**angular.json:**

angular.json is the **main configuration file** for an Angular workspace. It tells Angular CLI **how to build, serve, test, and lint** your application(s).

It defines **project structure, build options, styles, scripts**, environments, and **CLI behaviors**.

Defines the entry point (like main.ts)

Specifies SSR (server-side rendering) builds like main.server.ts

**package.json and package-lock.json**

**Purpose**: Lists dependencies and metadata for project.

 package.json: Direct dependencies, scripts, project info.

 package-lock.json: Exact versioning tree of installed packages.

**tsconfig\*.json Files**

|  |  |
| --- | --- |
| tsconfig.json | **tsconfig.json** is a configuration file for the **TypeScript compiler (tsc)**. |
|  |  |

|  |  |
| --- | --- |
| tsconfig.app.json | Settings for compiling the app (excluding tests). |

|  |  |
| --- | --- |
| tsconfig.spec.json | Settings for test files (like \*.spec.ts). |

tsconfig.json

It tells TypeScript:

* Which files to compile
* What compiler options to use (e.g., target ES version, module system)
* Which files or folders to include or exclude
* How to generate the output (e.g., source maps)

**.editorconfig**

* Editor preferences for consistent coding style (indentation, line endings, etc.)
* Works with most editors including VS Code.

### main.ts

* **Main entry point of the browser app**.

main.server.ts & server.ts

 main.server.ts: Bootstraps the server module.

 server.ts: Node Express server setup to serve Angular app with pre-rendering.

### index.html

* The root HTML file served to the browser.
* <app-root> here is where Angular renders the AppComponent.

| * **File** | **Purpose** |
| --- | --- |

|  |  |
| --- | --- |
| **environment**.ts | Default configuration (used for development) |

|  |  |
| --- | --- |
| **environment**.prod.ts | Used automatically when building with ng build --prod |

**app.component.html**

This is the **HTML template** of your root component.

This dynamically loads child components based on routing.

### app.component.scss

This is the **component-level styling** — scoped to app.component.

### app.component.spec.ts

This is the **unit test file** for the root component

## app.routes.ts

This file defines **your application routes**.

This is passed to provideRouter(routes) in main.ts.

**app.config.ts**

** Defining standalone providers**

** Custom bootstrap configurations**

ACLI command:  
**ng generate component component-name**

Why standalone over app.module.ts

With standalone: true, a component declares **it manages its own dependencies** and **doesn’t need to be declared in a module**:

|  |
| --- |
| No need to declare in AppModule. |

|  |  |
| --- | --- |
| ✅ **Better Modularity** | Each component is self-contained and easier to reuse. |

### 1. ****Entry Point:**** main.ts

* This is the first file that runs when your app starts.
* It bootstraps the Angular application by calling bootstrapApplication (in standalone apps) or platformBrowserDynamic().bootstrapModule(AppModule) (for modular apps).

### 2. ****App Component or Module Loads****

* AppComponent or AppModule acts as the **root** of your app.
* It loads the base layout and initializes routing and services.

**Routing (app.routes.ts or app-routing.module.ts)**

* Angular uses the router to determine **which component** to load based on the URL.

### 4. ****Component Loads****

* A specific component loads (like DashboardComponent).
* Its lifecycle hooks (ngOnInit) are triggered.
* In ngOnInit(), it often calls a **service** to fetch data from the backend.

### 5. ****Services and Dependency Injection****

* A component uses a service (like AuthService or ApiService) to communicate with backend APIs.
* Services are injected using Angular’s **Dependency Injection (DI)** system.

### ****6. HTTP Request via**** HttpClient

* Services use Angular’s HttpClient to send requests to the backend.

### ****7. Interceptors (Optional)****

* Angular interceptors (like AuthInterceptor) automatically attach tokens or handle errors before the request is sent or after response is received.

**8. Backend Receives Request**

* The backend (Node.js, Django, Java, etc.) processes the API request, connects to the database if needed, and sends a JSON response.

**🔁 9. Response Comes Back**

* The response returns to the frontend.
* It’s handled by the .subscribe() callback or via async pipe.

**📤 10. Data Rendered in the Component**

* The component receives data and renders it using Angular’s **template syntax**:

Working of project

### 1. ng serve kicks off the app:

* Angular CLI reads angular.json.
* Uses main.ts as the **entry point**.

2. main.ts (Entry Point)

* AppComponent is a **standalone component**.
* appConfig includes all necessary routing, providers, etc.

### 3. app.config.ts (Configuration for bootstrap)

📍 src/app/app.config.ts

This is where you configure:

* Routing
* HTTPClient
* Interceptors
* Any other providers

### 4. app.component.ts (Root Component)

📍 src/app/app.component.ts

This is the root standalone component. It will import things like RouterModule, CommonModule, and any other needed components:

This loads the first template, and routes are handled through <router-outlet>.

### 5. app.routes.ts (Routing)

📍 src/app/app.routes.ts

Contains route definitions for your app:

So when a user visits /login, Angular dynamically imports and renders the LoginComponent.

### Standalone Components (Like Login/Signup)

📍 src/app/components/auth/login/login.component.ts

hese components can:

* Import modules themselves (e.g., ReactiveFormsModule, HttpClientModule)
* Use services via DI
* Be loaded via routes using loadComponent

### Services

📍 src/app/services/auth.service.ts and others

Services (like AuthService) are marked as:

And can be injected directly into any component (including standalone components).

### 8. Interceptors

📍 src/app/interceptors/

These are added via provideHttpClient(withInterceptors(...)) in app.config.ts.

Used to:

* Add JWT tokens
* Catch HTTP errors globally

## Flow Recap (for Standalone Apps)

1. main.ts uses bootstrapApplication(AppComponent, appConfig)
2. app.config.ts sets up routing, HTTP, interceptors
3. AppComponent is rendered with <router-outlet>
4. app.routes.ts maps URLs to components using loadComponent
5. Components like LoginComponent are loaded on-demand
6. Components use services and forms as needed
7. Interceptors work globally on HTTP requests

Why standalone ?

### ****Simplicity — No More NgModules****

With standalone components:

* You don’t need to create AppModule or declare components in it.
* Each component **self-declares what it needs** (imports, providers, etc.).

**Better Lazy Loading**

You can lazy load a component directly.

With NgModules, you'd have to lazy load a **whole module**, which increases complexity and dependencies.

**Versions:**

**Package Version**

**---------------------------------------------------------**

**@angular-devkit/architect 0.1902.5**

**@angular-devkit/build-angular 19.2.5**

**@angular-devkit/core 19.2.5**

**@angular-devkit/schematics 19.2.5**

**@angular/cdk 19.2.7**

**@angular/cli 19.2.5**

**@angular/material 19.2.7**

**@angular/ssr 19.2.5**

**@schematics/angular 19.2.5**

**rxjs 7.8.2**

**typescript 5.7.3**

**zone.js 0.15.0**