Angular 17 Development Guide

Quick Introduction to Angular :

Angular is a frontend **TypeScript-based** framework used for building **dynamic web applications**. It allows developers to create reusable components and manage application state efficiently.

Install Prerequisites:

- Install Node.js (LTS Version) : https://nodejs.org/
- Install Angular CLI (Version 17) : npm install -g @angular/cli@17

Key Concepts to briefly explain:

- **Components**: UI elements (buttons, forms, etc.).
- **Directives**: Modify elements (`*ngIf`, `*ngFor`).
- **Services**: Handle logic & data (API calls, storage).
- **Routing**: Navigating between pages (`RouterModule`).
- **Dependency Injection**: Sharing data across components.

Common Commands :

- **Create a new Angular application:**

  ng new my-app

- **Serve the application locally:**

  ng serve

- **Generate a standalone component:**

  ng generate component components/my-component --standalone

- **Generate a service:**

  ```
  ng generate service shared/utils/my-service
  ```

This project follows **Atomic Design** and is built using **Angular 17 Standalone Components**. Below is the **folder structure** and how each part works.

Why Use Atomic Design :

- Better Component Reusability
- Improved Code Maintainability
- Faster Development with a Clear Structure
- Easier Debugging
- Works perfectly with **Angular 17's Standalone Components**

Project Folder Structure:

```
src/
├── app/
│   ├── components/     👈 (Re-usable UI Components)
│   │   ├── atoms/      👈 (Basic elements: Button, Input)
│   │   ├── molecules/  👈 (Combination of atoms: FormGroup)
│   │   ├── organisms/  👈 (Larger components: Navbar, Sidebar)
│   ├── templates/      👈 (Page layouts: Main Layout)
│   ├── pages/          👈 (Actual pages: Home, Login)
│   ├── shared/         👈 (Reusable utilities: Services, Pipes, Directives)
│   ├── styles/         👈 (Global styles: Colors, Fonts, Variables)
│   ├── app.config.ts   👈 (App configuration: Routing, Services)
│   ├── main.ts         👈 (Bootstrap application)
│   ├── app.component.ts👈 (Root component)
```

Coding Standards and Best Practices

- **Use meaningful component names** (user-profile, login-form).
- **Keep components small** and focused on one function.
- **Use TypeScript types** (string, number, boolean, interface).
- **Follow the folder structure** (Atomic Design).
- **Write comments** for complex logic.