@switch`) & `*ngIf`/`*ngFor`) • Hands-On Verification (Modern Built-In Control Flow (`@if`, `@for`, `@empty`, `@switch`) & `*ngIf`/`*ngFor`) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Modern Built-In Control Flow (`@if`, `@for`, `@empty...

Purpose: Using `@for (emp of employees(); track emp.id)` with `@empty` to get optimal DOM reuse and zero-boilerplate empty states out of the box!

What Topic Is Next → Next Topic 4.17: Built-In Pipes (`currency`, `date`, `number`, `uppercase`, `json`, `async`)

4.17 Built-In Pipes (`currency`, `date`, `number`, `uppercase`, `json`, `async`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Declarative pure template formatting `{{ amount | currency:'INR':'symbol':'1.0-0' }}` • `{{ joinedAt | date:'dd MMM yyyy' }}` • Chaining pipes & pipe parameters • Production Pitfall Guard (Built-In Pipes (`currency`, `date`, `number`, `uppercase`, `json`, `async`)) • Hands-On Verification (Built-In Pipes (`currency`, `date`, `number`, `uppercase`, `json`, `async`)) • Built-In Pipes (`currency`, `date`, `number`, `uppercase`, `json`, `async`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Built-In Pipes (`currency`, `date`, `number`, `upper...

Purpose: Formatting Indian Rupee amounts and timestamps cleanly in templates via pure memoized pipes instead of component getter methods.

What Topic Is Next → Next Topic 4.18: Creating Custom Pure Pipes (`@Pipe`)

4.18 Creating Custom Pure Pipes (`@Pipe`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `PipeTransform` interface (`transform(value • ...args)`) • Pure (`pure: true` default — runs only when input reference changes!) vs impure pipes (`pure: false` performance danger!) • Production Pitfall Guard (Creating Custom Pure Pipes (`@Pipe`)) • Hands-On Verification (Creating Custom Pure Pipes (`@Pipe`)) • Creating Custom Pure Pipes (`@Pipe`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Creating Custom Pure Pipes (`@Pipe`)

Purpose: Writing pure `TruncatePipe`, `InitialsPipe` (`"Ravi Mishra" -> "RM"`), and `FileSizePipe` (`1048576 -> "1.0 MB"`).

What Topic Is Next → Next Topic 4.19: Component Lifecycle Hooks (`ngOnInit`, `ngOnChanges`, `ngAfterViewInit`, `ngOnDestroy`)

4.19 Component Lifecycle Hooks (`ngOnInit`, `ngOnChanges`, `ngAfterViewInit`, `ngOnDestroy`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Execution sequence from `constructor` -> `ngOnChanges` -> `ngOnInit` -> `ngAfterViewInit` -> `ngOnDestroy` (• Modern `DestroyRef.onDestroy` / `viewChild()` signal queries) • Production Pitfall Guard (Component Lifecycle Hooks (`ngOnInit`, `ngOnChanges`, `ngAfterViewInit`, `ngOnDestroy`)) • Hands-On Verification (Component Lifecycle Hooks (`ngOnInit`, `ngOnChanges`, `ngAfterViewInit`, `ngOnDestroy`)) • Component Lifecycle Hooks (`ngOnInit`, `ngOnChanges`, `ngAfterViewInit`, `ngOnDestroy`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Component Lifecycle Hooks (`ngOnInit`, `ngOnChanges`... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Initializing data subscriptions in `ngOnInit`, measuring DOM elements in `ngAfterViewInit`, and cleaning up timers/streams via `DestroyRef` / `ngOnDestroy`.

What Topic Is Next → Next Topic 4.20: View Encapsulation (`Emulated`, `None` and `ShadowDom`)

4.20 View Encapsulation (`Emulated`, `None` and `ShadowDom`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How default `ViewEncapsulation.Emulated` scopes component CSS with `[_ngcontent-ng-c...]` attributes • `:host` selector styling • Why `::ng-deep` is deprecated • Production Pitfall Guard (View Encapsulation (`Emulated`, `None` and `ShadowDom`)) • Hands-On Verification (View Encapsulation (`Emulated`, `None` and `ShadowDom`)) • View Encapsulation (`Emulated`, `None` and `ShadowDom`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for View Encapsulation (`Emulated`, `None` and `ShadowDo...

Purpose: Styling the component host element cleanly (`:host { display: block; }`) without leaking CSS rules into sibling features.

What Topic Is Next → Next Topic 4.21: Debugging with Angular DevTools

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

Subtopics Covered: Inspecting component hierarchy • Live `@Input`/Signal values • Dependency injection resolution tree • Change Detection flamegraph profiler in Chrome Angular DevTools • Production Pitfall Guard (Debugging with Angular DevTools) • Hands-On Verification (Debugging with Angular DevTools) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Debugging with Angular DevTools

Purpose: Profiling change-detection cycles and verifying Signal values in real time using the official Angular DevTools Chrome extension.

What Topic Is Next → Module 4 Capstone (Week 17 Lab Build: Standalone Angular Inventory Manager with Signals & Pipes) → Next Topic 5.1: Service Layer Architecture (`@Injectable`)

Module 4 Hands-On Capstone Project (100 Marks): Week 17 Lab Build: Standalone Angular Inventory Manager with Signals & Pipes — Build a Standalone Angular 17+ inventory tracker using `signal()`, `computed()`, `input()`/`output()`, `@if`/`@for` (`track item.id`), built-in `currency:'INR'` formatting, and custom `Initi...

MODULE 5 OF 10 • STAGE 2 • 11 LIVE + LAB HOURS • 15 TOPICS (120 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Angular Services, Dependency Injection, Router, Lazy Loading & Guards

Architect enterprise Angular service layers, Hierarchical Dependency Injection (`providedIn: 'root'`, `InjectionToken`, `inject()`), multi-level Angular Routing (params, query params, child routes, lazy `loadComponent`/`loadChildren`, functional `CanActivateFn` guards, resolvers), and typed `HttpClient` REST API integration with functional HTTP Interceptors.

Objectives: ✓ Extract business logic and API orchestration out of components into testable `@Injectable` services injected via `inject()` or constructor DI. • ✓ Configure multi-view Angular Router navigation with nested `<router-outlet>`, route/query params, lazy-loaded feature routes, and functional route guards (`CanActivateFn`, `CanDeactivateFn`). • ✓ Perform typed `GET`, `POST`, `PUT`/`PATCH`, and `DELETE` REST operations via `provideHttpClient(withInterceptors([...]))`. • ✓ Implement functional HTTP Interceptors attaching `Authorization: Bearer` headers and centralized RxJS `catchError` handling for `401`/`403`/`500` responses.

5.1 Service Layer Architecture (`@Injectable`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Thin presentation components + cohesive `@Injectable` domain services (`EmployeeApiService` • `AuthService` • `NotificationService`) • Production Pitfall Guard (Service Layer Architecture (`@Injectable`)) • Hands-On Verification (Service Layer Architecture (`@Injectable`)) • Service Layer Architecture (`@Injectable`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Service Layer Architecture (`@Injectable`)

Purpose: Keeping components focused strictly on UI state & bindings while `EmployeeApiService` owns endpoint URLs, DTO mapping, and caching.

What Topic Is Next → Next Topic 5.2: Dependency Injection (DI) Design Pattern in Angular

5.2 Dependency Injection (DI) Design Pattern in Angular [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inversion of Control: requesting abstractions/tokens from Angular's Injector instead of `new EmployeeService()` • Swapping mocks effortlessly in unit tests • Production Pitfall Guard (Dependency Injection (DI) Design Pattern in Angular) • Hands-On Verification (Dependency Injection (DI) Design Pattern in Angular) • Dependency Injection (DI) Design Pattern in Angular — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Dependency Injection (DI) Design Pattern in Angular • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Providing a fake in-memory `AuthService` stub in `TestBed.configureTestingModule` with zero changes to component code!

What Topic Is Next → Next Topic 5.3: Providers (`useClass`, `useValue`, `useFactory`, `InjectionToken`)

5.3 Providers (`useClass`, `useValue`, `useFactory`, `InjectionToken`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Typing non-class configuration values (`API_BASE_URL`) safely via `new InjectionToken<string>('API_BASE_URL')` • Factory providers • Production Pitfall Guard (Providers (`useClass`, `useValue`, `useFactory`, `InjectionToken`)) • Hands-On Verification (Providers (`useClass`, `useValue`, `useFactory`, `InjectionToken`)) • Providers (`useClass`, `useValue`, `useFactory`, `InjectionToken`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Providers (`useClass`, `useValue`, `useFactory`, `In... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Injecting `API_BASE_URL` via a typed `InjectionToken<string>` configured once in `app.config.ts`.

What Topic Is Next → Next Topic 5.4: Modern `inject()` Function vs Constructor Injection

5.4 Modern `inject()` Function vs Constructor Injection [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Field-initializer `private readonly auth = inject(AuthService) • ` (injection context rules!) • Using `inject()` inside functional guards • Interceptors • Production Pitfall Guard (Modern `inject()` Function vs Constructor Injection) • Hands-On Verification (Modern `inject()` Function vs Constructor Injection) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Modern `inject()` Function vs Constructor Injection

Purpose: Writing concise 6-line `authGuard: CanActivateFn` functions that call `inject(AuthService)` and `inject(Router)` directly!

What Topic Is Next → Next Topic 5.5: Hierarchical Injectors (`root` vs Route vs Component Providers)

5.5 Hierarchical Injectors (`root` vs Route vs Component Providers) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: App-wide tree-shakeable singletons (`providedIn: 'root'`) vs route-scoped providers vs per-component instances (`@Component({ providers: [DraftWizardService] })`) • Production Pitfall Guard (Hierarchical Injectors (`root` vs Route vs Component Providers)) • Hands-On Verification (Hierarchical Injectors (`root` vs Route vs Component Providers)) • Hierarchical Injectors (`root` vs Route vs Component Providers) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Hierarchical Injectors (`root` vs Route vs Component... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Providing `WizardStepService` at the parent `<app-checkout-wizard>` component level so its state auto-disposes when leaving checkout!

What Topic Is Next → Next Topic 5.6: Angular Router Architecture (`provideRouter`)

5.6 Angular Router Architecture (`provideRouter`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring `Routes` in `app.routes.ts` • Registering `provideRouter(routes • WithComponentInputBinding())` in `app.config.ts` • HTML5 pushState navigation • Production Pitfall Guard (Angular Router Architecture (`provideRouter`)) • Hands-On Verification (Angular Router Architecture (`provideRouter`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Angular Router Architecture (`provideRouter`)

Purpose: Enabling `withComponentInputBinding()` so route params `:id` and query params bind directly to component `input()` signals!

What Topic Is Next → Next Topic 5.7: Route Configuration, Redirects and Wildcard 404 Routes

5.7 Route Configuration, Redirects and Wildcard 404 Routes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `path` (no leading slash!) • `redirectTo` with `pathMatch: 'full'` • `title` page-title resolver • `**` 404 fallback • Production Pitfall Guard (Route Configuration, Redirects and Wildcard 404 Routes) • Hands-On Verification (Route Configuration, Redirects and Wildcard 404 Routes) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Route Configuration

Purpose: Setting route `title: 'Employees | Enterprise Admin'` so browser tab titles and screen-reader route announcements update automatically.

What Topic Is Next → Next Topic 5.8: Rendering Views with `<router-outlet>`

5.8 Rendering Views with `<router-outlet>` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Primary `<router-outlet />` placement inside shell layouts • Sibling header/sidebar persistence during route swaps • Production Pitfall Guard (Rendering Views with `<router-outlet>`) • Hands-On Verification (Rendering Views with `<router-outlet>`) • Rendering Views with `<router-outlet>` — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Rendering Views with `<router-outlet>` • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Rendering `<app-sidebar>` + `<app-topbar>` once in `AdminShellComponent` while child admin pages swap cleanly inside `<router-outlet />`.

What Topic Is Next → Next Topic 5.9: Declarative Navigation (`routerLink` and `routerLinkActive`)

5.9 Declarative Navigation (`routerLink` and `routerLinkActive`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `routerLink="/employees"` • Array link `[routerLink]="['/employees' • Emp.id]"` • `routerLinkActive="bg-indigo-600 text-white"` • Production Pitfall Guard (Declarative Navigation (`routerLink` and `routerLinkActive`)) • Hands-On Verification (Declarative Navigation (`routerLink` and `routerLinkActive`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Declarative Navigation (`routerLink` and `routerLink...

Purpose: Setting `{ exact: true }` on the root `/admin` link so it isn't permanently highlighted on `/admin/users` and `/admin/orders`!

What Topic Is Next → Next Topic 5.10: Dynamic Route Parameters (`ActivatedRoute` & Input Binding)

5.10 Dynamic Route Parameters (`ActivatedRoute` & Input Binding) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reading `:id` reactively via `ActivatedRoute.paramMap` (or `withComponentInputBinding()` `id = input.required<string>()`) • Reacting when navigating from `/users/1` to `/users/2` • Production Pitfall Guard (Dynamic Route Parameters (`ActivatedRoute` & Input Binding)) • Hands-On Verification (Dynamic Route Parameters (`ActivatedRoute` & Input Binding)) • Dynamic Route Parameters (`ActivatedRoute` & Input Binding) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Dynamic Route Parameters (`ActivatedRoute` & Input B... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Subscribing to `route.paramMap.pipe(switchMap(p => api.getById(p.get('id')!)))` so clicking "Next Employee" updates data without remounting!

What Topic Is Next → Next Topic 5.11: Query Parameters (`?search=&status=&page=`)

5.11 Query Parameters (`?search=&status=&page=`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `[queryParams]="{ page: 2 }"` • `queryParamsHandling="merge"` • Programmatic `router.navigate([] • { queryParams • Production Pitfall Guard (Query Parameters (`?search=&status=&page=`)) • Hands-On Verification (Query Parameters (`?search=&status=&page=`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Query Parameters (`?search=&status=&page=`)

Purpose: Syncing table search, status filter, sort column, and page number to URL query params so bookmarked/shared admin links restore exact filters!

What Topic Is Next → Next Topic 5.12: Nested Child Routes (`children: [...]`)

5.12 Nested Child Routes (`children: [...]`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-level layouts: Auth layout (`/login` • `/forgot-password`) vs Admin layout (`/admin/overview` • `/admin/users` • `/admin/settings`) • Production Pitfall Guard (Nested Child Routes (`children: [...]`)) • Hands-On Verification (Nested Child Routes (`children: [...]`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Nested Child Routes (`children: [...]`)

Purpose: Nesting `Profile`, `Attendance`, and `Payroll` child routes inside `/employees/:id` with a shared employee header card.

What Topic Is Next → Next Topic 5.13: Lazy-Loaded Feature Routes (`loadComponent` & `loadChildren`)

5.13 Lazy-Loaded Feature Routes (`loadComponent` & `loadChildren`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Splitting standalone pages via `loadComponent: () => import(...).then(m => m.UsersPageComponent)` • Entire feature route files via `loadChildren` • Production Pitfall Guard (Lazy-Loaded Feature Routes (`loadComponent` & `loadChildren`)) • Hands-On Verification (Lazy-Loaded Feature Routes (`loadComponent` & `loadChildren`)) • Lazy-Loaded Feature Routes (`loadComponent` & `loadChildren`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Lazy-Loaded Feature Routes (`loadComponent` & `loadC... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Keeping initial `main.js` small by lazy-loading `/reports`, `/audit-logs`, and `/settings` feature route trees on navigation.

What Topic Is Next → Next Topic 5.14: Functional Route Guards (`CanActivateFn`, `CanMatchFn`, `CanDeactivateFn`)

5.14 Functional Route Guards (`CanActivateFn`, `CanMatchFn`, `CanDeactivateFn`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Returning `boolean | UrlTree` (`router.createUrlTree(['/login'] • { queryParams: { returnUrl: state.url } })`) • Unsaved-form navigation warning via `CanDeactivateFn` • Production Pitfall Guard (Functional Route Guards (`CanActivateFn`, `CanMatchFn`, `CanDeactivateFn`)) • Hands-On Verification (Functional Route Guards (`CanActivateFn`, `CanMatchFn`, `CanDeactivateFn`)) • Functional Route Guards (`CanActivateFn`, `CanMatchFn`, `CanDeactivateFn`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Functional Route Guards (`CanActivateFn`, `CanMatchF...

Purpose: Redirecting unauthenticated visitors via `UrlTree` (instead of side-effect `router.navigate` + `return false`) to avoid router race conditions!

What Topic Is Next → Next Topic 5.15: Route Resolvers (`ResolveFn`) and Pre-Navigation Data

5.15 Route Resolvers (`ResolveFn`) and Pre-Navigation Data [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Writing `ResolveFn<Employee>` vs skeleton-in-component UX trade-offs (when pre-fetching blocks navigation vs immediate shell + skeleton) • Production Pitfall Guard (Route Resolvers (`ResolveFn`) and Pre-Navigation Data) • Hands-On Verification (Route Resolvers (`ResolveFn`) and Pre-Navigation Data) • Route Resolvers (`ResolveFn`) and Pre-Navigation Data — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Route Resolvers (`ResolveFn`) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Understanding when lightweight metadata/lookup resolvers help vs why modern UX prefers instant route transition + in-page skeleton loaders.

What Topic Is Next → Module 5 Capstone (Week 5 Practical Build: Angular Services, Dependency Injection, Router, Lazy Loading & Guards) → Next Topic 6.1: Configuring `provideHttpClient` and `HttpClient` Fundamentals

Module 5 Hands-On Capstone Project (100 Marks): Week 5 Practical Build: Angular Services, Dependency Injection, Router, Lazy Loading & Guards — Implement and verify a clean, runnable ANGU repository covering Architect enterprise Angular service layers, Hierarchical Dependency Injection (`providedIn: 'root'`, `InjectionToken`, `injec...

MODULE 6 OF 10 • STAGE 2 • 11 LIVE + LAB HOURS • 9 TOPICS (72 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Angular `HttpClient` REST APIs, Functional Interceptors & Role Auth

Architect enterprise Angular service layers, Hierarchical Dependency Injection (`providedIn: 'root'`, `InjectionToken`, `inject()`), multi-level Angular Routing (params, query params, child routes, lazy `loadComponent`/`loadChildren`, functional `CanActivateFn` guards, resolvers), and typed `HttpClient` REST API integration with functional HTTP Interceptors.

Objectives: ✓ Extract business logic and API orchestration out of components into testable `@Injectable` services injected via `inject()` or constructor DI. • ✓ Configure multi-view Angular Router navigation with nested `<router-outlet>`, route/query params, lazy-loaded feature routes, and functional route guards (`CanActivateFn`, `CanDeactivateFn`). • ✓ Perform typed `GET`, `POST`, `PUT`/`PATCH`, and `DELETE` REST operations via `provideHttpClient(withInterceptors([...]))`. • ✓ Implement functional HTTP Interceptors attaching `Authorization: Bearer` headers and centralized RxJS `catchError` handling for `401`/`403`/`500` responses.

6.1 Configuring `provideHttpClient` and `HttpClient` Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `provideHttpClient(withFetch() • WithInterceptors([...]))` in `app.config.ts` • Cold Observable nature of `HttpClient` (request fires ONLY on `.subscribe()` or `| async` / `toSignal`!) • Auto JSON parsing • Production Pitfall Guard (Configuring `provideHttpClient` and `HttpClient` Fundamentals) • Hands-On Verification (Configuring `provideHttpClient` and `HttpClient` Fundamentals) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Configuring `provideHttpClient`

Purpose: Typing `this.http.get<PagedResponse<Employee>>(...)` for compile-time DTO verification across the service layer.

What Topic Is Next → Next Topic 6.2: Typed HTTP `GET` Requests with `HttpParams` and `HttpHeaders`

6.2 Typed HTTP `GET` Requests with `HttpParams` and `HttpHeaders` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable `new HttpParams().set('page' • Page).set('q' • Query)` builder • Typed generics `http.get<Employee[]>(url • Production Pitfall Guard (Typed HTTP `GET` Requests with `HttpParams` and `HttpHeaders`) • Hands-On Verification (Typed HTTP `GET` Requests with `HttpParams` and `HttpHeaders`) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Typed HTTP `GET` Requests with `HttpParams`

Purpose: Building safe URL-encoded query parameters via immutable `HttpParams` instead of manual string concatenation.

What Topic Is Next → Next Topic 6.3: HTTP `POST` Requests (Creating Resources)

6.3 HTTP `POST` Requests (Creating Resources) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sending typed `CreateEmployeeDto` JSON payloads `this.http.post<Employee>(url • Dto)` • Disabling submit controls while request is in flight • Production Pitfall Guard (HTTP `POST` Requests (Creating Resources)) • Hands-On Verification (HTTP `POST` Requests (Creating Resources)) • HTTP `POST` Requests (Creating Resources) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for HTTP `POST` Requests (Creating Resources)

Purpose: Creating new employee/product records with typed request & response interfaces and `finalize(() => this.saving.set(false))`.

What Topic Is Next → Next Topic 6.4: HTTP `PUT` and `PATCH` Requests (Updating Resources)

6.4 HTTP `PUT` and `PATCH` Requests (Updating Resources) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Full replacement `PUT /employees/:id` vs partial `PATCH /employees/:id/status` (`Partial<UpdateEmployeeDto>`) • Optimistic vs confirmed state refresh • Production Pitfall Guard (HTTP `PUT` and `PATCH` Requests (Updating Resources)) • Hands-On Verification (HTTP `PUT` and `PATCH` Requests (Updating Resources)) • HTTP `PUT` and `PATCH` Requests (Updating Resources) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for HTTP `PUT` • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Updating an employee's active status via `PATCH` and editing full profile details via `PUT`.

What Topic Is Next → Next Topic 6.5: HTTP `DELETE` Requests and Safe Confirmation Flows

6.5 HTTP `DELETE` Requests and Safe Confirmation Flows [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `this.http.delete<void>(`${url}/${id}`)` • Confirmation dialog before destructive deletion • Handling `204 No Content` responses • Production Pitfall Guard (HTTP `DELETE` Requests and Safe Confirmation Flows) • Hands-On Verification (HTTP `DELETE` Requests and Safe Confirmation Flows) • HTTP `DELETE` Requests and Safe Confirmation Flows — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for HTTP `DELETE` Requests

Purpose: Guarding row deletions behind an accessible confirmation modal and refreshing list state cleanly upon `204` completion.

What Topic Is Next → Next Topic 6.6: Functional HTTP Interceptors (`HttpInterceptorFn`)

6.6 Functional HTTP Interceptors (`HttpInterceptorFn`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cloning immutable requests `req.clone({ setHeaders: { Authorization: `Bearer ${token}` } })` • Chaining `next(authReq)` • Logging request duration with `tap`/`finalize` • Production Pitfall Guard (Functional HTTP Interceptors (`HttpInterceptorFn`)) • Hands-On Verification (Functional HTTP Interceptors (`HttpInterceptorFn`)) • Functional HTTP Interceptors (`HttpInterceptorFn`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Functional HTTP Interceptors (`HttpInterceptorFn`)

Purpose: Attaching JWT Bearer tokens and `X-Request-Id` headers centrally in 12 lines via `authInterceptor: HttpInterceptorFn`.

What Topic Is Next → Next Topic 6.7: Centralized RxJS HTTP Error Handling (`HttpErrorResponse`)

6.7 Centralized RxJS HTTP Error Handling (`HttpErrorResponse`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Distinguishing network failure (`err.status === 0`) vs validation (`422`) vs expired session (`401`) vs forbidden (`403`) vs server error (`500+`) • `catchError` + `throwError(() => domainErr)` • Production Pitfall Guard (Centralized RxJS HTTP Error Handling (`HttpErrorResponse`)) • Hands-On Verification (Centralized RxJS HTTP Error Handling (`HttpErrorResponse`)) • Centralized RxJS HTTP Error Handling (`HttpErrorResponse`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Centralized RxJS HTTP Error Handling (`HttpErrorResp... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Mapping raw `HttpErrorResponse` payloads into user-friendly domain messages in an `errorInterceptor` while rethrowing for field-level `422` forms.

What Topic Is Next → Next Topic 6.8: End-to-End Angular Authentication Workflow

6.8 End-to-End Angular Authentication Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `AuthService` holding `currentUser = signal<User | null>(null)` + `isAuthenticated = computed(...)` • Login/session restore on startup (`APP_INITIALIZER` / `provideAppInitializer`) • Logout cleanup • Production Pitfall Guard (End-to-End Angular Authentication Workflow) • Hands-On Verification (End-to-End Angular Authentication Workflow) • End-to-End Angular Authentication Workflow — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for End-to-End Angular Authentication Workflow

Purpose: Restoring authenticated user state cleanly before initial route guard evaluation so refreshing `/admin/users` doesn't falsely kick logged-in admins to `/login`!

What Topic Is Next → Next Topic 6.9: Role-Based Protected Routes (`ADMIN`, `MANAGER`, `VIEWER`)

6.9 Role-Based Protected Routes (`ADMIN`, `MANAGER`, `VIEWER`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Parameterized `hasRoleGuard(['ADMIN' • 'MANAGER'])` • Hiding unauthorized UI actions via custom `*appHasRole` / `@if (auth.hasRole('ADMIN'))` alongside route guards • Production Pitfall Guard (Role-Based Protected Routes (`ADMIN`, `MANAGER`, `VIEWER`)) • Hands-On Verification (Role-Based Protected Routes (`ADMIN`, `MANAGER`, `VIEWER`)) • Role-Based Protected Routes (`ADMIN`, `MANAGER`, `VIEWER`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Role-Based Protected Routes (`ADMIN`, `MANAGER`, `VI...

Purpose: Restricting `/admin/payroll` and Delete actions to `ADMIN` role while allowing `VIEWER` read-only dashboard access.

What Topic Is Next → Module 6 Capstone (Week 18 Lab Build: Guarded Angular Router & `HttpClient` Interceptor Portal) → Next Topic 7.1: Template-Driven Forms (`FormsModule` and `ngForm`)

Module 6 Hands-On Capstone Project (100 Marks): Week 18 Lab Build: Guarded Angular Router & `HttpClient` Interceptor Portal — Build a multi-route Angular portal with nested `<router-outlet>`, lazy `loadComponent` routes, functional `authGuard`/`roleGuard`, typed `HttpClient` CRUD service, and `authInterceptor` + `e...

MODULE 7 OF 10 • STAGE 3 • 11 LIVE + LAB HOURS • 7 TOPICS (56 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Strictly Typed Angular Reactive Forms, Custom Validators & `FormArray`

Master Angular Template-Driven vs Strictly Typed Reactive Forms (`FormGroup`, `FormControl`, `NonNullableFormBuilder`, sync/async custom validators, dynamic `FormArray`), deep RxJS reactive streams (`Observable`, `Subject`, `BehaviorSubject`, `switchMap`/`mergeMap`/`concatMap`/`exhaustMap`, `combineLatest`, `AsyncPipe`, Signals interop), and NgRx Store architecture.

Objectives: ✓ Build complex, strictly typed Angular Reactive Forms using `NonNullableFormBuilder`, cross-field validators, async uniqueness validators, and dynamic `FormArray` line items. • ✓ Think reactively with RxJS Observables, Subjects, `BehaviorSubject`, automatic teardown (`takeUntilDestroyed` / `AsyncPipe`), and Signal interop (`toSignal` / `toObservable`). • ✓ Choose the exact flattening operator (`switchMap` for search cancel, `exhaustMap` for login double-click prevention, `concatMap` for ordered saves, `mergeMap` for parallel tasks) in real apps and interviews. • ✓ Manage shared Angular state cleanly using Signal/BehaviorSubject Services AND enterprise NgRx (`Store`, `createActionGroup`, `createReducer`, `createEffect`, `createSelector`).

7.1 Template-Driven Forms (`FormsModule` and `ngForm`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `#f="ngForm"` • `ngModel` with `name` • `required`/`email`/`minlength` attributes • `#emailCtrl="ngModel"` `dirty`/`touched`/`invalid` CSS states (`ng-touched` • Production Pitfall Guard (Template-Driven Forms (`FormsModule` and `ngForm`)) • Hands-On Verification (Template-Driven Forms (`FormsModule` and `ngForm`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Template-Driven Forms (`FormsModule` and `ngForm`)

Purpose: Understanding quick 2-field newsletter/filter forms and maintaining existing Template-Driven forms in legacy Angular modules.

What Topic Is Next → Next Topic 7.2: Strictly Typed Reactive Forms (`ReactiveFormsModule`, `FormControl`, `FormGroup`)

7.2 Strictly Typed Reactive Forms (`ReactiveFormsModule`, `FormControl`, `FormGroup`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Explicit TypeScript model in TS class (`FormControl<string>` • `FormGroup<{...}>`) • `[formGroup]` • `formControlName` • Production Pitfall Guard (Strictly Typed Reactive Forms (`ReactiveFormsModule`, `FormControl`, `FormGroup`)) • Hands-On Verification (Strictly Typed Reactive Forms (`ReactiveFormsModule`, `FormControl`, `FormGroup`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Strictly Typed Reactive Forms (`ReactiveFormsModule`...

Purpose: Testing complex 20-field enterprise payroll/tax validation rules in plain TypeScript unit tests in 5ms!

What Topic Is Next → Next Topic 7.3: `NonNullableFormBuilder` Ergonomics

7.3 `NonNullableFormBuilder` Ergonomics [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Injecting `fb = inject(NonNullableFormBuilder)` • `fb.group({ ... })` • `patchValue` (partial fill) vs `setValue` (strict full shape) • `reset()` restoring initial defaults • Production Pitfall Guard (`NonNullableFormBuilder` Ergonomics) • Hands-On Verification (`NonNullableFormBuilder` Ergonomics) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for `NonNullableFormBuilder` Ergonomics

Purpose: Pre-populating edit forms cleanly via `this.form.patchValue(employee)` when loading an existing record from the API.

What Topic Is Next → Next Topic 7.4: Built-In Validators (`Validators`) and Accessible Error Display

7.4 Built-In Validators (`Validators`) and Accessible Error Display [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `required` • `email` • `minLength` • `maxLength` • Production Pitfall Guard (Built-In Validators (`Validators`) and Accessible Error Display) • Hands-On Verification (Built-In Validators (`Validators`) and Accessible Error Display) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Built-In Validators (`Validators`)

Purpose: Calling `if (this.form.invalid) { this.form.markAllAsTouched(); return; }` so clicking Submit immediately highlights every invalid field!

What Topic Is Next → Next Topic 7.5: Custom Synchronous & Asynchronous Validators (`ValidatorFn`, `AsyncValidatorFn`)

7.5 Custom Synchronous & Asynchronous Validators (`ValidatorFn`, `AsyncValidatorFn`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Field `ValidatorFn` returning `ValidationErrors | null` • Cross-field group validator (`password === confirmPassword` • `endDate >= startDate`) • Debounced `AsyncValidatorFn` checking email availability • Production Pitfall Guard (Custom Synchronous & Asynchronous Validators (`ValidatorFn`, `AsyncValidatorFn`)) • Hands-On Verification (Custom Synchronous & Asynchronous Validators (`ValidatorFn`, `AsyncValidatorFn`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Custom Synchronous

Purpose: Validating Indian PAN/GSTIN format synchronously and checking employee-code uniqueness asynchronously on `updateOn: 'blur'`.

What Topic Is Next → Next Topic 7.6: Dynamic Forms with `FormArray`

7.6 Dynamic Forms with `FormArray` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `this.fb.array([this.createSkillGroup()])` • `addSkill()` • `removeAt(index)` • `formArrayName` + `@for (ctrl of skills.controls • Production Pitfall Guard (Dynamic Forms with `FormArray`) • Hands-On Verification (Dynamic Forms with `FormArray`) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Dynamic Forms with `FormArray`

Purpose: Building multi-row Invoice Line Items, Employee Work Experience entries, or Dynamic Filter Rule Builders with live row totals.

What Topic Is Next → Next Topic 7.7: Enterprise Form UX: Dirty State, Conditional Validators & Server `422` Mapping

7.7 Enterprise Form UX: Dirty State, Conditional Validators & Server `422` Mapping [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reacting to `roleControl.valueChanges` via `setValidators` + `updateValueAndValidity()` • Mapping backend `422` field errors via `ctrl.setErrors({ server: msg })` • Production Pitfall Guard (Enterprise Form UX: Dirty State, Conditional Validators & Server `422` Mapping) • Hands-On Verification (Enterprise Form UX: Dirty State, Conditional Validators & Server `422` Mapping) • Enterprise Form UX: Dirty State, Conditional Validators & Server `422` Mapping — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Enterprise Form UX • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Making `contractorVendorName` required dynamically ONLY when `employmentType === 'CONTRACT'`, and always calling `updateValueAndValidity()`!

What Topic Is Next → Module 7 Capstone (Week 7 Practical Build: Strictly Typed Angular Reactive Forms, Custom Validators & `FormArray`) → Next Topic 8.1: Introduction to Reactive Programming & RxJS

Module 7 Hands-On Capstone Project (100 Marks): Week 7 Practical Build: Strictly Typed Angular Reactive Forms, Custom Validators & `FormArray` — Implement and verify a clean, runnable ANGU repository covering Master Angular Template-Driven vs Strictly Typed Reactive Forms (`FormGroup`, `FormControl`, `NonNullableFormBuilder`, sync/as...

MODULE 8 OF 10 • STAGE 3 • 11 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: RxJS Reactive Streams (`switchMap`, `combineLatest`, `AsyncPipe`) & NgRx Store

Master Angular Template-Driven vs Strictly Typed Reactive Forms (`FormGroup`, `FormControl`, `NonNullableFormBuilder`, sync/async custom validators, dynamic `FormArray`), deep RxJS reactive streams (`Observable`, `Subject`, `BehaviorSubject`, `switchMap`/`mergeMap`/`concatMap`/`exhaustMap`, `combineLatest`, `AsyncPipe`, Signals interop), and NgRx Store architecture.

Objectives: ✓ Build complex, strictly typed Angular Reactive Forms using `NonNullableFormBuilder`, cross-field validators, async uniqueness validators, and dynamic `FormArray` line items. • ✓ Think reactively with RxJS Observables, Subjects, `BehaviorSubject`, automatic teardown (`takeUntilDestroyed` / `AsyncPipe`), and Signal interop (`toSignal` / `toObservable`). • ✓ Choose the exact flattening operator (`switchMap` for search cancel, `exhaustMap` for login double-click prevention, `concatMap` for ordered saves, `mergeMap` for parallel tasks) in real apps and interviews. • ✓ Manage shared Angular state cleanly using Signal/BehaviorSubject Services AND enterprise NgRx (`Store`, `createActionGroup`, `createReducer`, `createEffect`, `createSelector`).

8.1 Introduction to Reactive Programming & RxJS [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Push vs Pull streams • Treating events • Form valueChanges • Router params • Production Pitfall Guard (Introduction to Reactive Programming & RxJS) • Hands-On Verification (Introduction to Reactive Programming & RxJS) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Introduction to Reactive Programming

Purpose: Combining search input + department dropdown + page changes into a single declarative HTTP stream without nested flags or race bugs.

What Topic Is Next → Next Topic 8.2: Observables: Cold vs Hot Streams (`of`, `from`, `interval`, `timer`, `shareReplay`)

8.2 Observables: Cold vs Hot Streams (`of`, `from`, `interval`, `timer`, `shareReplay`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Observable vs Promise (lazy • Multi-value over time • Cancellable `.unsubscribe()`!) • Unicast Cold HTTP streams vs multicast Hot `shareReplay({ bufferSize: 1 • Production Pitfall Guard (Observables: Cold vs Hot Streams (`of`, `from`, `interval`, `timer`, `shareReplay`)) • Hands-On Verification (Observables: Cold vs Hot Streams (`of`, `from`, `interval`, `timer`, `shareReplay`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Observables: Cold vs Hot Streams (`of`, `from`, `int...

Purpose: Using `shareReplay({ bufferSize: 1, refCount: true })` so 3 `| async` bindings in a template share ONE HTTP request instead of firing 3 duplicate GETs!

What Topic Is Next → Next Topic 8.3: Observers (`next`, `error`, `complete`) and Stream Contract

8.3 Observers (`next`, `error`, `complete`) and Stream Contract [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: RxJS stream grammar: `0..N` `next` emissions followed optionally by EITHER one `complete` OR one `error` (terminal!) • Why uncaught errors kill long-lived streams • Production Pitfall Guard (Observers (`next`, `error`, `complete`) and Stream Contract) • Hands-On Verification (Observers (`next`, `error`, `complete`) and Stream Contract) • Observers (`next`, `error`, `complete`) and Stream Contract — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Observers (`next`, `error`, `complete`) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Placing `catchError(...)` INSIDE the inner `switchMap` HTTP call so one failed search query does NOT permanently kill the outer `searchControl.valueChanges` stream!

What Topic Is Next → Next Topic 8.4: Subscription Lifecycle and Automatic Teardown (`takeUntilDestroyed`)

8.4 Subscription Lifecycle and Automatic Teardown (`takeUntilDestroyed`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `HttpClient` one-shot streams auto-complete vs why `interval` • `valueChanges` • `route.paramMap` • `Subject` leak memory if unsubscribed • Production Pitfall Guard (Subscription Lifecycle and Automatic Teardown (`takeUntilDestroyed`)) • Hands-On Verification (Subscription Lifecycle and Automatic Teardown (`takeUntilDestroyed`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Subscription Lifecycle

Purpose: Piping `.pipe(takeUntilDestroyed(this.destroyRef))` on every manual long-lived subscription when `AsyncPipe` or `toSignal` isn't used.

What Topic Is Next → Next Topic 8.5: Core Pipeable Operators (`map`, `filter`, `tap`, `debounceTime`, `distinctUntilChanged`, `startWith`)

8.5 Core Pipeable Operators (`map`, `filter`, `tap`, `debounceTime`, `distinctUntilChanged`, `startWith`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Transforming emissions (`map`) • Guarding (`filter`) • Side-effect logging (`tap`) • Deduplicating identical trimmed search terms (`distinctUntilChanged()`) • Production Pitfall Guard (Core Pipeable Operators (`map`, `filter`, `tap`, `debounceTime`, `distinctUntilChanged`, `startWith`)) • Hands-On Verification (Core Pipeable Operators (`map`, `filter`, `tap`, `debounceTime`, `distinctUntilChanged`, `startWith`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Core Pipeable Operators (`map`, `filter`, `tap`, `de...

Purpose: Building `valueChanges.pipe(startWith(''), map(s => s.trim()), debounceTime(300), distinctUntilChanged())` for clean typeahead streams.

What Topic Is Next → Next Topic 8.6: Higher-Order Flattening Operators (`switchMap`, `mergeMap`, `concatMap`, `exhaustMap`)

8.6 Higher-Order Flattening Operators (`switchMap`, `mergeMap`, `concatMap`, `exhaustMap`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `switchMap` (cancel previous in-flight: live search/route param) • `exhaustMap` (ignore clicks while busy: login/checkout submit!) • `concatMap` (strict FIFO queue: sequential updates) • `mergeMap` (parallel fan-out) • Production Pitfall Guard (Higher-Order Flattening Operators (`switchMap`, `mergeMap`, `concatMap`, `exhaustMap`)) • Hands-On Verification (Higher-Order Flattening Operators (`switchMap`, `mergeMap`, `concatMap`, `exhaustMap`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Higher-Order Flattening Operators (`switchMap`, `mer...

Purpose: Preventing double-charge order submissions with `exhaustMap` and cancelling stale search XHRs automatically with `switchMap`!

What Topic Is Next → Next Topic 8.7: Combination Operators (`combineLatest`, `forkJoin`, `withLatestFrom`)

8.7 Combination Operators (`combineLatest`, `forkJoin`, `withLatestFrom`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `forkJoin({ depts • Roles • Stats })` (`Promise.all` equivalent waiting for completion) vs `combineLatest([query$ • Dept$ • Production Pitfall Guard (Combination Operators (`combineLatest`, `forkJoin`, `withLatestFrom`)) • Hands-On Verification (Combination Operators (`combineLatest`, `forkJoin`, `withLatestFrom`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Combination Operators (`combineLatest`, `forkJoin`,...

Purpose: Loading initial reference dropdowns in parallel via `forkJoin` and driving reactive table reloads via `combineLatest([search$, status$, page$])`.

What Topic Is Next → Next Topic 8.8: Resilient Stream Error Recovery (`catchError` and `retry`)

8.8 Resilient Stream Error Recovery (`catchError` and `retry`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `retry({ count: 2 • Delay: 800 })` on transient `502`/`503` GET errors (never blindly retry non-idempotent POSTs!) • Fallback state mapping via `catchError` • Production Pitfall Guard (Resilient Stream Error Recovery (`catchError` and `retry`)) • Hands-On Verification (Resilient Stream Error Recovery (`catchError` and `retry`)) • Resilient Stream Error Recovery (`catchError` and `retry`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Resilient Stream Error Recovery (`catchError` and `r...

Purpose: Retrying transient network hiccups on read-only dashboard analytics endpoints with controlled backoff.

What Topic Is Next → Next Topic 8.9: Multicasting with `Subject`, `BehaviorSubject` and `ReplaySubject`

8.9 Multicasting with `Subject`, `BehaviorSubject` and `ReplaySubject` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Subject<void>` event bus • `BehaviorSubject<T>(initial)` stateful stream with `.getValue()` & instant replay to late subscribers • Exposing read-only `.asObservable()` • Production Pitfall Guard (Multicasting with `Subject`, `BehaviorSubject` and `ReplaySubject`) • Hands-On Verification (Multicasting with `Subject`, `BehaviorSubject` and `ReplaySubject`) • Multicasting with `Subject`, `BehaviorSubject` and `ReplaySubject` — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Multicasting with `Subject`

Purpose: Encapsulating `private readonly userSubject = new BehaviorSubject<User | null>(null)` while exposing `readonly user$ = this.userSubject.asObservable()`.

What Topic Is Next → Next Topic 8.10: Declarative Template Subscriptions with `AsyncPipe` (`| async`)

8.10 Declarative Template Subscriptions with `AsyncPipe` (`| async`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Auto-subscribe on init • Auto-markForCheck under `OnPush` change detection • Auto-unsubscribe on destroy • Single subscription aliasing `@if (vm$ | async • Production Pitfall Guard (Declarative Template Subscriptions with `AsyncPipe` (`| async`)) • Hands-On Verification (Declarative Template Subscriptions with `AsyncPipe` (`| async`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Declarative Template Subscriptions with `AsyncPipe`...

Purpose: Composing a single `vm$` view-model stream (`{ loading, data, error }`) consumed once via `@if (vm$ | async; as vm)` with zero manual `.subscribe()`!

What Topic Is Next → Next Topic 8.11: Signals and RxJS Interoperability (`toSignal` and `toObservable`)

8.11 Signals and RxJS Interoperability (`toSignal` and `toObservable`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@angular/core/rxjs-interop`: converting HTTP/Router Observables to synchronous template Signals via `toSignal(obs$ • { initialValue: [] })` • Debouncing a Signal via `toObservable(query)` • Production Pitfall Guard (Signals and RxJS Interoperability (`toSignal` and `toObservable`)) • Hands-On Verification (Signals and RxJS Interoperability (`toSignal` and `toObservable`)) • Signals and RxJS Interoperability (`toSignal` and `toObservable`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Signals

Purpose: Using `toObservable(this.searchQuery).pipe(debounceTime(300), switchMap(...))` -> `toSignal(...)` to get the best of RxJS async operators AND ergonomic Signal templates!

What Topic Is Next → Next Topic 8.12: Service-Based Reactive State Management (Signals / `BehaviorSubject` Store)

8.12 Service-Based Reactive State Management (Signals / `BehaviorSubject` Store) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When clean Signal/RxJS Store Services cover 85% of feature state needs without full Redux/NgRx ceremony: immutable `.update()` • Computed selectors • Async methods • Production Pitfall Guard (Service-Based Reactive State Management (Signals / `BehaviorSubject` Store)) • Hands-On Verification (Service-Based Reactive State Management (Signals / `BehaviorSubject` Store)) • Service-Based Reactive State Management (Signals / `BehaviorSubject` Store) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Service-Based Reactive State Management (Signals / `...

Purpose: Building a clean 50-line `EmployeeStoreService` with Signals (`employees`, `loading`, `error`, `filteredEmployees`) for medium-sized enterprise modules.

What Topic Is Next → Next Topic 8.13: NgRx Fundamentals and Enterprise Redux Architecture in Angular

8.13 NgRx Fundamentals and Enterprise Redux Architecture in Angular [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@ngrx/store` • `@ngrx/effects` • `@ngrx/store-devtools` • When large banking/ERP Angular teams adopt NgRx (strict action audit log • Production Pitfall Guard (NgRx Fundamentals and Enterprise Redux Architecture in Angular) • Hands-On Verification (NgRx Fundamentals and Enterprise Redux Architecture in Angular) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for NgRx Fundamentals

Purpose: Working productively in enterprise Angular monorepos standardized on `@ngrx/store` + `@ngrx/effects` (or modern `@ngrx/signals` `signalStore`).

What Topic Is Next → Next Topic 8.14: NgRx Centralized Store and Feature State (`createFeature`)

8.14 NgRx Centralized Store and Feature State (`createFeature`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Defining typed `EmployeesState` (`ids` • `entities` • `loading` • `error`) • Production Pitfall Guard (NgRx Centralized Store and Feature State (`createFeature`)) • Hands-On Verification (NgRx Centralized Store and Feature State (`createFeature`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for NgRx Centralized Store

Purpose: Cutting NgRx boilerplate in half with modern `createFeature({ name: 'employees', reducer })`.

What Topic Is Next → Next Topic 8.15: NgRx Actions (`createAction` and `createActionGroup`)

8.15 NgRx Actions (`createAction` and `createActionGroup`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Good Action Hygiene (event naming `[Employees Page] Opened` vs `[Employees API] Load Success`) • Grouping related actions cleanly with `createActionGroup` + `props<T>()` / `emptyProps()` • Production Pitfall Guard (NgRx Actions (`createAction` and `createActionGroup`)) • Hands-On Verification (NgRx Actions (`createAction` and `createActionGroup`)) • NgRx Actions (`createAction` and `createActionGroup`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for NgRx Actions (`createAction` and `createActionGroup`... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Defining `EmployeesActions` (`opened`, `loadSuccess`, `loadFailure`, `deleted`) in 10 concise lines via `createActionGroup`.

What Topic Is Next → Next Topic 8.16: NgRx Pure Reducers (`createReducer` and `on`)

8.16 NgRx Pure Reducers (`createReducer` and `on`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable spread updates `on(EmployeesActions.loadSuccess • (state • { items }) => ({ ...state • Items • Production Pitfall Guard (NgRx Pure Reducers (`createReducer` and `on`)) • Hands-On Verification (NgRx Pure Reducers (`createReducer` and `on`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for NgRx Pure Reducers (`createReducer` and `on`)

Purpose: Unit-testing NgRx state transitions with plain synchronous `expect(reducer(initialState, action)).toEqual(...)` assertions.

What Topic Is Next → Next Topic 8.17: NgRx Side Effects (`createEffect` and `ofType`)

8.17 NgRx Side Effects (`createEffect` and `ofType`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Listening to `actions$.pipe(ofType(EmployeesActions.pageOpened) • SwitchMap(() => api.list().pipe(map(LoadSuccess) • CatchError(err => of(LoadFailure)))))` • `{ dispatch: false }` toast/router effects • Production Pitfall Guard (NgRx Side Effects (`createEffect` and `ofType`)) • Hands-On Verification (NgRx Side Effects (`createEffect` and `ofType`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for NgRx Side Effects (`createEffect` and `ofType`)

Purpose: Handling API requests, toast notifications, and post-login redirects cleanly inside functional `createEffect` streams.

What Topic Is Next → Next Topic 8.18: NgRx Memoized Selectors (`createSelector` & `store.selectSignal`)

8.18 NgRx Memoized Selectors (`createSelector` & `store.selectSignal`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Composing memoized derived selectors (`selectActiveEmployees` • `selectDepartmentCounts`) • Reading selectors as Signals in templates via `this.store.selectSignal(selectActiveEmployees)`! • Production Pitfall Guard (NgRx Memoized Selectors (`createSelector` & `store.selectSignal`)) • Hands-On Verification (NgRx Memoized Selectors (`createSelector` & `store.selectSignal`)) • NgRx Memoized Selectors (`createSelector` & `store.selectSignal`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for NgRx Memoized Selectors (`createSelector` & `store.s...

Purpose: Consuming NgRx state as synchronous Signals via `store.selectSignal(...)` with zero manual `AsyncPipe` boilerplate in templates!

What Topic Is Next → Module 8 Capstone (Week 19 Lab Build: Typed Angular Reactive `FormArray` + RxJS Live Search + NgRx Store) → Next Topic 9.1: Structuring Large Enterprise Angular Applications

Module 8 Hands-On Capstone Project (100 Marks): Week 19 Lab Build: Typed Angular Reactive `FormArray` + RxJS Live Search + NgRx Store — Implement a dynamic multi-item Invoice `FormArray` with custom sync/async validators, an RxJS `debounceTime` + `switchMap` search bar, and a typed NgRx feature slice (`createActionGroup`, `c...

MODULE 9 OF 10 • STAGE 3 • 11 LIVE + LAB HOURS • 14 TOPICS (112 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Enterprise Architecture, Angular Material 3 Tables/Dialogs & CDK Layout

Scale large enterprise Angular applications with Standalone domain architecture, Angular Material & CDK (`MatTable`, `MatPaginator`, `MatSort`, `MatDialog`, `MatSidenav`, `MatMenu`, `MatAutocomplete`, `BreakpointObserver`), `ChangeDetectionStrategy.OnPush`, dynamic component rendering, unit/HTTP testing (`TestBed`, `HttpTestingController`), production build budgets, and build Major Project 5: Enterprise Admin Dashboard using Angular.

Objectives: ✓ Organize large Angular codebases into `core/`, `shared/`, and lazy-loaded `features/` while understanding legacy `FeatureModule` boundaries. • ✓ Build polished enterprise admin interfaces rapidly using Angular Material 3 theming, `MatTable`, `MatPaginator`, `MatSort`, `MatDialog`, `MatSidenav`, `MatAutocomplete`, and CDK `BreakpointObserver`. • ✓ Optimize Angular rendering performance with `ChangeDetectionStrategy.OnPush`, Signals, `AsyncPipe`, and lazy feature chunks. • ✓ Write Angular unit and HTTP service tests using `TestBed` and `HttpTestingController`, verify `ng build` bundle budgets, and deliver **Project 5: Enterprise Admin Dashboard using Angular**.

9.1 Structuring Large Enterprise Angular Applications [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Clean layering: `src/app/core/` (auth • Guards • Interceptors • Layout shell) • Production Pitfall Guard (Structuring Large Enterprise Angular Applications) • Hands-On Verification (Structuring Large Enterprise Angular Applications) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Structuring Large Enterprise Angular Applications

Purpose: Keeping 40+ enterprise screens maintainable with strict one-way dependency rules (`features -> shared/core`, never `shared -> features`!).

What Topic Is Next → Next Topic 9.2: Legacy Feature Modules (`NgModule`) Awareness for Enterprise Maintenance

9.2 Legacy Feature Modules (`NgModule`) Awareness for Enterprise Maintenance [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reading `CommonModule` • `SharedModule` • `CoreModule` • `RouterModule.forRoot` vs `forChild` in pre-v17 Angular codebases • Production Pitfall Guard (Legacy Feature Modules (`NgModule`) Awareness for Enterprise Maintenance) • Hands-On Verification (Legacy Feature Modules (`NgModule`) Awareness for Enterprise Maintenance) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Legacy Feature Modules (`NgModule`) Awareness for En...

Purpose: Confidently onboarding onto an existing bank/insurance Angular codebase that mixes older `NgModule` features with newer Standalone screens.

What Topic Is Next → Next Topic 9.3: Modern Standalone Application Architecture at Scale

9.3 Modern Standalone Application Architecture at Scale [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Route-level providers (`providers: [FeatureStore]` inside `Routes`) • Granular component `imports` arrays keeping lazy chunks lean • Production Pitfall Guard (Modern Standalone Application Architecture at Scale) • Hands-On Verification (Modern Standalone Application Architecture at Scale) • Modern Standalone Application Architecture at Scale — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Modern Standalone Application Architecture at Scale • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Scoping feature state/effects providers directly to `{ path: 'analytics', providers: [...], loadChildren: ... }` so analytics code loads ONLY when visited.

What Topic Is Next → Next Topic 9.4: Designing Reusable Shared UI Components in Angular

9.4 Designing Reusable Shared UI Components in Angular [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Stateless `<app-page-header>` • `<app-stat-kpi-card>` • `<app-empty-state>` • `<app-status-chip>` • Production Pitfall Guard (Designing Reusable Shared UI Components in Angular) • Hands-On Verification (Designing Reusable Shared UI Components in Angular) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Designing Reusable Shared UI Components in Angular

Purpose: Using multi-slot `<ng-content select="[header-actions]">` so every admin screen shares identical spacing, breadcrumbs, and typography.

What Topic Is Next → Next Topic 9.5: Angular Material Setup and Material 3 Custom Theming

9.5 Angular Material Setup and Material 3 Custom Theming [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ng add @angular/material` • Material 3 `mat.theme(...)` SCSS tokens • Typography & density scales (`density: -1` compact enterprise admin density!) • Dark/light theme support • Production Pitfall Guard (Angular Material Setup and Material 3 Custom Theming) • Hands-On Verification (Angular Material Setup and Material 3 Custom Theming) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Angular Material Setup

Purpose: Applying compact `density: -1` on enterprise admin tables and forms so operators see 35% more actionable rows without clutter.

What Topic Is Next → Next Topic 9.6: Angular Material Form Controls (`MatFormField`, `MatInput`, `MatSelect`, `MatDatepicker`)

9.6 Angular Material Form Controls (`MatFormField`, `MatInput`, `MatSelect`, `MatDatepicker`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<mat-form-field appearance="outline">` • `<mat-label>` • `<input matInput>` • `<mat-error>` • Production Pitfall Guard (Angular Material Form Controls (`MatFormField`, `MatInput`, `MatSelect`, `MatDatepicker`)) • Hands-On Verification (Angular Material Form Controls (`MatFormField`, `MatInput`, `MatSelect`, `MatDatepicker`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Angular Material Form Controls (`MatFormField`, `Mat...

Purpose: Building accessible, consistent enterprise Create/Edit forms where `<mat-error>` syncs automatically with Reactive Form touched/invalid states.

What Topic Is Next → Next Topic 9.7: Enterprise Data Tables with `MatTable` and `MatSort`

9.7 Enterprise Data Tables with `MatTable` and `MatSort` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `mat-table [dataSource]` • `ng-container matColumnDef` • `mat-header-cell` • `mat-cell` • Production Pitfall Guard (Enterprise Data Tables with `MatTable` and `MatSort`) • Hands-On Verification (Enterprise Data Tables with `MatTable` and `MatSort`) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Enterprise Data Tables with `MatTable`

Purpose: Rendering sortable, sticky-header enterprise inventory/user/order tables with formatted action menus and zero-results `*matNoDataRow`.

What Topic Is Next → Next Topic 9.8: Confirmation and Form Modals with `MatDialog`

9.8 Confirmation and Form Modals with `MatDialog` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `inject(MatDialog).open<ConfirmDialogComponent • ConfirmData • Boolean>(...)` • `MAT_DIALOG_DATA` • Production Pitfall Guard (Confirmation and Form Modals with `MatDialog`) • Hands-On Verification (Confirmation and Form Modals with `MatDialog`) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Confirmation

Purpose: Opening a reusable typed `<app-confirm-dialog>` returning `Observable<boolean>` (`filter(Boolean)`, `switchMap(() => api.delete(id))`) cleanly!

What Topic Is Next → Next Topic 9.9: Navigation Chrome with `MatSidenav`, `MatToolbar` and `MatMenu`

9.9 Navigation Chrome with `MatSidenav`, `MatToolbar` and `MatMenu` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<mat-sidenav-container>` • Desktop `mode="side"` vs mobile drawer `mode="over"` • Top `<mat-toolbar>` • Accessible row overflow actions `<mat-menu #rowMenu="matMenu">` • Production Pitfall Guard (Navigation Chrome with `MatSidenav`, `MatToolbar` and `MatMenu`) • Hands-On Verification (Navigation Chrome with `MatSidenav`, `MatToolbar` and `MatMenu`) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Navigation Chrome with `MatSidenav`

Purpose: Switching `<mat-sidenav>` automatically from persistent `side` rail on `>= 1024px` desktop to collapsible `over` drawer on tablets/phones.

What Topic Is Next → Next Topic 9.10: Client and Server-Side Pagination with `MatPaginator`

9.10 Client and Server-Side Pagination with `MatPaginator` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<mat-paginator [length]="totalCount()" [pageSize]="pageSize()" [pageSizeOptions]="[10 • 50]" (page)="onPageChange($event)">` • Server-driven `PageEvent` vs client `MatTableDataSource` • Production Pitfall Guard (Client and Server-Side Pagination with `MatPaginator`) • Hands-On Verification (Client and Server-Side Pagination with `MatPaginator`) • Client and Server-Side Pagination with `MatPaginator` — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Client

Purpose: Driving server-paginated REST endpoints (`?page=&limit=`) from `MatPaginator` `(page)` events while resetting `pageIndex = 0` whenever filters change!

What Topic Is Next → Next Topic 9.11: Typeahead Selection with `MatAutocomplete`

9.11 Typeahead Selection with `MatAutocomplete` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `<input matInput [formControl]="assigneeCtrl" [matAutocomplete]="auto">` • `<mat-autocomplete #auto="matAutocomplete" [displayWith]="displayFn">` • Async RxJS filtered options • Production Pitfall Guard (Typeahead Selection with `MatAutocomplete`) • Hands-On Verification (Typeahead Selection with `MatAutocomplete`) • Typeahead Selection with `MatAutocomplete` — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Typeahead Selection with `MatAutocomplete`

Purpose: Selecting an Assignee or Product SKU from 5,000 server records via debounced `MatAutocomplete` with `[displayWith]` label formatting.

What Topic Is Next → Next Topic 9.12: Responsive Layouts with Angular CDK (`BreakpointObserver`)

9.12 Responsive Layouts with Angular CDK (`BreakpointObserver`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@angular/cdk/layout` `BreakpointObserver` • Matching `(max-width: 1023.98px)` into a Signal via `toSignal` • Angular CDK Overlay/Clipboard/DragDrop awareness • Production Pitfall Guard (Responsive Layouts with Angular CDK (`BreakpointObserver`)) • Hands-On Verification (Responsive Layouts with Angular CDK (`BreakpointObserver`)) • Responsive Layouts with Angular CDK (`BreakpointObserver`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Responsive Layouts with Angular CDK (`BreakpointObse...

Purpose: Adapting `MatSidenav` mode and table column visibility reactively via `BreakpointObserver` without manual `window.resize` listeners.

What Topic Is Next → Next Topic 9.13: Advanced Component Interaction (`viewChild`, `contentChildren` & Service Bus)

9.13 Advanced Component Interaction (`viewChild`, `contentChildren` & Service Bus) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Signal queries `viewChild.required(MatPaginator)` • `contentChildren()` • `@HostBinding` / `@HostListener` • Coordinating sibling drawers/tables cleanly • Production Pitfall Guard (Advanced Component Interaction (`viewChild`, `contentChildren` & Service Bus)) • Hands-On Verification (Advanced Component Interaction (`viewChild`, `contentChildren` & Service Bus)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Advanced Component Interaction (`viewChild`, `conten...

Purpose: Querying `readonly paginator = viewChild.required(MatPaginator)` with modern Signal queries instead of nullable `@ViewChild(...) !` assertions.

What Topic Is Next → Next Topic 9.14: Dynamic Component Rendering (`NgComponentOutlet` & `ViewContainerRef`)

9.14 Dynamic Component Rendering (`NgComponentOutlet` & `ViewContainerRef`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Declarative `*ngComponentOutlet="widget.component • Inputs: widget.inputs"` for configurable dashboard widgets vs imperative `ViewContainerRef.createComponent` • Production Pitfall Guard (Dynamic Component Rendering (`NgComponentOutlet` & `ViewContainerRef`)) • Hands-On Verification (Dynamic Component Rendering (`NgComponentOutlet` & `ViewContainerRef`)) • Dynamic Component Rendering (`NgComponentOutlet` & `ViewContainerRef`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Dynamic Component Rendering (`NgComponentOutlet` & `... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Rendering user-customizable dashboard KPI/chart widgets dynamically from a role configuration array via `*ngComponentOutlet`!

What Topic Is Next → Module 9 Capstone (Week 9 Practical Build: Enterprise Architecture, Angular Material 3 Tables/Dialogs & CDK Layout) → Next Topic 10.1: Angular Change Detection Internals (Zone.js vs Zoneless Signals)

Module 9 Hands-On Capstone Project (100 Marks): Week 9 Practical Build: Enterprise Architecture, Angular Material 3 Tables/Dialogs & CDK Layout — Implement and verify a clean, runnable ANGU repository covering Scale large enterprise Angular applications with Standalone domain architecture, Angular Material & CDK (`MatTable`, `MatPagin...

MODULE 10 OF 10 • STAGE 3 • 12 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Angular OnPush Performance, TestBed Unit/HTTP Testing, CI/CD & Enterprise Admin Capstone

Optimize enterprise Angular rendering with `ChangeDetectionStrategy.OnPush` & `@defer`, write `TestBed` and `HttpTestingController` unit tests, enforce production bundle budgets, deploy live with SPA rewrites, and deliver the Enterprise Admin Dashboard Capstone.

Objectives: ✓ Apply `ChangeDetectionStrategy.OnPush` and `@defer (on viewport)` across enterprise dashboards. • ✓ Write isolated Angular component, guard, and `HttpTestingController` service tests. • ✓ Deploy a production `ng build` Angular application live on Vercel/Netlify with GitHub Actions CI and pass Capstone Viva.

10.1 Angular Change Detection Internals (Zone.js vs Zoneless Signals) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How `Default` change detection checks the whole component tree top-down after every async browser event • `ExpressionChangedAfterItHasBeenCheckedError` root cause & fix • Modern Signal-driven direction • Production Pitfall Guard (Angular Change Detection Internals (Zone.js vs Zoneless Signals)) • Hands-On Verification (Angular Change Detection Internals (Zone.js vs Zoneless Signals)) • Angular Change Detection Internals (Zone.js vs Zoneless Signals) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Angular Change Detection Internals (Zone.js vs Zonel...

Purpose: Fixing `NG0100: ExpressionChangedAfterItHasBeenCheckedError` cleanly by deriving state via `computed()` instead of mutating parent state inside `ngAfterViewInit`!

What Topic Is Next → Next Topic 10.2: Performance Optimization with `ChangeDetectionStrategy.OnPush`

10.2 Performance Optimization with `ChangeDetectionStrategy.OnPush` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When an `OnPush` component re-renders ONLY: 1) `@Input` reference changes • 2) component/child DOM event fires • 3) `| async` emits • Or 4) a read `signal()` updates! • Production Pitfall Guard (Performance Optimization with `ChangeDetectionStrategy.OnPush`) • Hands-On Verification (Performance Optimization with `ChangeDetectionStrategy.OnPush`) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Performance Optimization with `ChangeDetectionStrate...

Purpose: Adding `changeDetection: ChangeDetectionStrategy.OnPush` across all admin cards, rows, and pages so typing in a filter skips checking 95% of the tree!

What Topic Is Next → Next Topic 10.3: Lazy Loading Feature Areas and Deferrable Views (`@defer`)

10.3 Lazy Loading Feature Areas and Deferrable Views (`@defer`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Route-level `loadChildren` + Angular template `@defer (on viewport)` / `@placeholder` / `@loading` lazy-loading below-the-fold heavy analytics charts! • Production Pitfall Guard (Lazy Loading Feature Areas and Deferrable Views (`@defer`)) • Hands-On Verification (Lazy Loading Feature Areas and Deferrable Views (`@defer`)) • Lazy Loading Feature Areas and Deferrable Views (`@defer`) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Lazy Loading Feature Areas • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Wrapping a heavy Revenue Chart component in `@defer (on viewport)` with `@placeholder` so chart JS downloads ONLY when scrolled into view!

What Topic Is Next → Next Topic 10.4: Angular Component Unit Testing (`TestBed` and `ComponentFixture`)

10.4 Angular Component Unit Testing (`TestBed` and `ComponentFixture`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `TestBed.configureTestingModule({ imports: [StandaloneComp] })` • `fixture.componentRef.setInput(...)` • `fixture.detectChanges()` • Querying accessible DOM & verifying emitted `@Output`s • Production Pitfall Guard (Angular Component Unit Testing (`TestBed` and `ComponentFixture`)) • Hands-On Verification (Angular Component Unit Testing (`TestBed` and `ComponentFixture`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Angular Component Unit Testing (`TestBed` and `Compo...

Purpose: Writing fast, isolated unit tests for Standalone Components, Reactive Form validation rules, and custom Pipes.

What Topic Is Next → Next Topic 10.5: Testing Angular HTTP Services & Interceptors (`provideHttpClientTesting`)

10.5 Testing Angular HTTP Services & Interceptors (`provideHttpClientTesting`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `provideHttpClient()` + `provideHttpClientTesting()` • `inject(HttpTestingController)` • `httpMock.expectOne('/api/users')` • Asserting HTTP method/headers (`Authorization`) • Production Pitfall Guard (Testing Angular HTTP Services & Interceptors (`provideHttpClientTesting`)) • Hands-On Verification (Testing Angular HTTP Services & Interceptors (`provideHttpClientTesting`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Testing Angular HTTP Services

Purpose: Proving `EmployeeApiService` sends exact query params and `authInterceptor` attaches `Bearer <token>` without hitting any live backend!

What Topic Is Next → Next Topic 10.6: Angular Production Build Optimization & Bundle Budgets

10.6 Angular Production Build Optimization & Bundle Budgets [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ng build --configuration production` (esbuild + AOT template compilation + minification + dead-code elimination) • `angular.json` `budgets` (`initial` warning/error KB thresholds) • Production Pitfall Guard (Angular Production Build Optimization & Bundle Budgets) • Hands-On Verification (Angular Production

Build Optimization & Bundle Budgets) • Angular Production Build Optimization & Bundle Budgets — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Angular Production Build Optimization • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Enforcing `budgets` in `angular.json` so CI immediately flags any PR that accidentally balloons the initial JS bundle!

What Topic Is Next → Next Topic 10.7: Deploying Angular SPAs (Vercel / Netlify Rewrite Rules)

10.7 Deploying Angular SPAs (Vercel / Netlify Rewrite Rules) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Output directory `dist/<project>/browser` • Configuring SPA fallback rewrite `/* -> /index.html (200)` so refreshing `/admin/users` deep links never returns host `404`! • Production Pitfall Guard (Deploying Angular SPAs (Vercel / Netlify Rewrite Rules)) • Hands-On Verification (Deploying Angular SPAs (Vercel / Netlify Rewrite Rules)) • Deploying Angular SPAs (Vercel / Netlify Rewrite Rules) — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Deploying Angular SPAs (Vercel / Netlify Rewrite Rul... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Adding `vercel.json` / `public/_redirects` SPA rewrite rules so every client-side Angular route works on direct browser refresh.

What Topic Is Next → Next Topic 10.8: Major Project 5 Integration: Enterprise Admin Dashboard in Angular

10.8 Major Project 5 Integration: Enterprise Admin Dashboard in Angular [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Delivering complete Angular Admin Dashboard (Auth + Guards + Interceptors • KPI Overview • Users/Products/Orders `MatTable` CRUD + Search/Filter/Pagination • Reactive Forms • Production Pitfall Guard (Major Project 5 Integration: Enterprise Admin Dashboard in Angular) • Hands-On Verification (Major Project 5 Integration: Enterprise Admin Dashboard in Angular) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Major Project 5 Integration

Purpose: Demonstrating side-by-side proficiency in both React (Project 4) and Enterprise Angular (Project 5) in your GitHub portfolio!

What Topic Is Next → Next Topic 10.9: Atomic Commits (`git commit`) and Commit Hygiene

10.9 Atomic Commits (`git commit`) and Commit Hygiene [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: One logical change per commit • Imperative subject (`feat(cart): apply coupon discount`) • Explaining *why* in the commit body • `--amend` on unpushed local commits only • Production Pitfall Guard (Atomic Commits (`git commit`) and Commit Hygiene) • Hands-On Verification (Atomic Commits (`git commit`) and Commit Hygiene) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Atomic Commits (`git commit`)

Purpose: Making code reviews 3x faster and rollbacks surgical when each commit compiles and passes tests independently.

What Topic Is Next → Next Topic 10.10: Inspecting History (`git log`, `git show`, `git diff`, `git blame`)

10.10 Inspecting History (`git log`, `git show`, `git diff`, `git blame`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git log --oneline --graph --decorate` • Filtering by file `git log -p -- src/cartSlice.ts` • `git show <sha>` • `git blame` context lookup • Production Pitfall Guard (Inspecting History (`git log`, `git show`, `git diff`, `git blame`)) • Hands-On Verification (Inspecting History (`git log`, `git show`, `git diff`, `git blame`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Inspecting History (`git log`, `git show`, `git diff...

Purpose: Checking `git log -p -- src/services/api.ts` to understand why a teammate added a custom timeout header 3 months ago before deleting it.

What Topic Is Next → Next Topic 10.11: Branching (`git branch`, `git switch -c`)

10.11 Branching (`git branch`, `git switch -c`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lightweight pointer mechanics of Git branches • Naming conventions (`feat/wishlist-drawer` • `fix/safari-modal-scroll` • `chore/ci-pipeline`) • Production Pitfall Guard (Branching (`git branch`, `git switch -c`)) • Hands-On Verification (Branching (`git branch`, `git switch -c`)) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Branching (`git branch`, `git switch -c`)

Purpose: Creating short-lived (< 2 day) feature branches off fresh `main` for every ticket.

What Topic Is Next → Next Topic 10.12: Merging and Resolving Merge Conflicts Calmly

10.12 Merging and Resolving Merge Conflicts Calmly [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fast-forward vs 3-way merge commits • Reading `<<<<<<< HEAD` • `=======` • `>>>>>>> feat/...` conflict markers • Production Pitfall Guard (Merging and Resolving Merge Conflicts Calmly) • Hands-On Verification (Merging and Resolving Merge Conflicts Calmly) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Merging

Purpose: Resolving conflicting edits in `AppRoutes.tsx` and `package.json` cleanly, then running `npm test && npm run build` before `git commit`.

What Topic Is Next → Next Topic 10.13: Rebasing Fundamentals (`git rebase main`) and Golden Rule

10.13 Rebasing Fundamentals (`git rebase main`) and Golden Rule [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Replaying your local feature commits linearly on top of latest `origin/main` • `git rebase --continue` / `--abort` • Golden Rule: NEVER rebase shared public `main`! • Production Pitfall Guard (Rebasing Fundamentals (`git rebase main`) and Golden Rule) • Hands-On Verification (Rebasing Fundamentals (`git rebase main`) and Golden Rule) • Rebasing Fundamentals (`git rebase main`) and Golden Rule — Interview Readiness • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Rebasing Fundamentals (`git rebase main`)

Purpose: Updating your local `feat/search` branch onto latest `origin/main` before opening a Pull Request for a crisp, linear diff.

What Topic Is Next → Next Topic 10.14: Saving Work-in-Progress with `git stash`

10.14 Saving Work-in-Progress with `git stash` [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git stash push -u -m "wip: checkout address form"` • `git stash list` • `git stash pop` vs `apply` • `git stash drop` • Production Pitfall Guard (Saving Work-in-Progress with `git stash`) • Hands-On Verification (Saving Work-in-Progress with `git stash`) • Idiomatic ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Saving Work-in-Progress with `git stash`

Purpose: Pausing unfinished feature edits (`-u` including untracked new files!) for 10 minutes to fix an urgent production P0 hotfix on `main`.

What Topic Is Next → Next Topic 10.15: Deploying Frontend Applications to Vercel & Netlify (with SPA Rewrites)

10.15 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Deploying Frontend Applications to Vercel

Purpose: Ensuring direct navigation or F5 refresh on `https://your-app.vercel.app/products/laptops` serves `index.html` (200) instead of a 404 page!

What Topic Is Next → Next Topic 10.16: Custom Domains, DNS (`A` / `CNAME`) and Automatic HTTPS/TLS

10.16 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Custom Domains

Purpose: Remembering to add your live `https://shop.yourdomain.com` origin to the backend API's CORS allowlist when moving off `localhost:5173`!

What Topic Is Next → Next Topic 10.17: Managing Production, Preview and Local Environment Variables

10.17 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for Managing Production

Purpose: Pointing PR Preview deploys at a safe Staging/Sandbox API while `main` Production deploys talk to the Production API.

What Topic Is Next → Next Topic 10.18: GitHub Actions CI/CD Pipeline & Production Error Monitoring (Sentry Basics)

10.18 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 ANGU Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Production Runtime Execution & Memory Lifecycle for GitHub Actions CI/CD Pipeline

Purpose: Running automated lint + TypeScript + Vitest + production build verification on every Pull Request AND capturing uncaught production browser exceptions with stack traces!

What Topic Is Next → Module 10 Capstone → End-of-Course Live Client Project: Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) — Enterprise Client Production Platform

Module 10 Hands-On Capstone Project (100 Marks): Project 5: Enterprise Admin Dashboard using Angular & Angular Material — Develop a modular, role-protected Enterprise Back-Office Admin Dashboard using modern Standalone Angular, TypeScript 5, Signals, RxJS, Reactive Forms (`FormGroup` & `FormArray`), Functional...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

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

Project: Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) — Enterprise Client Production Platform •

Confirmed Role: Complete Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Engineering Project Intern

Domain & Production Infra: Live Enterprise Enterprise Frontend & TypeScript SPA Client Engagement (HTML5 & Accessible Forms, CSS3, Flexbox, Grid & SCSS, JavaScript ES6+ & Async/DOM, TypeScript 5 & Decorators, Angular (Standalone + Signals) • Git CI/CD + Automated Testing + Live HTTPS Cloud Production)

- Translated live client functional and non-functional requirements into modular HTML5 & Accessible Forms, CSS3, Flexbox, Grid & SCSS, JavaScript ES6+ & Async/DOM, TypeScript 5 & Decorators, Angular (Standalone + Signals) 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 Angular Developer Course (with HTML5, CSS3/SCSS, JS & TypeScript) Developer, Associate Enterprise Frontend & TypeScript SPA 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/angular-complete-developer-training

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