Admissions & Support:
support@mockattempt.com
Phone / WhatsApp: +91 98765 43210
Course Code: ANGL • **Updated:** 04 Nov 2026
https://mockattempt.com/courses/angular-training

ANGL • ENTERPRISE FRONTEND & TYPESCRIPT SPA **BEGINNER TO SENIOR ANGULAR ARCHITECT**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

Angular Training

Master Enterprise Angular from Ivy AOT Compilation, CLI Workspaces, Standalone Bootstrapping (`bootstrapApplication`) & Dependency Injection (`inject()`) to Structural Directives (`*ngIf`/`@if`, `*ngFor`/`@for` track), Signals, Data Binding, Typed Reactive Forms & Full-Stack Spring Boot JWT CRUD — with TypeScript programs, 244 Topic Mock Tests (10,440 MCQs), and Spoken Voice + MockAttempt AI Interviews at ₹2,999 (Incl. 18% GST).

Duration: 15–17 Weeks • Live + Hands-On Projects (208 Hrs) • **Modules:** 16 • **Live Topics:** 42 • **Subtopics:** 336+ • **Labs & Tasks:** 84 • **Mocks / AI Vivas:** 244 / 244 • **Projects:** 16 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹2,999 (Net Tuition ₹2,542 + Included 18% GST ₹457) • **Split Option:** 100% Credit to Job Track • **Effective ROI:** ~₹14/Guided Hr (~₹71/Live Topic)

Faculty: 100% Active Working IT Professionals • **Live Batch Tracks (7:00 AM – 8:00 PM IST Flexible Slots):** Weekday Live Online Batch (Mon – Fri • 1 Hr/Day • 7:00 AM – 8:00 PM) (Monday to Friday (1 Hour Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 1-Hour Slot Based on Trainer Availability) • 17–19 Weeks Fast-Track) | Weekend Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Slot Based on Trainer Availability) • 19–21 Weeks Weekend Track)

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

Core Technologies & Engineering Stack: Angular (Ivy AOT & Standalone APIs) • Fine-Grained Angular Signals • TypeScript 5 Strict Mode • Built-in Control Flow & Directives • RxJS Observables & Operators • Strictly Typed Reactive Forms • HttpClient, Interceptors & Route Guards

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Architect modular Angular SPAs using Standalone Components, hierarchical Dependency Injection (`inject()`), Ivy AOT tree-shaking, and lazy-loaded routes (`loadComponent`)
2. Eliminate unnecessary DOM re-renders using `ChangeDetectionStrategy.OnPush`, Angular Signals (`signal`, `computed`, `model`), and `trackBy` / `@for (...; track item.id)`
3. Build strongly typed `FormGroup` Reactive Forms, custom validators, and full-stack CRUD + JWT `HttpInterceptorFn` authentication pipelines with Spring Boot REST APIs
4. Clear 244 Timed Mock Exams (10,000 Angular MCQs) and 200 Spoken Voice + MockAttempt AI Frontend & Full-Stack Angular Interviews
5. Standalone component architecture (NgModule-free), fine-grained Angular Signals, and `OnPush` change detection.
6. Crack LeetCode and HackerRank problems in TypeScript 5 with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
7. Reactive Forms with typed form controls, cross-field async validators, and reusable dynamic form builders.
8. Clean RxJS reactive data flows (`switchMap`, `catchError`, `takeUntilDestroyed`), functional HTTP interceptors, and guards.

Eligibility & Prerequisites: Basic HTML5, CSS3, and modern JavaScript (ES6+ arrow functions, classes, modules) — teaches TypeScript & Angular architecture from scratch • Any laptop/PC with Node.js LTS, npm, `@angular/cli` (`ng`), and VS Code with Angular Language Service • No prior Angular industry experience required — Module 1 starts from environment setup, core mental models, and plain-English fundamentals. • 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 live Weekday or Weekend classes, complete daily hands-on lab checklists, and practice topic-wise Mock Tests & AI Interviews.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (16 MODULES • 42 LIVE TOPICS • 336+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Stage 1: Day-1 Foundations & Core Mental Models (8 Topics / 64 Subtopics) • Modules 5–8: Stage 2: Applied Engineering, Data & API Workflows (8 Topics / 64 Subtopics) • Modules 9–12: Stage 3: Internals, Concurrency, Security & Performance (12 Topics / 96 Subtopics) • Modules 13–16: Stage 4: Production Architecture, Testing, CI/CD & Expert Mastery (14 Topics / 112 Subtopics)

MODULE 1 OF 16 • STAGE 1 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Angular Platform Foundations, SPA Evolution & Opinionated Framework Design

Understand Angular's evolution from AngularJS (1.x) to modern TypeScript-based Angular, Single-Page Application routing, and batteries-included enterprise conventions.

Objectives: ✓ Angular Platform Overview — TypeScript SPA Evolution, Semantic Versioning & Batteries-Included Ecosystem — How Angular provides a cohesive first-party platform combining Router, HttpClient, Forms, DI, CLI, and Signals under synchronized major releases. • ✓ Angular Introduction — Component Trees, Declarative Templates & Zone vs Zoneless Reactivity — Structuring enterprise UIs as encapsulated component trees where component state drives declarative HTML templates deterministically. • ✓ Build, verify, and ship Module Capstone: Reactive Course Catalog Summary Engine with Angular Signals.

1.1 Angular Platform Overview — TypeScript SPA Evolution, Semantic Versioning & Batteries-Included Ecosystem

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: AngularJS 1.x vs modern TypeScript Angular & Single-Page Application (SPA) lifecycle • Core Syntax, CLI & Production APIs: @Component({ standalone: true, ... }) + signal() + computed() • Deep-Dive Mechanics: First-party Router/Forms/HttpClient cohesion & Ahead-of-Time (AOT) template compilation • Advanced Patterns: Angular Signals reactive primitives & Immutable signal.update() state updates • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Angular `signal()` & Memoized `computed()` State Model • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular Platform Overview

Purpose: To understand and apply Angular Platform Overview — TypeScript SPA Evolution, Semantic Versioning & Batteries-Included Ecosystem in real Angular systems — specifically: H...

What Topic Is Next → Next Topic 1.2: Angular Introduction — Component Trees, Declarative Templates & Zone vs Zoneless Reactivity

1.2 Angular Introduction — Component Trees, Declarative Templates & Zone vs Zoneless Reactivity

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Encapsulated component tree hierarchy & ChangeDetectionStrategy.OnPush optimization • Core Syntax, CLI & Production APIs: @Component({ selector, standalone: true, template, changeDetection: ChangeDetectionStrategy.... • Deep-Dive Mechanics: Zone.js vs Zoneless change detection & Shadow DOM / Emulated view encapsulation • Advanced Patterns: Avoiding function calls in templates & Pure @Pipe memoization • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Pure `@Pipe` Formatter vs Expensive Template Recalculation • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular Introduction

Purpose: To understand and apply Angular Introduction — Component Trees, Declarative Templates & Zone vs Zoneless Reactivity in real Angular systems — specifically: Structuring en...

What Topic Is Next → Module 1 Capstone (Reactive Course Catalog Summary Engine with Angular Signals) → Next Topic 2.1: Core Angular Compiler & Runtime Features — Ivy Tree-Shaking, Dynamic Lazy Imports & Web Workers

Module 1 Hands-On Capstone Project (100 Marks): Reactive Course Catalog Summary Engine with Angular Signals — Build an Angular state model using `signal()` and `computed()` that filters published courses by track and derives live enrollment totals without manual imperative recalculation.

MODULE 2 OF 16 • STAGE 1 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Ivy Compiler Internals, Lazy Loading & Angular CLI Workspace Tooling

Explore Ivy tree-shakable locality compilation, dynamic ES `import()` route lazy loading, and Angular CLI (`angular.json`, budgets, environments, esbuild/Vite).

Objectives: ✓ Core Angular Compiler & Runtime Features — Ivy Tree-Shaking, Dynamic Lazy Imports & Web Workers — How Angular's Ivy instruction set, differential/modern ES module output, dynamic `import()` lazy routing, and Web Worker offloading keep enterprise SPAs fast. • ✓ Angular CLI & Workspace Architecture — Node LTS, `@angular/cli`, `angular.json` & `tsconfig` — Setting up reproducible Angular workspaces with Node.js LTS, `@angular/cli`, schematic generators (`ng g c`), environment replacements, and CI bundle budgets. • ✓ Build, verify, and ship Module Capstone: Lazy-Loaded Enterprise Route Map & CLI Bundle Budget Guard.

2.1 Core Angular Compiler & Runtime Features — Ivy Tree-Shaking, Dynamic Lazy Imports & Web Workers

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Ivy locality principle & tree-shaking & Dynamic import() route code-splitting • Core Syntax, CLI & Production APIs: export const routes: Routes = [{ path: 'admin', loadComponent: () => import('./admin.compone... • Deep-Dive Mechanics: LoadComponent vs eager component routing & StrictTemplates compile-time checking • Advanced Patterns: Web Worker CPU-heavy thread offloading & Esbuild + Vite Angular application builder • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Typed Angular Lazy-Loaded `Routes` Configuration • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Core Angular Compiler

Purpose: To understand and apply Core Angular Compiler & Runtime Features — Ivy Tree-Shaking, Dynamic Lazy Imports & Web Workers in real Angular systems — specifically: How Angula...

What Topic Is Next → Next Topic 2.2: Angular CLI & Workspace Architecture — Node LTS, `@angular/cli`, `angular.json` & `tsconfig`

2.2 Angular CLI & Workspace Architecture — Node LTS, `@angular/cli`, `angular.json` & `tsconfig`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Node.js LTS + @angular/cli toolchain & Angular.json build & budget targets • Core Syntax, CLI & Production APIs: ng new academy-web --standalone --routing --strict + angular.json budgets • Deep-Dive Mechanics: StrictTemplates in angularCompilerOptions & FileReplacements environment configuration • Advanced Patterns: Client bundle public visibility rule & Ng generate schematics standardization • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Angular CLI Bundle Budget Threshold Evaluator • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms &

HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular CLI

Purpose: To understand and apply Angular CLI & Workspace Architecture — Node LTS, `@angular/cli`, `angular.json` & `tsconfig` in real Angular systems — specifically: Setting up re...

What Topic Is Next → Module 2 Capstone (Lazy-Loaded Enterprise Route Map & CLI Bundle Budget Guard) → Next Topic 3.1: First Angular Application — `@Component` Metadata, Selectors, Standalone `imports` & Lifecycle Hooks

Module 2 Hands-On Capstone Project (100 Marks): Lazy-Loaded Enterprise Route Map & CLI Bundle Budget Guard — Configure a strongly typed Angular `Routes` table using dynamic `loadComponent` lazy chunks and enforce `angular.json` initial bundle size budgets + `strictTemplates`.

MODULE 3 OF 16 • STAGE 1 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: First Angular App Anatomy & Application Bootstrapping Pipeline

Trace every step from `index.html` (`<app-root>`) and `main.ts` (`bootstrapApplication` / `AppConfig`) to root component mounting and lifecycle initialization.

Objectives: ✓ First Angular Application — `@Component` Metadata, Selectors, Standalone `imports` & Lifecycle Hooks — Anatomy of an Angular root/feature component: `@Component` decorator metadata, CSS element selector (`app-root`), standalone `imports`, and `ngOnInit` vs `constructor`. • ✓ How an Angular App Loads & Starts — `index.html`, `main.ts`, `bootstrapApplication` & `appConfig` — Step-by-step browser boot sequence: builder entry point `main.ts` -> environment/ApplicationConfig providers -> `bootstrapApplication` (or `platformBrowserDynamic`) -> `<app-root>` hydration. • ✓ Build, verify, and ship Module Capstone: Standalone Shell Bootstrapper with `appConfig` Providers & `DestroyRef` Cleanup.

3.1 First Angular Application — `@Component` Metadata, Selectors, Standalone `imports` & Lifecycle Hooks [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: @Component decorator metadata & Standalone: true explicit imports[] • Core Syntax, CLI & Production APIs: @Component({ selector: 'app-root', standalone: true, imports: [RouterOutlet], template: ..... • Deep-Dive Mechanics: Constructor vs ngOnInit input timing & NgOnChanges SimpleChanges tracking • Advanced Patterns: DestroyRef / ngOnDestroy teardown & Scoped component CSS encapsulation • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Angular Component Lifecycle Execution Order Verifier • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for First Angular Application

Purpose: To understand and apply First Angular Application — `@Component` Metadata, Selectors, Standalone `imports` & Lifecycle Hooks in real Angular systems — specifically: Anato...

What Topic Is Next → Next Topic 3.2: How an Angular App Loads & Starts — `index.html`, `main.ts`, `bootstrapApplication` & `appConfig`

3.2 How an Angular App Loads & Starts — `index.html`, `main.ts`, `bootstrapApplication` & `appConfig` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Angular.json browser/main entry point & Main.ts bootstrapApplication() call • Core Syntax, CLI & Production APIs: bootstrapApplication(AppComponent, { providers: [provideRouter(routes), provideHttpClient()]... • Deep-Dive Mechanics: ApplicationConfig tree-shakable providers & Index.html <app-root> host element match • Advanced Patterns: APP_INITIALIZER / provideAppInitializer startup & SSR non-destructive client hydration • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Angular Boot Pipeline & Selector Mount Validator • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for How an Angular App Loads

Purpose: To understand and apply How an Angular App Loads & Starts — `index.html`, `main.ts`, `bootstrapApplication` & `appConfig` in real Angular systems — specifically: Step-by-...

What Topic Is Next → Module 3 Capstone (Standalone Shell Bootstrapper with `appConfig` Providers & `DestroyRef` Cleanup) → Next Topic 4.1: Angular Architecture — Components, Templates, `@Injectable` Services & Hierarchical DI (`inject()`)

Module 3 Hands-On Capstone Project (100 Marks): Standalone Shell Bootstrapper with `appConfig` Providers & `DestroyRef` Cleanup — Create an Angular startup pipeline wiring `provideRouter(routes)` and `provideHttpClient(withFetch())` in `appConfig` and verifying deterministic `DestroyRef` subscription cleanup.

MODULE 4 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Hierarchical Dependency Injection Architecture & Custom Directives

Master Angular's building blocks (`@Injectable`, `providedIn: "root"`, `inject()`, Services) and distinguish Component, Attribute, and Structural Directives.

Objectives: ✓ Angular Architecture — Components, Templates, `@Injectable` Services & Hierarchical DI (`inject()`) — How Angular separates presentation (`@Component`) from business/data state (`@Injectable` Services) wired together through a multi-level Hierarchical Dependency Injector. • ✓ Angular Directives — Component vs Structural vs Attribute Directives, `@HostBinding` & `@HostListener` — Comparing the 3 Angular directive categories: Components (directives with templates), Structural Directives (altering DOM layout), and Attribute Directives (`ngClass`, `ngStyle`, custom `[app*]`). • ✓ Build, verify, and ship Module Capstone: Hierarchical Theme Service & Role-Aware Attribute Directive Suite.

4.1 Angular Architecture — Components, Templates, `@Injectable` Services & Hierarchical DI (`inject()`) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Component vs Service separation of concerns & @Injectable({ providedIn: 'root' }) singleton • Core Syntax, CLI & Production APIs: @Injectable({ providedIn: 'root' }) + private readonly api = inject(CatalogService) • Deep-Dive Mechanics: Inject() functional dependency injection & Hierarchical ElementInjector vs EnvironmentInjector • Advanced Patterns: InjectionToken<T> for non-class config values & RxJS BehaviorSubject / Signal state services • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Root Singleton vs Component-Scoped Injector Resolution Simulator • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular Architecture

Purpose: To understand and apply Angular Architecture — Components, Templates, `@Injectable` Services & Hierarchical DI (`inject()`) in real Angular systems — specifically: How An...

What Topic Is Next → Next Topic 4.2: Angular Directives — Component vs Structural vs Attribute Directives, `@HostBinding` & `@HostListener`

4.2 Angular Directives — Component vs Structural vs Attribute Directives, `@HostBinding` & `@HostListener`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: 3 directive types (Component, Structural, Attribute) & [ngClass] and [ngStyle] built-in attribute directives • Core Syntax, CLI & Production APIs: @Directive({ selector: '[appBadge]', standalone: true }) + @HostBinding + @HostListener • Deep-Dive Mechanics: @HostBinding declarative host property binding & @HostListener host event subscription • Advanced Patterns: Renderer2 SSR-safe DOM abstraction & Directive Composition API (hostDirectives) • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Attribute Directive Host Class & Hover State Resolver • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular Directives

Purpose: To understand and apply Angular Directives — Component vs Structural vs Attribute Directives, `@HostBinding` & `@HostListener` in real Angular systems — specifically: Com...

What Topic Is Next → Module 4 Capstone (Hierarchical Theme Service & Role-Aware Attribute Directive Suite) → Next Topic 5.1: Angular `*ngIf` & `@if` Control Flow — DOM Lifecycle Destruction, `else` Templates & `async` Aliasing

Module 4 Hands-On Capstone Project (100 Marks): Hierarchical Theme Service & Role-Aware Attribute Directive Suite — Build a `providedIn: "root"` state service consumed via `inject()` alongside a reusable Attribute Directive (`[appHighlightRole]`) using `@HostBinding` and `@HostListener`.

MODULE 5 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Conditional & List Rendering — `*ngIf` / `@if` and `*ngFor` / `@for` (`track`)

Control DOM creation/destruction with `*ngIf` (and modern `@if` / `@else`) and render 10,000-row lists cleanly with `*ngFor` `trackBy` and `@for (...; track item.id)`.

Objectives: ✓ Angular `*ngIf` & `@if` Control Flow — DOM Lifecycle Destruction, `else` Templates & `async` Aliasing — Physically mounting and unmounting component subtrees (and firing `ngOnInit`/`ngOnDestroy`) via `*ngIf="user$ | async as user; else loading"` and modern `@if (user(); as u) { ... } @else { ... }`. • ✓ Angular `*ngFor` & `@for` — `trackBy` / Mandatory `track` Diffing, `@empty` & Context Variables — Rendering dynamic collections at 60 FPS using identity-based reconciliation (`@for (row of rows(); track row.id) { ... } @empty { ... }`) and context variables (`$index`, `$first`, `$last`, `$count`). • ✓ Build, verify, and ship Module Capstone: High-Throughput Exam Leaderboard with `@if` Skeleton States & `track` Identity Diffing.

5.1 Angular `*ngIf` & `@if` Control Flow — DOM Lifecycle Destruction, `else` Templates & `async` Aliasing

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: *ngIf vs [hidden] DOM lifecycle difference & Asterisk * microsyntax <ng-template> desugaring • Core Syntax, CLI & Production APIs: @if (user$ | async; as user) { ... } @else { ... } /* and *ngIf="cond; else tpl" */ • Deep-Dive Mechanics: Modern @if / @else if / @else blocks & Async pipe single-subscription aliasing • Advanced Patterns: <ng-container> zero-DOM-node grouping & Two-structural-directives-per-element ban • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: `@if` vs `[hidden]` Component Memory & Subscription Simulator • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular `*ngIf`

Purpose: To understand and apply Angular `*ngIf` & `@if` Control Flow — DOM Lifecycle Destruction, `else` Templates & `async` Aliasing in real Angular systems — specifically: Phys...

What Topic Is Next → Next Topic 5.2: Angular `*ngFor` & `@for` — `trackBy` / Mandatory `track` Diffing, `@empty` & Context Variables

5.2 Angular `*ngFor` & `@for` — `trackBy` / Mandatory `track` Diffing, `@empty` & Context Variables

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Mandatory @for (...; track item.id) syntax & *ngFor trackBy identity function • Core Syntax, CLI & Production APIs: @for (item of items(); track item.id; let idx = $index) { ... } @empty { ... } • Deep-Dive Mechanics: DOM node preservation on immutable refresh & @empty built-in zero-state block • Advanced Patterns: $index, $first, $last, $even, $odd, $count & Why track $index breaks stateful inputs • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Keyed `track item.id` vs Reference-Identity DOM Reuse Counter • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular `*ngFor`

Purpose: To understand and apply Angular `*ngFor` & `@for` — `trackBy` / Mandatory `track` Diffing, `@empty` & Context Variables in real Angular systems — specifically: Rendering...

What Topic Is Next → Module 5 Capstone (High-Throughput Exam Leaderboard with `@if` Skeleton States & `track` Identity Diffing) → Next Topic 6.1: Angular `[ngSwitch]` & `@switch` — Multi-Branch Strict Equality (`===`), `@case` & `@default`

Module 5 Hands-On Capstone Project (100 Marks): High-Throughput Exam Leaderboard with `@if` Skeleton States & `track` Identity Diffing — Build an Angular leaderboard template & view-model handling loading/empty/error states via `@if` / `@else` and diffing live score refreshes via stable primary-key `track` identities.

MODULE 6 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Multi-Branch Switching (`ngSwitch` / `@switch`) & Angular Data-Binding Architecture

Render clean multi-branch state machines via `[ngSwitch]` / `@switch` (`===`) and master all 4 directions of Angular's data-binding mental model.

Objectives: ✓ Angular `[ngSwitch]` & `@switch` — Multi-Branch Strict Equality (`===`), `@case` & `@default` — Declarative multi-branch view switching across status enums, role dashboards, and wizard steps via `[ngSwitch]` / `*ngSwitchCase` and modern `@switch (state()) { @case (...) @default }`. • ✓ Data Binding in Angular — The 4 Binding Modes, Unidirectional Flow & `NG0100` Prevention — Unifying Interpolation `{{ }}`, Property Binding `[ ]`, Event Binding `( )`, and Two-Way Binding `[( )]` under Angular's top-down unidirectional data-flow guarantee. • ✓ Build, verify, and ship Module Capstone: Multi-Stage AI Interview Status View with Exhaustive `@switch` & Unidirectional Flow.

6.1 Angular `[ngSwitch]` & `@switch` — Multi-Branch Strict Equality (`===`), `@case` & `@default` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: @switch / @case / @default control flow & [ngSwitch] + *ngSwitchCase + *ngSwitchDefault • Core Syntax, CLI & Production APIs: @switch (status()) { @case ('ready') { ... } @default { ... } } • Deep-Dive Mechanics: Strict === comparison without fallthrough & Discriminated union state modeling • Advanced Patterns: Exclusive single-branch DOM mounting & Compile-time type narrowing inside @case • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Discriminated Union State Machine for `@switch` Views • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular `[ngSwitch]`

Purpose: To understand and apply Angular `[ngSwitch]` & `@switch` — Multi-Branch Strict Equality (`===`), `@case` & `@default` in real Angular systems — specifically: Declarative...

What Topic Is Next → Next Topic 6.2: Data Binding in Angular — The 4 Binding Modes, Unidirectional Flow & `NG0100` Prevention

6.2 Data Binding in Angular — The 4 Binding Modes, Unidirectional Flow & `NG0100` Prevention [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Component -> DOM: {{ }} and [property] & DOM -> Component: (event) binding • Core Syntax, CLI & Production APIs: {{ expr }} | [target]="expr" | (event)="stmt($event)" | [(model)]="sig" • Deep-Dive Mechanics: Two-Way banana-in-a-box [( )] & Top-down unidirectional data flow • Advanced Patterns: NG0100 ExpressionChangedAfterChecked diagnosis & DOM property vs HTML attribute distinction • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Angular 4-Mode Binding Direction Classifier • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Data Binding in Angular

Purpose: To understand and apply Data Binding in Angular — The 4 Binding Modes, Unidirectional Flow & `NG0100` Prevention in real Angular systems — specifically: Unifying Interpol...

What Topic Is Next → Module 6 Capstone (Multi-Stage AI Interview Status View with Exhaustive `@switch` & Unidirectional Flow) → Next Topic 7.1: Property Binding `[ ]` — DOM Properties vs HTML Attributes, Boolean `[disabled]` & `[attr.*]`

Module 6 Hands-On Capstone Project (100 Marks): Multi-Stage AI Interview Status View with Exhaustive `@switch` & Unidirectional Flow — Build an Angular status view-model switching cleanly across `"idle" | "listening" | "verifying" | "scored"` union states while keeping parent-to-child data flow strictly unidirectional.

MODULE 7 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: DOM Property Binding (`[ ]`), Accessibility Attributes & XSS-Safe Interpolation (`{{ }}`)

Master boolean/object DOM Property Binding (`[disabled]`, `[attr.aria-*]`, `[class.active]`) vs string Interpolation (`{{ }}`) and Angular's built-in XSS sanitization engine.

Objectives: ✓ Property Binding `[ ]` — DOM Properties vs HTML Attributes, Boolean `[disabled]` & `[attr.*]` — Binding live JavaScript booleans, objects, arrays, and child `@Input()` / `input()` signals directly to DOM element properties, `[class.x]`, `[style.width.px]`, and `[attr.*]`. • ✓ String Interpolation `{{ }}` — Pure Template Expressions, Null-Safety (`?.`, `??`) & XSS Auto-Escaping — Rendering dynamic text expressions, pipes (`| currency:'INR'`), and optional chaining (`user?.profile?.name ?? 'Guest'`) with automatic HTML entity encoding against XSS. • ✓ Build, verify, and ship Module Capstone: Accessible Exam Question Card with ARIA Attribute Binding & XSS Sanitizer Guard.

7.1 Property Binding `[ ]` — DOM Properties vs HTML Attributes, Boolean `[disabled]` & `[attr.*]` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: DOM property vs initial HTML attribute & [disabled]="false" vs disabled="false" trap • Core Syntax, CLI & Production APIs: <button [disabled]="!isValid()" [class.primary]="isReady" [attr.aria-label]="label"> • Deep-Dive Mechanics: [attr.colspan] and [attr.aria-*] prefix & [class.active] single-class toggle • Advanced Patterns: [style.width.%] unit-suffixed style binding & Child component @Input() / input() passing • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Boolean Property Binding vs String Attribute Truthiness Demonstration • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Property Binding `[ ]`

Purpose: To understand and apply Property Binding `[ ]` — DOM Properties vs HTML Attributes, Boolean `[disabled]` & `[attr.*]` in real Angular systems — specifically: Binding live...

What Topic Is Next → Next Topic 7.2: String Interpolation `{{ }}` — Pure Template Expressions, Null-Safety (`?.`, `??`) & XSS Auto-Escaping

7.2 String Interpolation `{{ }}` — Pure Template Expressions, Null-Safety (`?.`, `??`) & XSS Auto-Escaping [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: TextNode safe string interpolation & Null and undefined render as empty string • Core Syntax, CLI & Production APIs: {{ student()?.name ?? 'Guest' }} • {{ price | number:'1.0-0' }} • Deep-Dive Mechanics: Optional chaining ?. and nullish ?? safety & Template expression purity (no =, +=, ++, new) • Advanced Patterns: Automatic XSS HTML escaping & Pipe operator | chaining inside {{ }} • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Angular Safe Interpolation & Nullish Coalescing Renderer • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for String Interpolation `{{ }}`

Purpose: To understand and apply String Interpolation `{{ }}` — Pure Template Expressions, Null-Safety (`?.`, `??`) & XSS Auto-Escaping in real Angular systems — specifically: Ren...

What Topic Is Next → Module 7 Capstone (Accessible Exam Question Card with ARIA Attribute Binding & XSS Sanitizer Guard) → Next Topic 8.1: Event Binding `( )` — Typed `$event`, Key Pseudo-Events `(keyup.enter)` & Child `output()`

Module 7 Hands-On Capstone Project (100 Marks): Accessible Exam Question Card with ARIA Attribute Binding & XSS Sanitizer Guard — Implement an accessible question card binding boolean `[disabled]`, `[attr.aria-expanded]`, `[class.selected]`, and safe text interpolation while blocking `<script>`/`onerror` XSS payloads.

MODULE 8 OF 16 • STAGE 2 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: DOM Event Binding (`( )`), Key Filters & Two-Way Binding (`[(ngModel)]` / `model()`)

Capture DOM & custom child events with typed `$event`, key pseudo-events (`(keyup.enter)`), `output()`, and build custom two-way bindings using `[( )]` / Signal `model()`.

Objectives: ✓ Event Binding `( )` — Typed `$event`, Key Pseudo-Events `(keyup.enter)` & Child `output()` — Listening to native DOM events and custom child component `@Output()` / `output<T>()` events cleanly without colliding with native DOM event names. • ✓ Two-Way Data Binding `[()]` — Banana-in-a-Box Desugaring, `FormsModule` `ngModel` & Signal `model()` — Understanding how `[(ngModel)]="email"` and custom `[(size)]="fontPx"` desugar into paired property + `*Change` event bindings, and simplifying custom two-way components with `model()`. • ✓ Build, verify, and ship Module Capstone: Keyboard-Accessible Rating & Search Bar with `output()` and `model()` Two-Way Sync.

8.1 Event Binding `( )` — Typed `$event`, Key Pseudo-Events `(keyup.enter)` & Child `output()` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: (event)="handler($event)" DOM -> TS flow & Typed KeyboardEvent / HTMLInputElement target • Core Syntax, CLI & Production APIs: (click)="save()" + (keyup.enter)="submitSearch()" + readonly saved = output<number>() • Deep-Dive Mechanics: Key pseudo-events (keyup.enter, keydown.esc) & @Output() EventEmitter<T> & modern output<T>() • Advanced Patterns: Avoiding native DOM event name collisions & Event bubbling and $event.stopPropagation() • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Typed Child `output<T>()` Emitter & Enter-Key Filter Simulation • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Event Binding `( )`

Purpose: To understand and apply Event Binding `( )` — Typed `$event`, Key Pseudo-Events `(keyup.enter)` & Child `output()` in real Angular systems — specifically: Listening to na...

What Topic Is Next → Next Topic 8.2: Two-Way Data Binding `[()]` — Banana-in-a-Box Desugaring, `FormsModule` `ngModel` & Signal `model()`

8.2 Two-Way Data Binding `[()]` — Banana-in-a-Box Desugaring, `FormsModule` `ngModel` & Signal `model()` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Banana-in-a-box [()] mnemonic & [prop] + (propChange) compile-time desugaring • Core Syntax, CLI & Production APIs: [(ngModel)]="coupon" (FormsModule) | readonly fontPx = model.required<number>() • Deep-Dive Mechanics: FormsModule requirement for [(ngModel)] & Mandatory name="" inside <form> for ngModel • Advanced Patterns: Modern Signal model() two-way primitive & When to prefer Reactive Forms over ngModel • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Custom `prop` + `propChange` Two-Way Desugaring Verifier • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Two-Way Data Binding `[()]`

Purpose: To understand and apply Two-Way Data Binding `[()]` — Banana-in-a-Box Desugaring, `FormsModule` `ngModel` & Signal `model()` in real Angular systems — specifically: Under...

What Topic Is Next → Module 8 Capstone (Keyboard-Accessible Rating & Search Bar with `output()` and `model()` Two-Way Sync) → Next Topic 9.1: Angular Forms — Template-Driven vs Strictly Typed Reactive Forms (`FormGroup`, `FormArray`, `Validators`)

Module 8 Hands-On Capstone Project (100 Marks): Keyboard-Accessible Rating & Search Bar with `output()` and `model()` Two-Way Sync — Create a reusable `<app-score-slider [(score)]="currentScore">` component using Angular's `x` / `xChange` (`model()`) contract plus `(keydown.arrowUp)` keyboard accessibility.

MODULE 9 OF 16 • STAGE 3 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Strictly Typed Reactive Forms & Enterprise Architecture Trade-Offs (Angular vs React)

Build immutable, unit-testable `NonNullableFormBuilder` Reactive Forms with cross-field/async validators and evaluate Angular vs React architectural trade-offs objectively.

Objectives: ✓ Angular Forms — Template-Driven vs Strictly Typed Reactive Forms (`FormGroup`, `FormArray`, `Validators`) — Controlling complex validation, dynamic `FormArray` rows, `valueChanges` RxJS streams, and headless unit testing with `ReactiveFormsModule` and `NonNullableFormBuilder`. • ✓ Angular vs React — Opinionated Platform vs UI Library, Signals/Ivy vs Virtual DOM & DI — Architectural comparison for senior technical interviews: Angular's AOT Ivy instructions, built-in DI/Router/Forms/HttpClient, and Signals vs React's JSX Virtual DOM reconciliation and unopinionated library ecosystem. • ✓ Build, verify, and ship Module Capstone: Strictly Typed Multi-Skill Enrollment `FormGroup` & Cross-Field Validator.

9.1 Angular Forms — Template-Driven vs Strictly Typed Reactive Forms (`FormGroup`, `FormArray`, `Validators`) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: ReactiveFormsModule vs Template-Driven FormsModule & Strictly Typed FormGroup & NonNullableFormBuilder • Core Syntax, CLI & Production APIs: inject(NonNullableFormBuilder).group({ email: ['', [Validators.required, Validators.email]]... • Deep-Dive Mechanics: Dynamic FormArray control collections & ValidatorFn (null = valid) & AsyncValidatorFn • Advanced Patterns: SetValue (all keys) vs patchValue (partial) & { emitEvent: false } infinite-loop guard • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Pure TypeScript Cross-Field Reactive Validator (`null` = Valid) • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular Forms

Purpose: To understand and apply Angular Forms — Template-Driven vs Strictly Typed Reactive Forms (`FormGroup`, `FormArray`, `Validators`) in real Angular systems — specifically:...

What Topic Is Next → Next Topic 9.2: Angular vs React — Opinionated Platform vs UI Library, Signals/Ivy vs Virtual DOM & DI

9.2 Angular vs React — Opinionated Platform vs UI Library, Signals/Ivy vs Virtual DOM & DI [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Full opinionated framework vs UI library & Ivy AOT direct DOM instructions vs Virtual DOM diff • Core Syntax, CLI & Production APIs: Angular (AOT Ivy + Signals + DI + RxJS) vs React (JSX + Fiber VDOM + Hooks) • Deep-Dive Mechanics: Angular Signals vs React useState/useMemo hooks & Built-in Hierarchical DI vs React Context • Advanced Patterns: HTML templates (@if/@for) vs JSX JavaScript expressions & Synchronized ng update migrations across large teams • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Architecture Decision Matrix Scorer (Enterprise Governance vs Micro-Widget) • Idiomatic ANGL

Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular vs React

Purpose: To understand and apply Angular vs React — Opinionated Platform vs UI Library, Signals/Ivy vs Virtual DOM & DI in real Angular systems — specifically: Architectural compa...

What Topic Is Next → Module 9 Capstone (Strictly Typed Multi-Skill Enrollment `FormGroup` & Cross-Field Validator) → Next Topic 10.1: Angular + Spring Boot CRUD Integration — Typed `HttpClient`, Cold Observables, CORS & `catchError`

Module 9 Hands-On Capstone Project (100 Marks): Strictly Typed Multi-Skill Enrollment `FormGroup` & Cross-Field Validator — Build a `NonNullableFormBuilder` checkout form model with email/phone validators, a dynamic `FormArray` of skill tags, and a cross-field discount-cap validator testable without DOM rendering...

MODULE 10 OF 16 • STAGE 3 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Full-Stack Angular + Spring Boot REST CRUD, JWT Interceptors & Route Guards

Connect Angular `HttpClient` services to Spring Boot REST APIs with RxJS `catchError`, cold Observable lifecycle discipline, functional `HttpInterceptorFn` JWT headers, and `CanActivateFn` guards.

Objectives: ✓ Angular + Spring Boot CRUD Integration — Typed `HttpClient`, Cold Observables, CORS & `catchError` — Wiring Angular `provideHttpClient()` services to Spring Boot `@RestController` endpoints (`GET /api/v1/employees`, `POST`, `PUT`, `DELETE`) with typed DTOs and RxJS error recovery. • ✓ Angular + Spring Security Login & Logout — Functional `HttpInterceptorFn`, `req.clone()` & `CanActivateFn` — End-to-end SPA authentication with Spring Security 6: exchanging credentials, attaching `Bearer` tokens via immutable `HttpRequest.clone()`, guarding routes with `CanActivateFn`, and handling `401` logout. • ✓ Build, verify, and ship Module Capstone: Full-Stack Angular + Spring Boot Student Portal Client with JWT `HttpInterceptorFn` & `CanActivateFn`.

10.1 Angular + Spring Boot CRUD Integration — Typed `HttpClient`, Cold Observables, CORS & `catchError`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: ProvideHttpClient(withInterceptors([...])) & Cold Observable execution on subscribe • Core Syntax, CLI & Production APIs: this.http.get<Employee[]>('/api/v1/employees').pipe(catchError(this.handleError)) • Deep-Dive Mechanics: Typed http.get<T> / post<T> / put<T> / delete<void> & RxJS pipe(retry, catchError) resilience • Advanced Patterns: Spring Boot @CrossOrigin / proxy.conf.json dev proxy & Immutable UI signal list refresh after CRUD • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Cold `HttpClient` Observable Execution & Immutable CRUD State Update • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular + Spring Boot CRUD Integration

Purpose: To understand and apply Angular + Spring Boot CRUD Integration — Typed `HttpClient`, Cold Observables, CORS & `catchError` in real Angular systems — specifically: Wiring...

What Topic Is Next → Next Topic 10.2: Angular + Spring Security Login & Logout — Functional `HttpInterceptorFn`, `req.clone()` & `CanActivateFn`

10.2 Angular + Spring Security Login & Logout — Functional `HttpInterceptorFn`, `req.clone()` & `CanActivateFn`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Functional HttpInterceptorFn + withInterceptors & HttpRequest immutability (req.clone) • Core Syntax, CLI & Production APIs: export const jwtInterceptor: HttpInterceptorFn = (req, next) => next(req.clone({ setHeaders:... • Deep-Dive Mechanics: Origin-scoped Bearer header attachment & CanActivateFn functional guard + UrlTree redirect • Advanced Patterns: HttpOnly cookie vs short-lived memory JWT storage & 401 Unauthorized auto-logout & state reset • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Immutable `HttpInterceptorFn` Clone & `CanActivateFn` Guard Verifier • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular + Spring Security Login

Purpose: To understand and apply Angular + Spring Security Login & Logout — Functional `HttpInterceptorFn`, `req.clone()` & `CanActivateFn` in real Angular systems — specifically:...

What Topic Is Next → Module 10 Capstone (Full-Stack Angular + Spring Boot Student Portal Client with JWT `HttpInterceptorFn` & `CanActivateFn`) → Next Topic 11.1: signal(), computed(), linkedSignal() & Explicit effect() Boundaries

Module 10 Hands-On Capstone Project (100 Marks): Full-Stack Angular + Spring Boot Student Portal Client with JWT `HttpInterceptorFn` & `CanActivateFn` — Implement a typed Angular `StudentApiService` CRUD layer, a functional JWT `authInterceptor` that clones requests immutably for `/api/**` calls, and an `authGuard` returning a `/login` `UrlT...

MODULE 11 OF 16 • STAGE 3 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Modern Angular Signals, Zoneless Change Detection & Reactive Stores

Production-grade Angular Training engineering module covering Modern Angular Signals, Zoneless Change Detection & Reactive Stores with hands-on labs and capstone build.

Objectives: ✓ signal(), computed(), linkedSignal() & Explicit effect() Boundaries — Purpose: Update exact DOM nodes surgically without scanning the entire component tree. • ✓ OnPush & Zoneless Change Detection Architecture — Purpose: Eliminate Zone.js overhead and achieve ultra-snappy Core Web Vitals INP scores. • ✓ NgRx SignalStore / Entity Management for Complex Domain Workflows — Purpose: Coordinate loading, optimistic updates, and normalized collections cleanly. • ✓ Bridging RxJS Streams & Signals (toSignal / toObservable) — Purpose: Combine powerful RxJS debounce/switchMap HTTP operators with ergonomic Signal templates.

11.1 signal(), computed(), linkedSignal() & Explicit effect() Boundaries

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of computed() • Integration with OnPush & Zoneless Change Detection Architecture • Live Debugging, Telemetry & Technical Interview Trade • What Is signal(), computed(), linkedSignal() & Explicit effect() Boundaries & Why Industry Teams Use It • In Angular Training, signal(), computed(), linkedSignal() & Explicit effect() Boundaries exists for one clear real • Even if you come from a non • Runnable Code Implementation

(TYPESCRIPT)

Purpose: To understand and apply signal(), computed(), linkedSignal() & Explicit effect() Boundaries in real Angular systems — specifically: Update exact DOM nodes surgically with...

What Topic Is Next → Next Topic 11.2: OnPush & Zoneless Change Detection Architecture

11.2 OnPush & Zoneless Change Detection Architecture [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Zoneless Change Detection Architecture • Integration with NgRx SignalStore / Entity Management for Complex Domain Workflows • Live Debugging, Telemetry & Technical Interview Trade • What Is OnPush & Zoneless Change Detection Architecture & Why Industry Teams Use It • In Angular Training, OnPush & Zoneless Change Detection Architecture exists for one clear real • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply OnPush & Zoneless Change Detection Architecture in real Angular systems — specifically: Eliminate Zone.js overhead and achieve ultra-snappy Core W...

What Topic Is Next → Next Topic 11.3: NgRx SignalStore / Entity Management for Complex Domain Workflows

11.3 NgRx SignalStore / Entity Management for Complex Domain Workflows [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Bridging RxJS Streams & Signals (toSignal / toObservable) • Live Debugging, Telemetry & Technical Interview Trade • What Is NgRx SignalStore / Entity Management for Complex Domain Workflows & Why Industry Teams Use It • In Angular Training, NgRx SignalStore / Entity Management for Complex Domain Workflows exists for one clear real • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply NgRx SignalStore / Entity Management for Complex Domain Workflows in real Angular systems — specifically: Coordinate loading, optimistic updates,...

What Topic Is Next → Next Topic 11.4: Bridging RxJS Streams & Signals (toSignal / toObservable)

11.4 Bridging RxJS Streams & Signals (toSignal / toObservable) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Signals (toSignal / toObservable) • Integration with signal(), computed(), linkedSignal() & Explicit effect() Boundaries • Live Debugging, Telemetry & Technical Interview Trade • What Is Bridging RxJS Streams & Signals (toSignal / toObservable) & Why Industry Teams Use It • In Angular Training, Bridging RxJS Streams & Signals (toSignal / toObservable) exists for one clear real • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply Bridging RxJS Streams & Signals (toSignal / toObservable) in real Angular systems — specifically: Combine powerful RxJS debounce/switchMap HTTP op...

What Topic Is Next → Module 11 Capstone (Fine-Grained Signal-Driven Enterprise Operations Console) → Next Topic 12.1: Strictly Typed FormGroup/FormArray & Reusable ControlValueAccessor

Module 11 Hands-On Capstone Project (100 Marks): Fine-Grained Signal-Driven Enterprise Operations Console — Design, implement, test, and document 'Fine-Grained Signal-Driven Enterprise Operations Console' applying every production concept from Module 11 (Modern Angular Signals, Zoneless Change Det...

MODULE 12 OF 16 • STAGE 3 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Enterprise Typed Reactive Forms, Functional Guards & HTTP Interceptors

Production-grade Angular Training engineering module covering Enterprise Typed Reactive Forms, Functional Guards & HTTP Interceptors with hands-on labs and capstone build.

Objectives: ✓ Strictly Typed FormGroup/FormArray & Reusable ControlValueAccessor — Purpose: Build custom date-pickers, currency inputs, and dynamic rule forms with full type safety. • ✓ Async Cross-Field Validators & Accessible Validation Summaries — Purpose: Validate PAN/GST/email uniqueness live without spamming backend APIs. • ✓ Functional CanActivateFn / CanMatchFn Route Guards & Preloading — Purpose: Enforce RBAC permissions before lazy route bundles are even downloaded. • ✓ Functional HttpInterceptorFn for JWT Refresh Queuing & Retry Backoff — Purpose: Handle 401 token renewal transparently across concurrent inflight HTTP calls.

12.1 Strictly Typed FormGroup/FormArray & Reusable ControlValueAccessor [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Reusable ControlValueAccessor • Integration with Async Cross • Live Debugging, Telemetry & Technical Interview Trade • What Is Strictly Typed FormGroup/FormArray & Reusable ControlValueAccessor & Why Industry Teams Use It • In Angular Training, Strictly Typed FormGroup/FormArray & Reusable ControlValueAccessor exists for one clear real • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply Strictly Typed FormGroup/FormArray & Reusable ControlValueAccessor in real Angular systems — specifically: Build custom date-pickers, currency inp...

What Topic Is Next → Next Topic 12.2: Async Cross-Field Validators & Accessible Validation Summaries

12.2 Async Cross-Field Validators & Accessible Validation Summaries [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Accessible Validation Summaries • Integration with Functional CanActivateFn / CanMatchFn Route Guards & Preloading • Live Debugging, Telemetry & Technical Interview Trade • What Is Async Cross • In Angular Training, Async Cross • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply Async Cross-Field Validators & Accessible Validation Summaries in real Angular systems — specifically: Validate PAN/GST/email uniqueness live with...

What Topic Is Next → Next Topic 12.3: Functional CanActivateFn / CanMatchFn Route Guards & Preloading

12.3 Functional CanActivateFn / CanMatchFn Route Guards & Preloading [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Preloading • Integration with Functional HttpInterceptorFn for JWT Refresh Queuing & Retry Backoff • Live Debugging, Telemetry & Technical Interview Trade • What Is Functional CanActivateFn / CanMatchFn Route Guards & Preloading & Why Industry Teams Use It • In Angular Training, Functional CanActivateFn / CanMatchFn Route Guards & Preloading exists for one clear real • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply Functional CanActivateFn / CanMatchFn Route Guards & Preloading in real Angular systems — specifically: Enforce RBAC permissions before lazy route...

What Topic Is Next → Next Topic 12.4: Functional HttpInterceptorFn for JWT Refresh Queuing & Retry Backoff

12.4 Functional HttpInterceptorFn for JWT Refresh Queuing & Retry Backoff [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Retry Backoff • Integration with Strictly Typed FormGroup/FormArray & Reusable ControlValueAccessor • Live Debugging, Telemetry & Technical Interview Trade • What Is Functional HttpInterceptorFn for JWT Refresh Queuing & Retry Backoff & Why Industry Teams Use It • In Angular Training, Functional HttpInterceptorFn for JWT Refresh Queuing & Retry Backoff exists for one clear real • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply Functional HttpInterceptorFn for JWT Refresh Queuing & Retry Backoff in real Angular systems — specifically: Handle 401 token renewal transparentl...

What Topic Is Next → Module 12 Capstone (Dynamic Multi-Step Banking KYC Portal in Angular) → Next Topic 13.1: Template @defer Blocks (on viewport, on interaction, on idle)

Module 12 Hands-On Capstone Project (100 Marks): Dynamic Multi-Step Banking KYC Portal in Angular — Design, implement, test, and document 'Dynamic Multi-Step Banking KYC Portal in Angular' applying every production concept from Module 12 (Enterprise Typed Reactive Forms, Functional Guards...

MODULE 13 OF 16 • STAGE 4 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Deferrable Views (@defer), SSR Hydration, Virtual Scroll & Accessibility

Production-grade Angular Training engineering module covering Deferrable Views (@defer), SSR Hydration, Virtual Scroll & Accessibility with hands-on labs and capstone build.

Objectives: ✓ Template @defer Blocks (on viewport, on interaction, on idle) — Purpose: Lazy-load heavy charts and below-the-fold widgets declaratively with zero boilerplate. • ✓ Angular CDK Virtual Scroll, Overlays, FocusTrap & Drag-and-Drop — Purpose: Render 25,000-row enterprise tables and accessible Kanban/modals effortlessly. • ✓ Server-Side Rendering (Angular SSR), TransferState & Non-Destructive Hydration — Purpose: Deliver instant SEO-ready first paint without double-fetching initial API data. • ✓ NgOptimizedImage, Bundle Budget Enforcement & Source-Map Audits — Purpose: Fail CI automatically if an accidental library import bloats the main JS chunk.

13.1 Template @defer Blocks (on viewport, on interaction, on idle) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of on interaction • Integration with Angular CDK Virtual Scroll, Overlays, FocusTrap & Drag • Live Debugging, Telemetry & Technical Interview Trade • What Is Template @defer Blocks (on viewport, on interaction, on idle) & Why Industry Teams Use It • In Angular Training, Template @defer Blocks (on viewport, on interaction, on idle) exists for one clear real • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply Template @defer Blocks (on viewport, on interaction, on idle) in real Angular systems — specifically: Lazy-load heavy charts and below-the-fold wi...

What Topic Is Next → Next Topic 13.2: Angular CDK Virtual Scroll, Overlays, FocusTrap & Drag-and-Drop

13.2 Angular CDK Virtual Scroll, Overlays, FocusTrap & Drag-and-Drop [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Overlays • Integration with Server • Live Debugging, Telemetry & Technical Interview Trade • What Is Angular CDK Virtual Scroll, Overlays, FocusTrap & Drag • In Angular Training, Angular CDK Virtual Scroll, Overlays, FocusTrap & Drag • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply Angular CDK Virtual Scroll, Overlays, FocusTrap & Drag-and-Drop in real Angular systems — specifically: Render 25,000-row enterprise tables and ac...

What Topic Is Next → Next Topic 13.3: Server-Side Rendering (Angular SSR), TransferState & Non-Destructive Hydration

13.3 Server-Side Rendering (Angular SSR), TransferState & Non-Destructive Hydration [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of TransferState • Integration with NgOptimizedImage, Bundle Budget Enforcement & Source • Live Debugging, Telemetry & Technical Interview Trade • What Is Server • In Angular Training, Server • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply Server-Side Rendering (Angular SSR), TransferState & Non-Destructive Hydration in real Angular systems — specifically: Deliver instant SEO-ready f...

What Topic Is Next → Next Topic 13.4: NgOptimizedImage, Bundle Budget Enforcement & Source-Map Audits

13.4 NgOptimizedImage, Bundle Budget Enforcement & Source-Map Audits [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Bundle Budget Enforcement • Integration with Template @defer Blocks (on viewport, on interaction, on idle) • Live Debugging, Telemetry & Technical Interview Trade • What Is NgOptimizedImage, Bundle Budget Enforcement & Source • In Angular Training, NgOptimizedImage, Bundle Budget Enforcement & Source • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply NgOptimizedImage, Bundle Budget Enforcement & Source-Map Audits in real Angular systems — specifically: Fail CI automatically if an accidental lib...

What Topic Is Next → Module 13 Capstone (High-Speed Accessible Angular Storefront & Data Grid) → Next Topic 14.1: Feature-Sliced Standalone Folder Structure & Library Boundaries

Module 13 Hands-On Capstone Project (100 Marks): High-Speed Accessible Angular Storefront & Data Grid — Design, implement, test, and document 'High-Speed Accessible Angular Storefront & Data Grid' applying every production concept from Module 13 (Deferrable Views (@defer), SSR Hydration, Virtu...

MODULE 14 OF 16 • STAGE 4 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Nx Workspace Architecture, Component Harnesses, Playwright & CI/CD

Production-grade Angular Training engineering module covering Nx Workspace Architecture, Component Harnesses, Playwright & CI/CD with hands-on labs and capstone build.

Objectives: ✓ Feature-Sliced Standalone Folder Structure & Library Boundaries — Purpose: Keep large 40-screen Angular apps maintainable across multiple agile squads. • ✓ Unit Testing Signals, Services & HttpTestingController Specs — Purpose: Verify interceptors, guards, and stores fast with deterministic HTTP mocks. • ✓ CDK Test Harnesses & Playwright End-to-End User Journey Suites — Purpose: Write resilient UI tests decoupled from internal CSS class names. • ✓ Multi-Environment Docker Nginx Builds, CSP Headers & Live Deployment — Purpose: Ship cache-optimized Angular builds with strict Content-Security-Policy protection.

14.1 Feature-Sliced Standalone Folder Structure & Library Boundaries [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Library Boundaries • Integration with Unit Testing Signals, Services & HttpTestingController Specs • Live Debugging, Telemetry & Technical Interview Trade • What Is Feature • In Angular Training, Feature • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply Feature-Sliced Standalone Folder Structure & Library Boundaries in real Angular systems — specifically: Keep large 40-screen Angular apps maintain...

What Topic Is Next → Next Topic 14.2: Unit Testing Signals, Services & HttpTestingController Specs

14.2 Unit Testing Signals, Services & HttpTestingController Specs [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Services • Integration with CDK Test Harnesses & Playwright End • Live Debugging, Telemetry & Technical Interview Trade • What Is Unit Testing Signals, Services & HttpTestingController Specs & Why Industry Teams Use It • In Angular Training, Unit Testing Signals, Services & HttpTestingController Specs exists for one clear real • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply Unit Testing Signals, Services & HttpTestingController Specs in real Angular systems — specifically: Verify interceptors, guards, and stores fast...

What Topic Is Next → Next Topic 14.3: CDK Test Harnesses & Playwright End-to-End User Journey Suites

14.3 CDK Test Harnesses & Playwright End-to-End User Journey Suites [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Playwright End • Integration with Multi • Live Debugging, Telemetry & Technical Interview Trade • What Is CDK Test Harnesses & Playwright End • In Angular Training, CDK Test Harnesses & Playwright End • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply CDK Test Harnesses & Playwright End-to-End User Journey Suites in real Angular systems — specifically: Write resilient UI tests decoupled from int...

What Topic Is Next → Next Topic 14.4: Multi-Environment Docker Nginx Builds, CSP Headers & Live Deployment

14.4 Multi-Environment Docker Nginx Builds, CSP Headers & Live Deployment [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of CSP Headers • Integration with Feature • Live Debugging, Telemetry & Technical Interview Trade • What Is Multi • In Angular Training, Multi • Even if you come from a non • Runnable Code Implementation (TYPESCRIPT)

Purpose: To understand and apply Multi-Environment Docker Nginx Builds, CSP Headers & Live Deployment in real Angular systems — specifically: Ship cache-optimized Angular builds w...

What Topic Is Next → Module 14 Capstone (Modular Angular Monorepo Pipeline with Automated E2E Gate) → Next Topic 15.1: Angular Signals (signal, computed, effect), input()/output() & Zoneless Readiness

Module 14 Hands-On Capstone Project (100 Marks): Modular Angular Monorepo Pipeline with Automated E2E Gate — Design, implement, test, and document 'Modular Angular Monorepo Pipeline with Automated E2E Gate' applying every production concept from Module 14 (Nx Workspace Architecture, Component Harne...

MODULE 15 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Angular Signals, Fine-Grained Reactivity & Enterprise State Architecture

Self-explanatory list-and-bullet production mastery module covering Angular Signals, Fine-Grained Reactivity & Enterprise State Architecture in Angular Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Angular Signals (signal, computed, effect), input()/output() & Zoneless Readiness — Purpose: Build predictable, ultra-fast reactive Angular components without manual ChangeDetectorRef juggling. • ✓ RxJS Interop (toSignal, toObservable), Cancellation & Functional HTTP Interceptors — Purpose: Combine debounced search streams, JWT refresh queues, and centralized retry/error handling. • ✓ Typed Reactive Forms, Cross-Field Validators & Reusable ControlValueAccessor Inputs — Purpose: Handle complex enterprise multi-step workflows with strict compile-time form type safety. • ✓ Build, verify, and

document 'Signals + RxJS Interop Operations Console with Optimistic HTTP Interceptors' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

15.1 Angular Signals (signal, computed, effect), input()/output() & Zoneless Readiness [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Angular Signals (signal, computed, effect), input()/output() & Zoneless Re... • Performance Optimization, Complexity & Latency Impact of Angular Signals (signal, computed, effect), input()/output... • Live Debugging, Stack • Runnable Code Implementation (TYPESCRIPT) • Hands • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular Signals (signal, computed, effect) • Production Hardening & Security: DomSanitizer XSS defense, CanActivateFn guards & JWT refresh queues

Purpose: To understand and apply Angular Signals (signal, computed, effect), input()/output() & Zoneless Readiness in real Angular systems — specifically: Build predictable, ultra...

What Topic Is Next → Next Topic 15.2: RxJS Interop (toSignal, toObservable), Cancellation & Functional HTTP Interceptors

15.2 RxJS Interop (toSignal, toObservable), Cancellation & Functional HTTP Interceptors [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for RxJS Interop (toSignal, toObservable), Cancellation & Functional HTTP Inte... • Performance Optimization, Complexity & Latency Impact of RxJS Interop (toSignal, toObservable), Cancellation & Func... • Live Debugging, Stack • Runnable Code Implementation (TYPESCRIPT) • Hands • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for RxJS Interop (toSignal, toObservable) • Production Hardening & Security: DomSanitizer XSS defense, CanActivateFn guards & JWT refresh queues

Purpose: To understand and apply RxJS Interop (toSignal, toObservable), Cancellation & Functional HTTP Interceptors in real Angular systems — specifically: Combine debounced searc...

What Topic Is Next → Next Topic 15.3: Typed Reactive Forms, Cross-Field Validators & Reusable ControlValueAccessor Inputs

15.3 Typed Reactive Forms, Cross-Field Validators & Reusable ControlValueAccessor Inputs [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Typed Reactive Forms, Cross • Performance Optimization, Complexity & Latency Impact of Typed Reactive Forms, Cross • Live Debugging, Stack • Runnable Code Implementation (TYPESCRIPT) • Hands • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Typed Reactive Forms • Production Hardening & Security: DomSanitizer XSS defense, CanActivateFn guards & JWT refresh queues

Purpose: To understand and apply Typed Reactive Forms, Cross-Field Validators & Reusable ControlValueAccessor Inputs in real Angular systems — specifically: Handle complex enterpr...

What Topic Is Next → Module 15 Capstone (Signals + RxJS Interop Operations Console with Optimistic HTTP Interceptors) → Next Topic 16.1: Template @defer Blocks, Route Lazy Loading, NgOptimizedImage & Bundle Budgets

Module 15 Hands-On Capstone Project (100 Marks): Signals + RxJS Interop Operations Console with Optimistic HTTP Interceptors — Architect, implement, test, and document 'Signals + RxJS Interop Operations Console with Optimistic HTTP Interceptors' applying every bullet-wise engineering concept from Module 15 (Angular...

MODULE 16 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Deferrable Views (@defer), SSR Hydration, Testing & Angular Interviews

Self-explanatory list-and-bullet production mastery module covering Deferrable Views (@defer), SSR Hydration, Testing & Angular Interviews in Angular Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Template @defer Blocks, Route Lazy Loading, NgOptimizedImage & Bundle Budgets — Purpose: Slash initial bundle size and achieve top Core Web Vitals on large enterprise Angular portals. • ✓ Testing Standalone Components, Signals, Guards & HTTP Flows with TestBed / Harness — Purpose: Automate regression tests for user interactions, route guards, and API error states. • ✓ Angular DI Hierarchy, Change Detection Internals & Senior Frontend Interviews — Purpose: Explain OnPush vs Signals, memory-leak-free takeUntilDestroyed(), and monorepo architecture. • ✓ Build, verify, and document 'Bundle-Optimized Standalone Angular App with Deferrable Charts & Component Harness Tests' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

16.1 Template @defer Blocks, Route Lazy Loading, NgOptimizedImage & Bundle Budgets [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Template @defer Blocks, Route Lazy Loading, NgOptimizedImage & Bundle Budg... • Performance Optimization, Complexity & Latency Impact of Template @defer Blocks, Route Lazy Loading, NgOptimizedIma... • Live Debugging, Stack • Runnable Code Implementation (TYPESCRIPT) • Hands • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Template @defer Blocks • Production Hardening & Security: DomSanitizer XSS defense, CanActivateFn guards & JWT refresh queues

Purpose: To understand and apply Template @defer Blocks, Route Lazy Loading, NgOptimizedImage & Bundle Budgets in real Angular systems — specifically: Slash initial bundle size an...

What Topic Is Next → Next Topic 16.2: Testing Standalone Components, Signals, Guards & HTTP Flows with TestBed / Harness

16.2 Testing Standalone Components, Signals, Guards & HTTP Flows with TestBed / Harness [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Testing Standalone Components, Signals, Guards & HTTP Flows with TestBed /... • Performance Optimization, Complexity & Latency Impact of Testing Standalone Components, Signals, Guards & HTTP Flow... • Live Debugging, Stack • Runnable Code Implementation (TYPESCRIPT) • Hands • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... •

Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Testing Standalone Components • Production Hardening & Security: DomSanitizer XSS defense, CanActivateFn guards & JWT refresh queues

Purpose: To understand and apply Testing Standalone Components, Signals, Guards & HTTP Flows with TestBed / Harness in real Angular systems — specifically: Automate regression tes...

What Topic Is Next → Next Topic 16.3: Angular DI Hierarchy, Change Detection Internals & Senior Frontend Interviews

16.3 Angular DI Hierarchy, Change Detection Internals & Senior Frontend Interviews [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Angular DI Hierarchy, Change Detection Internals & Senior Frontend Intervi... • Performance Optimization, Complexity & Latency Impact of Angular DI Hierarchy, Change Detection Internals & Senior.... • Live Debugging, Stack • Runnable Code Implementation (TYPESCRIPT) • Hands • Idiomatic ANGL Implementation: Standalone Components, OnPush strategy, typed Reactive Forms & HttpInterceptor... • Angular 18 Signals Graph, Zone/Zoneless Change Detection & RxJS Execution & Memory Lifecycle for Angular DI Hierarchy • Production Hardening & Security: DomSanitizer XSS defense, CanActivateFn guards & JWT refresh queues

Purpose: To understand and apply Angular DI Hierarchy, Change Detection Internals & Senior Frontend Interviews in real Angular systems — specifically: Explain OnPush vs Signals, m...

What Topic Is Next → Module 16 Capstone → End-of-Course Live Client Project: FinDesk Enterprise Wealth Management & Role-Gated Operations SPA

Module 16 Hands-On Capstone Project (100 Marks): Bundle-Optimized Standalone Angular App with Deferrable Charts & Component Harness Tests — Architect, implement, test, and document 'Bundle-Optimized Standalone Angular App with Deferrable Charts & Component Harness Tests' applying every bullet-wise engineering concept from Module...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

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

Project: FinDesk Enterprise Wealth Management & Role-Gated Operations SPA • **Confirmed Role:** Angular 18 & TypeScript Enterprise Frontend Intern

Domain & Production Infra: WealthTech & Corporate Treasury SaaS Client (Angular 18 Standalone Components + Signals + RxJS + Reactive Forms + Cloud CDN HTTPS)

- Built modular Angular 18 feature domains using Standalone Components, Signals, and OnPush change detection.
- Engineered complex multi-step Reactive Forms with async custom validators, HTTP interceptors, and route guards.
- Managed real-time data streams cleanly using RxJS operators (switchMap, combineLatest, shareReplay) without memory leaks.
- Shipped lazy-loaded, AOT-compiled production bundles passing strict TypeScript and unit test gates.

Target Job Roles: Angular Frontend Engineer, Enterprise UI Software Engineer, TypeScript & Angular SPA Developer, Frontend Application Specialist

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹5.0 LPA – ₹9.2 LPA **Mid-Level Angular Product Engineer (1–3 Yrs):** ₹9.5 LPA – ₹19.5 LPA **Senior Enterprise Frontend / Full-Stack Architect (3–6+ Yrs):** ₹20.0 LPA – ₹38.0+ LPA

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 98765 43210 • Web: https://mockattempt.com/courses/angular-training

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