Angular Platform Engineer Fieldbook

Angular Platform Engineer Fieldbook

> A current, plain-language guide to Angular 22, TypeScript 6, RxJS, Tailwind CSS 4.3, Flowbite 4, SSR, testing, security, and delivery.

\newpage

00. The modern Angular stack roadmap

Angular is a complete web application framework: component rendering, dependency injection, routing, forms, HTTP, build tooling, server rendering, testing, accessibility primitives, and an upgrade path.

Explain it like I am five

An Angular application is a tree of components. Each component owns a piece of the screen, receives inputs, emits events, reads state, and asks services for work. When tracked state changes, Angular updates the affected view.

Tailwind supplies small styling rules. Flowbite supplies styled interaction patterns. Neither replaces Angular's ownership of component state and lifecycle.

Current compatibility baseline, August 2026

Layer	Current baseline	Important constraint
Angular	22.1.x	Actively supported
Angular CLI	22.1.x	Keep framework and CLI majors aligned
Material / CDK	22.1.x	Match Angular major
TypeScript	6.0.x	Angular 22 requires `>=6.0 <6.1`
RxJS	7.8.x	Angular supports compatible 7.x
Node.js	22.22+, 24.15+, or supported 26.x	Use an Angular-supported line
Tailwind CSS	4.3.x	CSS-first configuration and separate integrations
Flowbite	4.0.x	Tailwind v4 themes and CSS variables
Unit tests	Vitest	Default for new CLI projects
Browser/E2E	Playwright or equivalent	Prove real browser behavior

“Latest” means the newest **compatible set**, not the highest version of every npm package. TypeScript 7 exists, but is outside Angular 22's supported range.

Architecture layers

```
browser → Angular router → feature shell → presentational components → application services/state → HTTP boundary → backend
```

Design tokens flow into Tailwind utilities and Flowbite themes. Accessibility, security, performance, testing, and observability cross every layer.

Learning loop

1. Model one user outcome.
2. Draw component and state ownership.

3. Implement the smallest route and vertical slice.
4. Test semantics and accessibility before visual polish.
5. Measure bundles, rendering, network, and Core Web Vitals.
6. Explain failure, loading, empty, and permission states.

Feynman check

Explain the difference between Angular, TypeScript, RxJS, Tailwind, Flowbite, Material/CDK, and the browser without calling any two of them “the frontend.”

\newpage

01. TypeScript 6 for Angular applications

TypeScript is Angular's compile-time language layer. It describes shapes and catches mismatches, then erases types to JavaScript for the browser.

Strict foundations

Enable strict TypeScript and Angular template checking. Avoid using `any` as a silence button. At untrusted boundaries, accept `unknown`, validate, and narrow.

```
type LoadState<T> =
  | { kind: 'idle' }
  | { kind: 'loading' }
  | { kind: 'ready'; value: T }
  | { kind: 'error'; message: string };
```

Discriminated unions make illegal UI states harder to represent and allow exhaustive switches.

Interfaces and types

Use interfaces for open object contracts and types for unions, mappings, and composition. Prefer domain names over transport names. An API response type is not automatically a valid domain model.

Generics

Generics relate input and output types. Keep constraints meaningful. A generic `ApiResponse<T>` may help at the transport boundary; a generic service with six type parameters may hide a poor abstraction.

Runtime validation

Types do not validate JSON. Use deliberate parsing, schema validation, or mapping before treating remote data as trusted. Dates arrive as strings unless you convert them. Numeric IDs can lose precision.

Immutability

Readonly types prevent accidental assignment at compile time; they do not deep- freeze runtime objects. Use immutable update patterns with signals and state so change is explicit.

Async types

Distinguish `Promise<T>`, `Observable<T>`, `Signal<T>`, and `Resource<T>`. They model different time and ownership semantics. Convert at clear boundaries, not repeatedly throughout templates.

Feynman check

TypeScript is a careful proofreader that leaves before the browser runs the story. Explain what security and validation work must remain at runtime.

\newpage

02. Standalone components and template control flow

Modern Angular applications are standalone by default. A component declares its own imports and can be lazy-loaded directly.

Component contract

```
@Component({
  selector: 'app-order-row',
  imports: [CurrencyPipe],
  templateUrl: './order-row.html',
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class OrderRow {
  order = input.required<Order>();
  selected = output<string>();
}
```

Inputs are data flowing in. Outputs are semantic events flowing out. Avoid outputs named `click`; emit business meaning such as `orderOpened`.

Template syntax

Use property binding for values, event binding for actions, and interpolation for text. Prefer Angular's built-in `@if`, `@for`, and `@switch` control flow. Give `@for` a stable track expression derived from identity.

```
@for (order of orders(); track order.id) {
  <app-order-row [order]="order" (selected)="open($event)" />
} @empty {
  <p>No matching orders.</p>
}
```

Host and content

Host bindings describe the component's own element. Content projection lets a parent provide controlled regions. View queries should not become a backdoor for mutating arbitrary child DOM.

DOM ownership

Let Angular own rendered DOM. Use template bindings, Renderer2, or post-render hooks for legitimate integration. Direct DOM mutation can break SSR hydration, tests, accessibility, and state consistency.

Smart and focused

Route/feature components coordinate data and user outcomes. Presentational components render explicit inputs and emit actions. Do not enforce a ceremonial split when one cohesive component is clearer.

Feynman check

A component is a small machine with labeled inputs, visible output, and named events. Explain who owns each piece of state on one real page.

\newpage

03. Dependency injection, providers, and application boundaries

Angular dependency injection supplies services according to a hierarchical provider tree.

inject and tokens

```
export const API_BASE_URL = new InjectionToken<string>('API_BASE_URL');
```

```
@Injectable({ providedIn: 'root' })
export class OrdersApi {
  private http = inject(HttpClient);
  private baseUrl = inject(API_BASE_URL);
}
```

Use class tokens for stable services and `InjectionToken` for configuration, interfaces, factories, and multi-provider extension points.

Provider scope

Root providers are shared for the application lifetime. Route providers create a feature-scoped instance. Component providers create a subtree instance. Scope is architecture: it decides state sharing, cleanup, and test boundaries.

Provider forms

`useClass`, `useValue`, `useFactory`, and `useExisting` express construction or aliasing. Multi providers collect extensions such as interceptors.

Injection context

`inject()` works in an injection context—constructors, field initializers, provider factories, and APIs that establish one. It is not a global service locator callable anywhere.

Environment configuration

Compile-time environment replacement is not enough for one immutable image across environments. Consider loading validated runtime configuration before bootstrap or serving a configuration endpoint.

Avoid god services

A service should own a coherent responsibility: HTTP adapter, application workflow, state store, analytics port, or browser capability. Do not put every feature's state into one root singleton.

Feynman check

The injector is a tree of cupboards. Angular looks in the nearest cupboard and then walks upward. Explain why the same service token can yield different instances on two routes.

\newpage

04. Signals, computed state, linkedSignal, and resources

Signals are Angular's granular synchronous reactive state model.

Writable and computed

```
items = signal<Order[]>([]);
query = signal('');
visible = computed(() =>
  this.items().filter(item => item.name.includes(this.query()))
);
```

Store source state in writable signals. Derive state with `computed`; do not copy derived values into another writable signal.

Effects

Effects synchronize state with non-reactive side effects such as logging, storage, or imperative APIs. They are not the default way to propagate state. Avoid writing signals inside effects when a computed relationship exists.

linkedSignal

`linkedSignal` models writable state that depends on another source—for example, a selected option that resets only when it becomes invalid after options change.

resource and httpResource

Resources combine reactive parameters with asynchronous loading, cancellation, status, value, and error signals. `resource` and `httpResource` are stable since Angular 22. `httpResource` uses HttpClient and interceptors. Guard `value()` with `hasValue()` because an error-state read can throw.

Use HttpClient methods directly for mutations. A resource is naturally suited to reads whose parameters are reactive.

Equality and mutation

Signals notify on updates according to equality. Mutating an array in place can hide change. Create the next value:

```
this.items.update(items => [...items, created]);
```

RxJS interop

`toSignal` subscribes to an Observable within lifecycle ownership. `toObservable` exposes signal changes as an Observable. Convert once at a boundary and keep the internal model consistent.

Feynman check

A signal is a labeled value with subscribers. A computed signal is a formula. An effect is a messenger leaving Angular's reactive graph. A resource is a managed async request with state.

\newpage

05. Zoneless change detection and render lifecycle

Zoneless is the default in Angular 21 and later. Angular refreshes views from known notifications instead of patching every browser async API through Zone.js.

Notifications

Angular learns a view may need work when a template-read signal changes, an input is set, a listener runs, `AsyncPipe` emits, a marked view is attached, or code calls `markForCheck`.

Code that mutates an untracked plain field after a timer may not update the view. Make state visible through signals, inputs, or supported notification.

OnPush

OnPush remains useful because it encourages explicit data flow and zoneless- compatible components. It is not a magic performance flag and does not repair mutated inputs.

Lifecycle

Use constructor/field initialization for dependency setup, `ngOnInit` for input- dependent initialization, and `ngOnDestroy` or `DestroyRef` for cleanup. Avoid work in frequently invoked checked hooks.

Render callbacks

`afterNextRender` and `afterEveryRender` coordinate DOM-dependent work after rendering. Keep browser-only APIs guarded for SSR. Use cleanup when the callback owns observers or third-party instances.

Expression changed errors

These errors expose a state mutation during a check that made the rendered result unstable. Fix ownership and timing rather than adding arbitrary `setTimeout`.

Performance model

Signals reduce the affected graph; component boundaries, stable identity, deferred code, DOM size, JavaScript work, and network latency still matter. Measure with browser performance tools and Angular DevTools.

Feynman check

Zoneless does not mean “Angular never checks.” It means Angular checks in response to explicit notifications. Name the notification for every state update on one page.

\newpage

06. Routing, lazy loading, guards, and route data

The router is an application composition and navigation system.

Route design

Prefer routes aligned to user capabilities and ownership boundaries. Lazy-load feature components or route arrays.

```
{
  path: 'orders',
  loadComponent: () => import('./orders/orders.page')
    .then(m => m.OrdersPage),
  providers: [OrdersStore],
}
```

Route-scoped providers isolate feature state and clean it up when the route leaves.

Parameters and data

Read route parameters as reactive state where practical. Validate and map URL strings into domain types. Resolvers can fetch critical navigation data, but they delay route completion; use page-level loading when progressive display is better.

Guards

Client guards improve navigation, not backend security. The server must enforce authorization. Return a `UrlTree` or redirect command rather than navigating as a side effect.

Nested layouts

Use parent routes for shared shells, context, and providers. Keep route configuration readable; giant nested arrays conceal product structure.

Preloading and view transitions

Preload based on likely navigation and network cost. View transitions can improve continuity but require reduced-motion handling and stable semantics.

Error and not-found routes

Make denied, missing, offline, and unexpected error states distinct. Preserve useful attempted destinations after authentication where safe.

Feynman check

A route is a URL contract plus code boundary, provider scope, and navigation policy. Explain what should happen when a deep link loads directly after a browser refresh.

\newpage

07. Reactive Forms and Signal Forms

Angular 22 offers established Reactive Forms and newer Signal Forms. Choose from maturity, ecosystem, typing, and project requirements.

Reactive Forms

Reactive Forms use explicit `FormControl`, `FormGroup`, and `FormArray` models. They are production-proven and work well for complex validation and existing component libraries.

Signal Forms

Signal Forms build a typed field tree over a writable signal model, with schema validation and `[formField]` binding. Angular documentation recommends Reactive Forms when production stability guarantees or

existing migrations dominate.

```
model = signal({ email: '', displayName: '' });
profileForm = form(this.model, path => {
  required(path.email);
  email(path.email);
});
```

Validation

Client validation improves feedback; the backend remains authoritative. Show errors after a meaningful interaction or submit, associate messages with fields, and focus the first invalid control when helpful.

Async validation

Debounce and cancel remote checks. Distinguish unavailable validation service from “value is invalid.” Never send a request for every keystroke without control.

Submission

Prevent duplicate submission, preserve values on recoverable failure, map server field errors, expose global errors, and make success explicit.

Custom controls

Custom controls must support labels, focus, keyboard behavior, disabled state, errors, touched/dirty semantics, and the selected forms integration. A pretty div is not a form control.

Feynman check

The form model is a draft. Validation is advice until the server accepts the command. Explain each state from untouched to submitting, failed, and complete.

\newpage

08. HTTP, resources, interceptors, and API contracts

HttpClient is Angular's typed transport API. Type parameters describe expected data; they do not validate responses.

Read paths

Use `httpResource` for signal-oriented reactive reads and HttpClient Observables for explicit composition. Resources are eager when their parameters are valid; HttpClient Observables are cold and request on subscription.

Mutations

Commands such as create, approve, and delete need explicit lifecycle: idle, submitting, success, validation conflict, authorization failure, network failure, and retry policy. Do not hide all failures behind “Something went wrong.”

Functional interceptors

Interceptors handle cross-cutting transport concerns: credentials, correlation, approved retry, timing, and error translation. Keep business decisions in application workflows.

Authentication

Prefer secure, short-lived identity. If using cookies, design CSRF protection. If using bearer tokens, avoid unsafe persistent storage and defend against XSS. The backend validates identity, audience, expiry, and authorization.

Cancellation and timeouts

Navigation, changing resource parameters, or destroying an owner should cancel obsolete work. Apply timeouts according to user action and backend contract.

Runtime validation

Validate critical remote data at the boundary. Map DTOs into domain/view models. Never render server-provided HTML without a reviewed sanitization contract.

Feynman check

The generic in `http.get<Order>()` is a promise you make to the compiler. It is not proof about bytes from the network. Explain where that proof occurs.

\newpage

09. RxJS and signal interoperability

RxJS models asynchronous streams with time, cancellation, and operators. Signals model current synchronous state. Angular applications often need both.

Flattening operators

- `switchMap`: cancel obsolete work; good for search;
- `concatMap`: queue in order; good for serialized commands;
- `exhaustMap`: ignore new triggers while busy; useful for submit protection;
- `mergeMap`: run concurrently; bound concurrency when needed.

The operator encodes product behavior. Choose it deliberately.

Ownership

Prefer AsyncPipe, `toSignal`, or `takeUntilDestroyed` to bind subscription lifetime to Angular ownership. Nested subscriptions hide cancellation and error flow.

Subjects

Subjects are both producer and consumer. Use them for actual event bridges, not as mutable global state by default. BehaviorSubject offers a current value but can spread imperative writes.

Errors

An Observable normally terminates on error. Place `catchError` at the boundary whose fallback is valid. Returning an empty stream can silently remove critical data.

Sharing

`shareReplay` can cache and share, but its buffer, reference-count, completion, and stale/error behavior must be understood. It is not an application cache strategy by itself.

Interop strategy

Keep server event streams, websockets, and complex async workflows in RxJS. Expose stable current UI state as signals. Avoid converting back and forth at every method.

Feynman check

A signal answers “what is the value now?” An Observable answers “what values arrive over time?” Explain why a search result page may use both.

\newpage

10. Feature architecture and state ownership

Organize by business capability, not by file type.

Vertical feature shape

```
orders/
  data-access/
  domain/
  feature-list/
  feature-detail/
  ui/
  orders.routes.ts
```

Keep dependencies flowing toward stable domain/application contracts. Shared must not become a dumping ground.

State categories

- server state: remote truth, caching, freshness, invalidation;
- URL state: navigation and shareable filters;
- feature state: workflow state scoped to one capability;
- component state: local interaction;
- persisted preference: deliberately versioned browser state.

Put each state in the narrowest owner that serves all real consumers.

Stores

A small signal service can be a good feature store. Larger event/state libraries help when auditability, devtools, entity patterns, or cross-feature workflows justify them. Do not adopt global state before defining ownership.

Ports and adapters

Application workflows depend on interfaces or stable services; HTTP, storage, analytics, and browser APIs are adapters. This makes testing and SSR boundaries clear.

Libraries and monorepos

Libraries need consumers, API contracts, ownership, and independent value. Enforce boundaries with lint/build tooling. Too many tiny libraries create versioning and navigation cost.

Feynman check

Draw one piece of state and its owner. List every reader and writer. If many unrelated features write it, the product model may be unclear.

\newpage

11. Tailwind CSS 4.3 with Angular

Tailwind is a utility-first CSS engine. Version 4 uses CSS-first configuration, modern CSS features, and dedicated integration packages.

Angular setup

Install compatible Tailwind and PostCSS packages, configure `@tailwindcss/postcss`, and import Tailwind:

```
@import "tailwindcss";

@theme {
  --color-brand-500: oklch(0.62 0.22 25);
  --breakpoint-3xl: 120rem;
}
```

Tailwind 4.3 includes scrollbar utilities, additional logical properties, container-size, zoom, tab-size, and improved variants.

Source detection

Tailwind scans source text for complete class tokens. Runtime-built fragments such as `'bg-' + color + '-500'` are invisible. Map variants to complete static strings or safelist only a constrained, justified set.

Design tokens

Define semantic tokens—surface, text, border, action, danger—not arbitrary component colors everywhere. Components consume the token system through utilities or CSS variables.

@apply

Use `@apply` sparingly for stable repeated primitives or third-party integration. It can hide utility composition and create CSS coupling. Angular components can share semantic primitives without rebuilding Bootstrap-like class hierarchies.

Responsive and logical layout

Start mobile-first. Use container queries when a component responds to its available space rather than viewport. Prefer logical properties for RTL and writing-mode resilience.

Migration traps

Tailwind v4 moves PostCSS integration to `@tailwindcss/postcss`, configuration into CSS, and performs stricter utility validation. Unknown custom utilities usually mean missing theme mapping—not a warning to suppress.

Feynman check

Tailwind generates CSS from class tokens it can find. It does not understand your Angular variables. Explain how every dynamic visual variant reaches the generated stylesheet.

\newpage

12. Flowbite 4 with Angular and SSR

Flowbite provides Tailwind-based visual components and JavaScript behaviors. Version 4 introduces theme CSS variables and Tailwind v4-oriented styling.

Integration

Flowbite's current Angular guide imports a theme, registers the plugin, and adds its source:

```
@import "tailwindcss";
@import "flowbite/src/themes/default";
@plugin "flowbite/plugin";
@source "../node_modules/flowbite";
```

Pin the current package version and verify the actual generated CSS and JavaScript. The documentation CDN examples may lag npm patch releases.

Ownership with Angular

Data attributes can initialize modals, dropdowns, tabs, tooltips, and drawers. Angular must still own business state and lifecycle. Repeatedly calling `initFlowbite()` after route changes can duplicate listeners or instances.

Prefer a wrapper directive/component or controlled service that:

- initializes after the element exists;
- stores the Flowbite instance;
- destroys it with `DestroyRef`;
- synchronizes open/closed state;
- exposes semantic inputs and outputs.

SSR

Flowbite accesses browser globals. Import it dynamically only in the browser, after render. Do not evaluate `document` on the server. Ensure the server HTML remains valid and stable for hydration.

Accessibility

Do not assume a styled example satisfies your product's accessibility. Verify dialog labeling, focus trap and restoration, escape behavior, keyboard navigation, active state, errors, touch targets, contrast, and reduced motion.

Flowbite Angular

Framework-specific wrappers may trail Angular majors. Evaluate compatibility, maintenance, API coverage, SSR behavior, accessibility, and whether a thin local wrapper around core Flowbite is safer.

Feynman check

Flowbite can animate a modal. Angular owns why it opens, what data it shows, what save means, and when its lifecycle ends.

\newpage

13. Design systems, accessibility, responsive UI, and i18n

A design system is a governed contract, not a gallery of components.

Layers

1. design primitives: color, type, spacing, motion;
2. semantic tokens: surface, action, border, status;
3. accessible interaction primitives;
4. product components;
5. page composition and usage guidance.

Tailwind expresses tokens and composition. Flowbite or CDK can supply starting behaviors. Angular packages ownership and contracts.

Accessibility

Use semantic HTML first. Every interactive element must be keyboard reachable, visibly focused, named, and operable without pointer or color alone. Announce meaningful dynamic changes without creating screen-reader noise.

Dialogs require focus placement, trap, escape/close policy, background inertness, labeling, and focus restoration. Menus are not generic navigation.

Responsive behavior

Design with real content, zoom, narrow screens, large text, RTL, empty states, errors, long translations, and slow networks. A mobile sidebar must have its own scroll area with the final action reachable.

Internationalization

Keep user-facing text extractable. Use locale-aware number/date/currency APIs. Do not concatenate translated sentence fragments or use visual direction words as logic.

Governance

Components need owners, tests, accessibility expectations, deprecation policy, changelog, and migration path. Count consumers before breaking shared behavior.

Feynman check

A token is a shared decision. A primitive enforces interaction. A component solves a repeated product need. Explain which layer owns “danger red” and which owns “delete invoice.”

\newpage

14. Angular Material, CDK, and choosing UI primitives

Angular Material is a Material Design component library. The Angular CDK provides lower-level behavior such as overlays, portals, drag/drop, focus, scrolling, tables, and accessibility utilities.

Keep majors aligned

Angular Material and CDK should normally match the Angular major. Update through supported schematics and inspect theme, typography, density, and component API changes.

Material versus Flowbite

Material offers Angular-native components with deep CDK integration and a defined visual system. Flowbite offers Tailwind-oriented markup, themes, and JavaScript behaviors. CDK can underpin a custom Tailwind design system.

Choose one primary interaction ownership model for each component. Do not layer a Flowbite modal controller over a Material dialog.

Overlay

Overlay handles stacking, position, scroll strategies, backdrop, portals, and cleanup. Use it for custom menus, tooltips, and dialogs when CDK behavior is worth the dependency.

Harnesses

Material/CDK component harnesses offer stable semantic testing APIs. Local design-system components can provide their own harnesses or role-based testing contracts.

Theming

Map brand tokens deliberately. Do not mix Material theming, Flowbite theme variables, and Tailwind colors without one source of truth.

Feynman check

Flowbite is a styled component kit. Material is Angular-native Material Design. CDK is an interaction toolbox. Explain which one owns focus and overlay positioning in your dialog.

\newpage

15. SSR, SSG, hybrid rendering, and hydration

Angular can render routes on the client, per request on the server, or ahead of time as static pages.

Choose per route

- CSR: private applications and highly interactive authenticated pages;
- SSR: dynamic public content needing first response and SEO;
- SSG: stable public content that can build ahead;
- hybrid: one application with route-specific strategies.

Rendering mode is a product/cache/security decision, not a checkbox.

Hydration

Hydration reuses server DOM on the client. Server and client output must match. Invalid HTML, direct DOM mutation, random/time differences, browser-only code, and third-party scripts can cause mismatch.

Incremental hydration

Incremental hydration combines server-rendered `@defer` content, event replay, and triggers such as interaction, viewport, idle, hover, timer, or condition. It reduces initial JavaScript while preserving initial content.

Data and privacy

TransferState/resource IDs can prevent duplicate server/client requests. Never serialize user-specific secrets into cacheable HTML. Define cache keys, vary headers, authentication, and invalidation.

Browser APIs

Guard `window`, `document`, storage, observers, and Flowbite imports. Use `isPlatformBrowser` and render callbacks where appropriate. Prefer injectable browser ports for testability.

Stability

Pending async work can delay server serialization and hydration cleanup. Temporarily use Angular's stability debugging tools and task tracking; remove debug tooling from production bundles.

Feynman check

SSR paints HTML. Hydration connects Angular behavior to it. Incremental hydration connects selected islands later. Explain what a click does before its island hydrates.

\newpage

16. Performance, bundles, rendering, and Core Web Vitals

Performance is a user outcome under real devices, networks, data, and load.

Budgets

Set Angular build budgets for initial, route, and component-style output. Track compressed transfer and parse/execute cost, not only raw artifact size.

Code delivery

Lazy-load feature routes. Use `@defer` for heavy below-the-fold or interaction- driven content. Avoid importing large libraries through shared barrels that pull them into the initial graph.

Rendering

Use stable `track` identity, immutable state, signals, OnPush-compatible components, virtual scrolling for large lists, and pagination for truly large data. A fast diff still cannot render 50,000 useful DOM nodes.

Images and fonts

Size media explicitly, use responsive sources, modern formats, priority only for real LCP assets, and subset/self-host fonts according to policy. Prevent layout shift.

Tailwind and Flowbite

Verify generated CSS source detection and minimize unused integration surfaces. Initialize only the interactive components needed. Avoid duplicate icon, datepicker, and JavaScript bundles.

Measure

Use Core Web Vitals, browser performance traces, Angular DevTools, bundle analysis, network waterfalls, user timing, and real-user monitoring. Compare p50 and p75/p95, not a single fast laptop run.

Feynman check

The initial bundle is a suitcase the user must download, unpack, and inspect before interaction. Name what you can leave for the next route or next click.

\newpage

17. Vitest, component tests, Playwright, and visual confidence

New Angular CLI projects use Vitest and jsdom by default. Real browser tests remain essential for layout, accessibility, and platform behavior.

Test pyramid by evidence

- pure tests: transformations and domain rules;
- service/store tests: workflows and state transitions;
- component tests: template semantics and user interaction;
- HTTP tests: request contract and failure mapping;
- browser tests: routing, auth, focus, responsive behavior, hydration;
- visual checks: deliberate appearance regressions.

Vitest and TestBed

Use TestBed when Angular injection, templates, or lifecycle are the subject. Use plain Vitest for ordinary TypeScript. Query by accessible roles, labels, and visible outcomes rather than CSS implementation classes.

HTTP testing

Provide HttpClient before `provideHttpClientTesting`, expect exact requests, flush success and failures, and verify no unexpected requests remain.

Browser mode and Playwright

Angular's test runner can use browser providers. Playwright E2E covers Chromium, Firefox, and WebKit. Test keyboard use, mobile drawers, focus restoration, deep links, offline/errors, and real server responses.

Flowbite components

Test one initialization per element, destroy cleanup, open/close synchronization, escape/backdrop behavior, focus trap, route transitions, and SSR browser guards.

Avoid brittle tests

Do not assert hundreds of Tailwind class tokens unless the utility is the contract. Prefer semantic outcome and a small number of intentional visual snapshots.

Feynman check

A unit test proves code. A component test proves Angular rendering. A browser test proves the platform interaction. A visual test proves selected pixels.

\newpage

18. Angular frontend security and trust boundaries

The browser is an untrusted client. Angular reduces some rendering risk but cannot enforce backend authorization or protect secrets delivered to users.

XSS

Angular escapes interpolated values and sanitizes supported contexts. Avoid `innerHTML`; never use `bypassSecurityTrust*` to silence warnings without a reviewed provenance and sanitization contract.

URLs, styles, HTML, and resource URLs have different trust rules. Content Security Policy and Trusted Types add defense in depth.

Authentication and authorization

Client route guards and hidden buttons improve experience only. APIs enforce permissions and resource ownership. Avoid storing long-lived bearer tokens in JavaScript-readable storage when safer architecture is

available.

CSRF and CORS

Cookie-authenticated state changes need CSRF defenses. CORS is a browser read-policy, not authentication. Configure exact trusted origins, credentials, methods, and headers.

Supply chain

Pin lockfiles, review install scripts and maintainers, scan dependencies, generate an SBOM, use trusted registries, sign artifacts, and patch supported Angular/Node/runtime versions. Tailwind and Flowbite are build/runtime dependencies with supply-chain impact.

Sensitive data

Never place secrets in environment files bundled into JavaScript. Treat logs, analytics, source maps, error reports, browser storage, SSR HTML, and Transfer State as possible disclosure paths.

Deployment headers

Set CSP, frame-ancestors, nosniff, referrer policy, permissions policy, HSTS, safe caching, and immutable hashed assets. Do not cache personalized SSR HTML publicly.

Feynman check

If the browser can use a secret, the user can inspect it. Explain why frontend code can guide authorization but never be its final authority.

\newpage

19. Real-life scenario: OpsCanvas

OpsCanvas is a global operations portal for logistics teams. Operators monitor shipments, resolve exceptions, edit schedules, and approve high-risk changes on desktop and rugged tablets. Public status pages need SEO; the authenticated control room does not.

Product requirements

- 50,000 live shipments with server filtering and virtualized results;
- deep links to every exception and preserved URL filters;
- real-time status events without losing current UI state;

- accessible keyboard-first dialogs and command palette;
- tenant and region authorization enforced by APIs;
- offline-aware read state and explicit mutation failure;
- English, Arabic/RTL, and Japanese;
- staged upgrades with observable performance budgets.

Feature architecture

```
app-shell/
shipments/
  domain/
  data-access/
  feature-board/
  feature-detail/
  ui/
exceptions/
approvals/
status-public/
design-system/
```

Each feature route lazy-loads and owns a scoped store. URL signals own filters that must survive sharing. Component signals own panel and selection state. HttpClient/httpResource owns server reads; RxJS owns the live event stream.

State flow

The shipment board derives visible rows from route filters plus paged server results. A websocket Observable merges validated events into the feature store. `computed` produces counters and selected detail. No effect copies one signal into another.

Tailwind and design tokens

Tailwind 4.3 defines semantic CSS variables for surface, text, action, warning, danger, focus, density, and logical spacing. Container queries let the detail panel adapt when docked. RTL uses logical properties rather than left/right assumptions.

Flowbite ownership

Flowbite 4 provides starting markup/themes for dropdowns, datepicker, and drawers. OpsCanvas wraps each imperative behavior in an Angular directive that initializes after render, records its instance, exposes semantic events, and destroys on route removal. The approval dialog uses CDK Overlay and focus tools because security confirmation, trap, restoration, and nested overlays need Angular-native control.

Rendering strategy

Public shipment status routes use SSG or SSR with hydration. Authenticated control-room routes use CSR behind secure APIs. Flowbite imports only in browser wrappers. User-specific resources are never serialized into shared cacheable HTML.

Command workflow

“Approve reroute” has:

1. backend-fetched permission and current version;
2. accessible confirmation with reason;
3. one in-flight submission using `exhaustMap` or explicit state;
4. idempotency key and optimistic concurrency version;
5. 409 conflict retaining entered reason;
6. audit correlation ID;
7. success updating server state and announcing completion.

Testing evidence

- pure tests for rules and DTO parsing;
- Vitest component tests for loading, empty, denied, error, and success;
- HTTP tests for auth, conflict, and cancellation;
- Playwright for deep links, keyboard dialogs, mobile drawer scrolling, RTL, hydration, and expired sessions;
- axe/accessibility review plus manual screen reader and keyboard checks;
- bundle budgets and p75 Core Web Vitals in release gates.

Deployment

CI uses a supported Node line, Angular/CLI 22.1, TypeScript 6.0, locked npm dependencies, Tailwind 4.3, and Flowbite 4.0. Build once into an immutable image. Runtime configuration is validated before bootstrap. Hashed assets cache immutably; the SPA/SSR document does not become stale. A canary cohort proves errors, latency, and user outcomes before full rollout.

Failure walkthrough

If Flowbite initializes twice after navigation, dropdown listeners fire twice. The wrapper registry detects an existing instance, destroys during route cleanup, and a browser test proves one event per action. If hydration sees client-only DOM mutation, initialization moves to an after-render browser hook rather than skipping hydration for the whole page.

Glossary and abbreviations

Term	Plain meaning
CSR / SSR / SSG	Client / request-time server / build-time rendering
DI	Dependency injection
Signal / computed	Tracked value / derived tracked value
Resource	Signal-oriented managed asynchronous value
RxJS / Observable	Reactive Extensions / values over time
CVA	ControlValueAccessor for custom form controls
CDK	Angular Component Dev Kit
CWV	Core Web Vitals
LCP / CLS / INP	Loading, layout stability, interaction metrics
CSP / XSS / CSRF	Browser security policy / script injection / forged request
RTL / i18n	Right-to-left layout / internationalization
DTO	Data transfer object
E2E	End-to-end browser test
RUM	Real-user monitoring
SBOM	Software bill of materials

Official references

- Angular release and support policy
- Angular version compatibility
- Angular signals
- Angular zoneless guide
- Angular SSR and hydration
- Angular testing
- Tailwind CSS 4.3
- Flowbite Angular guide
- Flowbite changelog

Final Feynman challenge

Trace one reroute approval from URL and component, through signals, form, dialog, HTTP interceptor, backend conflict, state update, Tailwind/Flowbite rendering, accessibility announcement, telemetry, and

rollback.