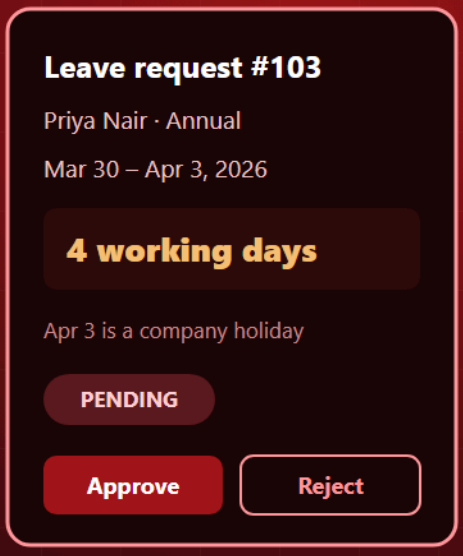
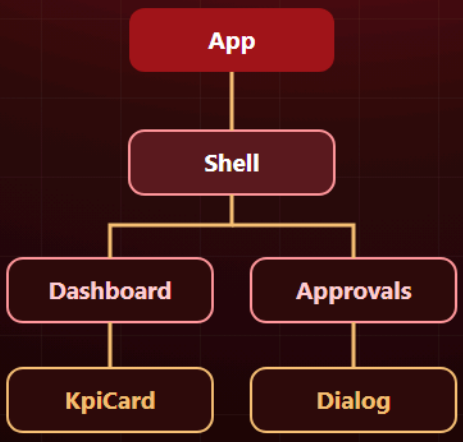





```
readonly searchTerm = signal('');
readonly filtered = computed(() =>
  this.employees().filter((e) =>
    matches(e, this.searchTerm())));

@for (e of filtered(); track e.id) {
  <app-employee-card [employee]="e" />
}
```



A COMPLETE, PRACTICAL GUIDE

Angular from Zero to Production

Build modern web apps with signals, standalone components, and RxJS

Components · Signals · Routing · HttpClient · RxJS · Forms
Signal Forms · Vitest · Security · Performance · SSR · Deployment

ONE REAL PROJECT · AN HR PORTAL, BUILT STEP BY STEP

THENEXBOOKS

Angular
from Zero to Production

Build Modern Web Apps with Signals, Standalone Components, RxJS, Forms, Testing, and Deployment
From your first component to a tested, secure, deployed business application — built step by step

First Edition

THENEXBOOKS

Angular from Zero to Production
Build Modern Web Apps with Signals, Standalone Components, RxJS, Forms, Testing, and Deployment

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Code examples may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Angular, its command-line tools, and the libraries around it change frequently; APIs, defaults, generated files, and command options can differ between versions. Always consult the official documentation for the version you use.

Angular is a trademark of Google LLC. TypeScript, Visual Studio Code, and Windows are trademarks of the Microsoft group of companies. Node.js is a trademark of the OpenJS Foundation. Docker is a trademark of Docker, Inc. GitHub is a trademark of GitHub, Inc. Other product names are trademarks of their respective owners. Their use in this book is for identification only and does not imply endorsement. This book is independent and is not affiliated with, sponsored by, or endorsed by Google LLC or the Angular team.

Brightwater Labs, the company used as the running example in this book, is fictional, as are its employees, departments, leave requests, and all sample data. Any resemblance to real people or organizations is coincidental. Email addresses use the reserved example domain.

Preface

Most web applications that companies depend on are not glossy marketing sites. They are the tools people open every morning: the directory, the request form, the approval queue, the dashboard that tells a manager what needs attention today. These applications live for years, are built by teams that change over time, and must keep working as requirements grow. They need structure, predictable patterns, strong typing, good tests, and a clear path to production.

Angular was designed for exactly this kind of work. It is a complete framework rather than a single library: components and templates, a router, forms, an HTTP client, dependency injection, testing support, and a command-line tool that creates, builds, tests, and updates your project. In recent versions Angular has also changed a great deal. Standalone components replaced most of the old module boilerplate, signals bring a simple and precise model for reactive state, the new template control flow is easier to read, new projects run without zone.js, and the toolchain builds and tests faster than before. If you learned Angular years ago, much of it will feel new; if you are starting today, you can learn the modern way from the first page.

This book teaches Angular from the very beginning and takes you all the way to a production-quality application. We do not just tour the APIs. We build one real system step by step, introducing each feature at the moment it solves a problem we actually have, and explaining *why* it works the way it does, so you can design your own Angular applications with confidence.

Who This Book Is For

- **Web developers who know HTML, CSS, and JavaScript** and want to build larger, structured applications with a framework that scales to teams.
- **Backend and full-stack developers** (Java, .NET, Node.js, Python) who need to build the front end of business applications and appreciate types, dependency injection, and clear architecture.
- **Developers coming from older Angular** who want to catch up with standalone components, signals, the new control flow, zoneless change detection, Signal Forms, and the modern testing setup.
- **Students and career changers** who want to learn a framework widely used in enterprise and government projects through a realistic project rather than isolated snippets.

You do not need prior experience with Angular, TypeScript, or RxJS. Chapter 2 teaches the TypeScript the book uses, and Part III introduces RxJS from first principles.

What You Will Learn

- How a modern Angular project is structured, what the CLI generates, and how to build, serve, and test it.

- Components and templates: bindings, the built-in control flow (`@if`, `@for`, `@switch`, `@defer`), styles, and content projection.
- Signals: `signal`, `computed`, `effect`, `linkedSignal`, signal inputs, outputs, and models, and how they drive change detection.
- Directives, pipes, services, and dependency injection, and how to organize a growing codebase into features.
- Routing: parameters, lazy loading, guards, resolvers, route input binding, layouts, and page titles.
- Data and RxJS: HttpClient, interceptors, Observables and operators, search-as-you-type, error handling and retries, and async data with `toSignal`, `resource`, and `httpResource`.
- Authentication and authorization: login, tokens, interceptors, role-based navigation, and why the server must have the final say.
- Forms three ways: template-driven forms, typed reactive forms, and Signal Forms, with built-in, custom, cross-field, and async validation.
- Testing with Vitest and TestBed: components, services, HTTP, the router, and forms, plus end-to-end and accessibility testing.
- Production: security (XSS, sanitization, XSRF, token handling), performance (OnPush, zoneless, `@defer`, budgets), server-side rendering and hydration, environments, Docker, static hosting, CI, error handling, monitoring, and upgrades.

How This Book Is Organized

Part I — Foundations (Chapters 1–5) introduces Angular and the CLI, the TypeScript you need, components and templates, signals, and communication between components. **Part II — Building Blocks** (Chapters 6–10) covers directives and pipes, services and dependency injection, routing and advanced routing, and how to structure and style a real application. **Part III — Data and RxJS** (Chapters 11–15) connects the app to a backend with HttpClient, teaches RxJS fundamentals and practical patterns, combines signals with async data, and adds authentication and authorization. **Part IV — Forms** (Chapters 16–20) covers template-driven, reactive, and Signal Forms, validation of every kind, and the complete leave request and approval workflow. **Part V — Testing and Security** (Chapters 21–24) covers unit, component, HTTP, router, end-to-end, and accessibility testing, and the security practices every Angular application needs. **Part VI — Production** (Chapters 25–28) covers performance, server-side rendering and hydration, building and deploying, and a final case study on error handling, monitoring, and upgrades. The appendices provide a cheat sheet, templates and reference material (including the complete mock API), a production checklist, and a glossary.

Each chapter ends with exercises and a **Key Takeaways** box for quick review.

The Running Example: Brightwater HR Portal

Throughout the book we build the internal HR portal of **Brightwater Labs**, a fictional software company with four departments — Engineering, Human Resources, Sales, and Finance — and eleven employees in our sample data. Employees browse the company directory, check their leave balance, and request annual, sick, or unpaid leave. Managers review and approve or reject their team's requests. HR administrators create and edit employee records and can act on any request. Everyone starts the day on a dashboard that shows headcount, pending approvals, and who is on leave today.

The portal starts as a single static component and grows chapter by chapter into a complete application: a responsive layout with role-aware navigation, a searchable employee directory, department pages, a leave request form with live working-day calculation and validation, an approval queue with dialogs and notifications, a signal-based store, authentication against a small mock API, a full test suite, and a production build ready to deploy. The sample data is small enough to check by hand, and the book uses the same names, numbers, and dates everywhere, so every result you see can be reproduced.

Versions and Conventions

The examples target **recent versions of Angular** and were checked against a project generated by a recent Angular CLI, using TypeScript, RxJS 7, and Vitest as that CLI sets them up. Angular evolves quickly: when a default changed recently — for example zoneless change detection, OnPush as the default strategy, Vitest as the default test runner, or file names without the `.component` suffix — a Note tells you what older projects look like. APIs that are still marked as developer preview or experimental in the version we used are labeled as such. You need a current Node.js LTS release; check the version compatibility table in the Angular documentation for your Angular release.

> **TIP**
> A recommended practice or a technique that makes your work easier.

> **NOTE**
> Background information, version differences, or a useful aside.

> **WARNING**
> A common mistake or something that can cause bugs, data loss, or security problems.

Code appears in `monospace`. A caption above each listing shows the file path relative to the project root, such as `src/app/core/auth-guard.ts`, or `Terminal` for commands. Long lines are wrapped to fit the page. Commands are shown for a terminal opened in the `hr-portal` folder; they work the same in

PowerShell, Command Prompt, macOS Terminal, and Linux shells unless a Note says otherwise. Dates in the sample data use the ISO format `2026-03-30`, and the book's "today" for every example is Monday, March 16, 2026.

Contents

Preface

PART I — FOUNDATIONS

1 Angular in Context: The Framework, the CLI, and Your First Project

2 TypeScript Essentials for Angular Developers

3 Components and Templates: Bindings, Control Flow, and Styles

4 Signals: Reactive State with signal, computed, and effect

5 Component Communication: Inputs, Outputs, Models, and Content Projection

PART II — BUILDING BLOCKS

6 Directives and Pipes: Reusable Behavior and Formatting

7 Services and Dependency Injection

8 Routing and Navigation

9 Advanced Routing: Guards, Resolvers, Lazy Loading, and Route Inputs

10 Application Structure: Layouts, Shared UI, and Styling

PART III — DATA AND RXJS

11 Talking to a Backend with HttpClient

12 RxJS Fundamentals: Observables, Operators, and Subscriptions

13 RxJS in Practice: Search, Combining Streams, Errors, and Retries

14 Async Data with Signals: toSignal, resource, httpResource, and Signal Stores

15 Authentication and Authorization: Login, Interceptors, and Role-Based Access

PART IV — FORMS

16 Template-Driven Forms

17 Reactive Forms: Typed FormGroup, FormControl, and FormArray

18 Validation: Built-in, Custom, Cross-Field, and Async Validators

19 Signal Forms: form(), Schemas, and the formField Directive

20 The Leave Request Workflow: Approvals, Dialogs, and Form UX

PART V — TESTING AND SECURITY

21 Unit Testing with Vitest and TestBed

22 Testing Components, Services, HTTP, and the Router

23 End-to-End and Accessibility Testing

24 Security: XSS, Sanitization, XSRF, and Safe Token Handling

PART VI — PRODUCTION

25 Performance: Change Detection, Zoneless, @defer, and Bundle Budgets

26 Server-Side Rendering, Hydration, and Prerendering

27 Building and Deploying: Environments, Docker, Static Hosting, and CI

28 Case Study: Error Handling, Monitoring, Upgrades, and the Finished HR Portal

APPENDICES

A Angular Cheat Sheet

B Templates and Reference

C Production Checklist

D Glossary

PART I

Foundations

What Angular is and when it fits, the CLI and the files it generates, the TypeScript every Angular developer needs, components and templates with the built-in control flow, reactive state with signals, and the ways components talk to each other.

1 · Angular in Context

2 · TypeScript Essentials

3 · Components and Templates

4 · Signals

5 · Component Communication

CHAPTER 1 · PART I — FOUNDATIONS

Angular in Context: The Framework, the CLI, and Your First Project

Before we write a single component, we need to know what Angular is, what problems it solves well, and how its tooling turns a folder of TypeScript files into a running web application. By the end of this chapter you will have a real Angular project on your machine, you will know what every generated file does, and you will have made your first change to the Brightwater HR Portal that grows throughout this book.

What Angular Is

Angular is a complete, opinionated framework for building web applications in TypeScript. "Complete" is the important word. Many front-end tools focus on one job, usually rendering a user interface from data, and leave everything else to separate libraries that you choose, combine, and keep compatible yourself. Angular ships the whole toolbox in one coordinated release:

- **Components and templates** — the building blocks of the user interface, written as TypeScript classes plus an HTML-based template language with bindings and built-in control flow.
- **Signals** — a reactive state primitive that tells Angular exactly which parts of the screen depend on which values.
- **Dependency injection (DI)** — a system that creates and shares services such as an API client, a clock, or a notification center, and lets tests replace them with fakes.
- **A router** — maps URLs such as `/employees/3` to screens, with guards, lazy loading, and route parameters.
- **Forms** — three approaches (template-driven, reactive, and the newer Signal Forms) with validation built in.
- **HttpClient** — a typed HTTP client with interceptors for cross-cutting concerns such as authentication headers and error handling.
- **Testing utilities** — `TestBed` and friends for unit and component tests, wired into the test runner that new projects use.
- **The Angular CLI** — the command-line tool that creates projects, generates code, runs the development server, builds for production, runs tests, and applies automated migrations when you upgrade.

"Opinionated" is the second important word. Angular has a preferred way to structure an application, to share code, to talk to a server, and to test. You can deviate, but the default path is well paved, documented, and the same in every Angular codebase. That consistency is the reason many teams

choose Angular: a developer who joins a project already knows where the routes live, how services are provided, and how to run the tests.

When Angular fits, and when something lighter is enough

Angular shines in larger applications that live for years and are built by teams: internal business tools, administration consoles, dashboards, banking and insurance portals, line-of-business software with many forms and many screens. Those projects benefit from strong typing, one coherent toolset, automated upgrade migrations, and shared conventions.

Angular is less of a natural fit for a small marketing page, a single interactive widget embedded in a content site, or a weekend prototype where you want to write a few lines of script and be done. You can build those with Angular, but a smaller tool, or no framework at all, may be the more economical choice. Choosing a framework is an engineering trade-off, not a matter of loyalty.

Other popular approaches, such as React or Vue, make different trade-offs. They typically focus on the view layer and let you assemble routing, forms, data fetching, and testing from a wider ecosystem, which gives more freedom of choice and asks for more decisions. Angular includes those pieces and decides more for you. Teams build excellent software with all of them; this book simply focuses on doing it well with Angular.

AngularJS is not Angular

You may meet two names on the web. **AngularJS** (version 1.x) was the original JavaScript framework with controllers, `$scope`, and two-way data binding everywhere. **Angular** (without "JS") is a complete rewrite in TypeScript with a component model, a compiler, and a different architecture. The two share a name and some ideas but not code; an AngularJS tutorial will not help you with modern Angular. Modern Angular has also changed considerably over its own lifetime: standalone components, signals, and the built-in `@if` and `@for` blocks replaced older patterns. This book teaches the modern style throughout and points out the older style only where you are likely to meet it in existing code.

Meet Brightwater Labs and the HR Portal

Learning a framework from isolated snippets is like learning carpentry by looking at single nails. In this book we build one realistic application from an empty folder to a deployable production build: the **Brightwater HR Portal**, the internal HR web app of **Brightwater Labs**, a fictional mid-sized software company. The company has four departments (Engineering, Human Resources, Sales, and Finance) and eleven employees in our sample data. The app's header reads **"Brightwater HR"**, and the project folder is called `hr-portal`.

The portal has three kinds of users, and each role sees a little more than the one before it:

Role	Demo user	What they can do

`employee`	Priya Nair, Senior Frontend Engineer	Browse the employee directory, request leave, see and cancel their own requests
`manager`	Marcus Lee, Engineering Manager	Everything an employee can do, plus approve or reject their team's leave requests
`hr-admin`	Amelia Hart, HR Director	Everything above, plus all approvals and creating and editing employee records

Everyone gets a dashboard with headcount, pending requests, and who is on leave today. The book uses a fixed "today" for every example and test: **Monday, March 16, 2026**. Fixing the date means the numbers you see on your screen match the numbers in the book, no matter when you read it. On that day Brightwater has eleven employees, ten of them active, one on leave, and four leave requests waiting for a decision.

Later, when the app talks to a server, you will sign in as any of these people with their work email address (for example `priya.nair@brightwater.example`) and the shared demo password `demo1234`. The domain `brightwater.example` is a reserved example domain, so no real mailbox is ever involved.

How the project grows

The book has six parts, and the HR Portal grows with each one:

1. **Part I — Foundations** (Chapters 1–5): TypeScript, components, templates, and signals. We build static and then reactive screens from in-memory sample data.
2. **Part II — Building Blocks** (Chapters 6–10): directives, pipes, services, dependency injection, routing, and the application shell with navigation.
3. **Part III — Data and RxJS** (Chapters 11–15): a tiny mock API that runs with plain Node.js, HttpClient, RxJS, async data with signals, and real sign-in with role-based access.
4. **Part IV — Forms** (Chapters 16–20): all three form approaches, validation, and the complete leave request and approval workflow.
5. **Part V — Testing and Security** (Chapters 21–24): unit, component, HTTP, router, and end-to-end tests, accessibility, and the security topics that matter for an app full of personal data.
6. **Part VI — Production** (Chapters 25–28): performance, server-side rendering, building and deploying, error handling, monitoring, and upgrades.

Each chapter leaves the project in a working state and states which files it creates or changes.

Setting Up Your Machine

You need Node.js, the Angular CLI, and a code editor. Every command in this book works in PowerShell on Windows, in Terminal on macOS, and in any Linux shell.

Install Node.js

Angular's tools run on Node.js, and npm (the Node package manager) comes with it. Install a current **Node.js LTS** (long-term support) release from the official Node.js website or through your operating system's package manager. Each Angular release supports a specific range of Node.js versions, so before you start a new project, check the version compatibility table in the Angular documentation on angular.dev. If your Node.js version is outside the supported range, the CLI tells you so when you run it.

Terminal

```
node --version
npm --version
```

Both commands should print a version number. If your system still has an old Node.js installed for another project, a version manager such as nvm (macOS and Linux) or nvm-windows lets you switch between versions per project.

Install the Angular CLI

The Angular CLI is an npm package called `@angular/cli`. Installing it globally gives you the `ng` command everywhere:

Terminal

```
npm install -g @angular/cli
ng version
```

`ng version` prints the CLI version, the Node.js version, and your operating system. When you run it inside a project, it also lists every Angular package the project uses, which is the first thing to paste into a bug report or a question to a colleague.

> **TIP**
> Inside a project you can always run the project's own copy of the CLI with `npx ng` instead of the global `ng`. The local copy is the exact version the project was created with, so `npx ng build` behaves the same on every developer's machine and on a build server. We write plain `ng` in the book for brevity; both work.

Choose an editor

Any editor works, but Visual Studio Code with the official **Angular Language Service** extension gives you the best experience: autocompletion inside templates, errors underlined as you type (for example, a misspelled property in `{{ employee.lastNmae }}`), go-to-definition from a template to the component class, and quick information on hover. New projects even contain a

`.vscode/extensions.json` file that recommends the extension, so VS Code offers to install it the first time you open the folder. Other editors, including the JetBrains IDEs, have their own Angular support.

Creating the HR Portal Project

Move to the folder where you keep your projects and ask the CLI to create a new workspace:

Terminal

```
ng new hr-portal
```

The CLI asks a few questions before it generates anything. The exact prompts differ between versions. At the time of writing they include the stylesheet format (choose **CSS** to follow along exactly), whether to enable server-side rendering and static site generation (answer **No**; Chapter 26 discusses when SSR is worth it), and, in recent versions, whether to add configuration files for AI coding tools (any answer is fine; the book does not depend on them). If your CLI asks something else, the default answer is almost always the right one for this book.

The CLI then creates the files and runs `npm install` to download the dependencies. This takes a minute or two the first time. You should see something like this:

Output

```
CREATE hr-portal/.prettierrc (173 bytes)
CREATE hr-portal/angular.json (1985 bytes)
CREATE hr-portal/package.json (784 bytes)
CREATE hr-portal/README.md (1520 bytes)
CREATE hr-portal/tsconfig.json (939 bytes)
CREATE hr-portal/.editorconfig (331 bytes)
CREATE hr-portal/.gitignore (666 bytes)
CREATE hr-portal/tsconfig.app.json (412 bytes)
CREATE hr-portal/tsconfig.spec.json (423 bytes)
CREATE hr-portal/.vscode/extensions.json (134 bytes)
CREATE hr-portal/.vscode/launch.json (490 bytes)
CREATE hr-portal/.vscode/tasks.json (1020 bytes)
CREATE hr-portal/src/main.ts (228 bytes)
CREATE hr-portal/src/index.html (307 bytes)
CREATE hr-portal/src/styles.css (81 bytes)
CREATE hr-portal/src/app/app.spec.ts (707 bytes)
CREATE hr-portal/src/app/app.ts (304 bytes)
CREATE hr-portal/src/app/app.css (0 bytes)
CREATE hr-portal/src/app/app.html (20488 bytes)
CREATE hr-portal/src/app/app.config.ts (322 bytes)
CREATE hr-portal/src/app/app.routes.ts (80 bytes)
CREATE hr-portal/public/favicon.ico (15086 bytes)
- Installing packages (npm)...
```

The CLI also initializes a Git repository and makes a first commit, unless Git is missing or you pass `--skip-git`. Having that first commit is useful: whenever an experiment goes wrong, you can compare against a known good state. Change into the new folder for every command that follows:

Terminal

```
cd hr-portal
```

A Tour of the Generated Files

A fresh project is small. It is worth knowing every file, because you will touch most of them in this book.

```
hr-portal/
  angular.json            workspace and build configuration
  package.json            dependencies and npm scripts
  tsconfig.json           shared TypeScript settings
  tsconfig.app.json       TypeScript settings for the app
  tsconfig.spec.json      TypeScript settings for the tests
  .prettierrc             code formatting rules
  .editorconfig           editor indentation and charset rules
  .vscode/                recommended extension, debug and task settings
  public/
    favicon.ico           static files copied to the build as-is
  src/
    index.html            the single HTML page the browser loads
    main.ts               the entry point that starts Angular
    styles.css            global styles
    app/
      app.ts              the root component, class App
      app.html            the root component's template
      app.css             the root component's styles
      app.spec.ts         unit tests for App
      app.config.ts       application-wide providers
      app.routes.ts       the route table
```

src/main.ts: where everything starts

src/main.ts

```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { App } from './app/app';

bootstrapApplication(App, appConfig)
  .catch((err) => console.error(err));
```

This is the first of our code that runs. `bootstrapApplication` takes two arguments: the root component (`App`) and the application configuration (`appConfig`). Angular creates the root component, finds the matching element in `index.html`, and renders the component there. If anything fails during startup,

the `catch` logs the error to the browser console.

src/index.html: the one real page

src/index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>HrPortal</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

An Angular app is a **single-page application** (SPA): the browser loads this one HTML page, and from then on Angular swaps content in and out as the user navigates. The `<app-root>` element is a placeholder that matches the selector of the root component. The `<base href="/">` tag tells the router where the application lives; Chapter 27 shows how to change it when you deploy under a sub-path. You never add `<script>` tags for your own code here; the build inserts them for you. Change the title to `Brightwater HR` now, since that is what employees should see in their browser tab. In Chapter 9 a custom title strategy sets per-page titles such as "Dashboard | Brightwater HR".

src/app/app.ts: the root component

src/app/app.ts

```
import { Component, signal } from '@angular/core';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.html',
  styleUrl: './app.css',
})
export class App {
  protected readonly title = signal('hr-portal');
}
```

A component is a TypeScript class decorated with `@Component`. The decorator's metadata tells Angular how to use the class: the `selector` is the HTML tag that renders it (`app-root`, the element in `index.html`); `imports` lists the other components and directives its template uses (here `RouterOutlet`, the placeholder where routed pages appear); `templateUrl` and `styleUrl` point to its HTML template

and its CSS. The class body holds the component's state and behavior; `title` is a **signal**, a container for a value that Angular can watch. Chapter 3 takes components apart in detail and Chapter 4 is all about signals.

The CLI may list the metadata properties in a different order than we do. Order does not matter to Angular; in this book we always write `selector`, `imports`, the template, and the styles, in that order, because it reads from "what is it called" to "what does it look like".

Notice what is *not* there: no `standalone: true` flag, no NgModule. In recent versions every component is standalone by default and declares its own dependencies in `imports`.

src/app/app.config.ts and app.routes.ts

src/app/app.config.ts

```
import { ApplicationConfig, provideBrowserGlobalErrorListeners } from '@angular/core';
import { provideRouter } from '@angular/router';
import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [
    provideBrowserGlobalErrorListeners(),
    provideRouter(routes),
  ],
};
```

The application configuration is a list of **providers**: instructions that tell Angular's dependency injection system what services the whole app can use. `provideBrowserGlobalErrorListeners()` makes sure errors that escape your code (for example, an unhandled promise rejection) reach Angular's error handling, which we customize in Chapter 28. `provideRouter(routes)` enables the router with the route table from `app.routes.ts`, which starts empty:

src/app/app.routes.ts

```
import { Routes } from '@angular/router';

export const routes: Routes = [];
```

We fill this array in Chapter 8. Over the course of the book, `app.config.ts` gains HTTP support, interceptors, a title strategy, and a session restore step, but its shape stays the same: one array of providers.

The other files

File	Purpose
`app.html`	The root template. The CLI fills it with a large welcome page that we delete in a moment.
`app.css`	Styles that apply only to the root component. Empty at first.

`app.spec.ts`	Unit tests for `App`, run by Vitest.
`styles.css`	Global styles for the whole app. Chapter 10 adds the Brightwater color palette here.
`public/`	Static files (favicon, images, a `robots.txt`) copied unchanged into the build output.
`angular.json`	Workspace configuration: build, serve, and test targets, budgets, and build configurations.
`package.json`	Dependencies and npm scripts: `npm start` runs `ng serve`, `npm test` runs `ng test`.
`tsconfig*.json`	TypeScript settings; the app and test files have separate configurations that extend the shared one.
`.prettierrc`	Formatting rules for Prettier: single quotes and a line width of 100.

Open `angular.json` and find the `build` target. Its builder is `@angular/build:application`, its entry point is `"browser": "src/main.ts"`, and its `production` configuration defines **budgets**: limits on bundle size that turn into a warning or an error when the app grows too large. Chapter 25 shows how to read and adjust them. The `test` target uses `@angular/build:unit-test`, the builder that runs Vitest.

In `package.json`, the `dependencies` are the Angular packages your app ships to the browser plus RxJS; the `devDependencies` are development tools: the CLI, the build system, TypeScript, Vitest, jsdom, and Prettier.

> **NOTE**
>
> Older projects kept static files in `src/assets/` and referenced them as `assets/logo.svg`. New projects use the `public/` folder, and its contents are served from the site root, so `public/logo.svg` becomes `/logo.svg`. Both layouts work; the `assets` array in `angular.json` decides which folders are copied.

How an Angular App Starts

It helps to have a mental picture of what happens between typing a URL and seeing the page:

```
browser requests /          server returns index.html
        |
        v
index.html loads main-[hash].js   (the compiled app; the build adds the script tag)
        |
        v
main.ts: bootstrapApplication(App, appConfig)
        |   creates the root injector from appConfig.providers
        v
Angular finds <app-root> and renders App's template into it
        |
        v
<router-outlet /> shows the component for the current URL (none yet)
```

Angular compiles your templates ahead of time during the build. The browser never receives raw template HTML that it has to interpret; it receives efficient JavaScript instructions that create and update the DOM. That is why template mistakes, such as a typo in a property name, show up as build errors instead of surprises in production.

Running the Development Server

Start the development server:

Terminal

```
ng serve
```

The CLI builds the app in development mode, starts a local web server, and watches your files. You should see something like this:

Output

```
> Building...
√ Building...
Initial chunk files | Names         | Raw size
main.js             | main          |  3.22 kB |
styles.css          | styles        | 95 bytes |

                    | Initial total |  3.31 kB

Application bundle generation complete. [1.529 seconds]

Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
  Local:   http://localhost:4200/
```

Open `http://localhost:4200` in your browser (or start the server with `ng serve --open` to have it open automatically). The default port is 4200; if another program already uses it, the CLI offers to pick another one, or you can choose your own with `ng serve --port 4300`. Leave the server running in its own terminal while you work. Every time you save a file, the CLI rebuilds the changed parts and the browser reloads. Stop the server with **Ctrl+C**.

The development server is for development only; never put `ng serve` in front of real users. Chapter 27 covers real hosting.

Your First Change: Make It Brightwater HR

The generated `app.html` is a large placeholder page with links to documentation. It has done its job of proving that the project works. Replace the entire file with this: