**ANGULAR**

Web Applicartion framework for single page application built by Google.

**Benefits**

* Faster development.
* Faster code generation (CLI)

**Notes**

**CLI :**

Angular CLI (Command Line Interface) is a powerful tool for developing, building, and managing Angular applications efficiently. It simplifies common development tasks such as project setup, code scaffolding, testing, and deployment.

**ANGULAR VS REACT**

**ANGULAR**

* It is a framework.
* Has built in CLI.
* Has tools and packages included for small-medium scaled applications.

**REACT**

* Is a library.
* No built in CLI.
* Requires to install additional packages.

**Creating a new angular application**

* Install angular cli: npm install –g @angular/cli
* ng –version
* Create a new app: ng new first-app
* To start the application: npm start

**Notes**

* Inorder to create a new app without storing it on the hard disk, ng new first-app –dry-run
* To understand what all the available parameters are: ng new --help
* Inorder to create an app with a component and without additional files:  
  ng new first-app --inline-style --inline-template

**File structure**

* Package.json: main file to run any scripts
* Index.html: entry point for any web application. Top level component of the angular application (app-root) will be rendered as follows:

<body>

<app-root></app-root>

</body>

* Main.ts: create angular application using AppComponent. We are specifying angular to render AppComponent at the top level:

import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent, appConfig)

.catch((err) => console.error(err));

**Angular components**

An **Angular component** is the fundamental building block of an Angular application. It controls a section of the UI and consists of three main parts:

**1. Structure of an Angular Component**

A component typically includes:

* **Template (HTML)** – Defines the UI layout.
* **Class (TypeScript)** – Contains the logic and data.
* **Styles (CSS/SCSS)** – Defines the appearance.

**Creating a Component**

ng generate component my-component #it will create inside src/app

# or

ng g c my-component

# generate inside a specified path

ng g c components/header #it will create inside src/app/components/header

**Example**

import { Component } from '@angular/core';

@Component({

selector: 'app-hello', // Custom HTML tag <app-hello>

template: `<h1>Hello, {{ name }}!</h1>`, // Inline HTML template

styles: [`h1 { color: blue; }`] // Inline CSS

})

export class HelloComponent {

name: string = 'Angular';

}

**Explanation**

* **@Component({...})** is a **decorator**, which tells Angular that HelloComponent is a component.

**selector: 'app-hello'**

* Defines a custom HTML tag <app-hello>.
* This allows you to use <app-hello></app-hello> in other templates.

**template: <h1>Hello, {{ name }}!</h1>`**

* Defines the **HTML template** (UI structure) of the component.
* Uses **interpolation ({{ name }})** to dynamically display the name property from the component’s class.

**styles: [h1 { color: blue; }]**

* Defines **inline CSS** that applies only to this component.
* This makes <h1> text **blue**.
* **export** makes the class available to other parts of the app.
* **HelloComponent** is the **name of the component class**.

**Notes**

**SCSS** : **Sassy Cascading Style Sheets**.

* SCSS is a syntax of **Sass (Syntactically Awesome Stylesheets)**, which is a CSS preprocessor.
* It extends CSS with features like **variables, nesting, mixins, functions, and inheritance**.
* SCSS is more readable and maintains full **CSS compatibility**.

**Example of scss:**

$primary-color: blue;

h1 {

color: $primary-color;

font-size: 24px;

.subheading {

font-size: 18px;

color: darken($primary-color, 10%);

}

}

**After compilation:**

h1 { color: blue; font-size: 24px; }

h1 .subheading { font-size: 18px; color: #0000cc; /\* Darker blue \*/ }

**Angular Data-Binding**

Its the way we bind the data between typescript class of the component and component’s template. **Signals** are a new reactive state management feature in Angular, introduced to simplify **reactive programming** and **optimize performance**.

Types of Data Binding in Angular with Signals:

1. Interpolation ({{ }})
2. Property Binding ([property]="value" )
3. Event Binding ((event)="method()")
4. Two-Way Binding ([(ngModel)]="value")

**Example:**

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

@Component({

selector: 'app-example',

template: `

<h1>{{ name() }}</h1> <!-- Interpolation -->

<button (click)="updateName()">Change Name</button> <!-- Event Binding -->

`,

})

export class ExampleComponent {

name = signal('Angular'); // Signal holds reactive state

updateName() {

this.name.set('Updated Angular'); // Update signal value

}

}

**Explanation:**

* signal('Angular') creates a **reactive state**.
* name() (not name) is called in the template to get the current value.
* name.set('Updated Angular') updates the value, and the UI **automatically updates** without needing manual change detection.

**Notes:**

Inorder to accept some value from parent component, child component should have the following :

export class GreetingComponent {

message = input()

}

From parent component, we can pass as follows:

<app-greeting [message]="'Hello World '"/>

Since we are putting message inside [], it will expect a javascript. That could be either string as shown above or signal as shown below.

Or

<app-greeting [message]="message()"/>

the message() will come from :

export class HomeComponent {

message = signal("Hello World!");

}

**Event listeners in Angular**

In Angular, **event listeners** are used to handle user interactions like clicks, input changes, or keypresses.

Example:

<input type="text" (keyup)="keyUpHandler()"/>

export class HomeComponent {

homeMessage = signal("Hello World!");

keyUpHandler(){

console.log('user typing');

}

}

Inorder to track the event:

export class HomeComponent {

homeMessage = signal("Hello World!");

keyUpHandler(event: KeyboardEvent){

console.log(`user pressed ${event.key} key`);

}

}

<input type="text" (keyup)="keyUpHandler($event)"/>

**Notes:**

**Difference between ‘ ‘ and ``**:

### **Using Single Quotes (' ') or Double Quotes (" ")**

* Used for **plain strings**.
* The $ symbol is treated **literally**, not as a variable placeholder.

### **Using Backticks (` `) → Template Literals**

* Used for **string interpolation** (injecting variables).
* **Supports ${} syntax** to insert JavaScript expressions inside strings.

**Additional notes:**

To update value:

counterValue = signal(0);

increment(){

this.counterValue.update((val) => val + 1);

}

Inside template:

<button (click)="increment()">increment</button>

**Routing in Angular**

Angular is a single page application. By using routes, we can navigate to different pages.

The browser only loads the javascript or bundle that are related to the route user has accessed.

Example:

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

export const routes: Routes = [

{

path:'',

pathMatch:'full',

loadComponent: () => {

return import('./home/home.component').then((m) => m.HomeComponent)

},

},

];

Explanation:

* The path property specifies the **URL path**.
* Here, '' (empty string) means **this is the default or root path (/)**.

PathMatch

* Ensures **exact matching** of the path.
* full means the route will be activated **only when the full URL exactly matches '' (the root path).**
* If omitted, Angular may apply **partial matching**, leading to unexpected behaviors.

LoadComponent()

* **Lazy loads** a component **only when needed**.
* Instead of component: HomeComponent, it uses loadComponent, which is **dynamic**.
* Uses **dynamic import** to **lazy-load** the HomeComponent from './home/home.component'.
* The .then((m) => m.HomeComponent) ensures the module is loaded before accessing HomeComponent.

**Angular services**

* A **TypeScript class** that contains reusable **methods and logic**.
* Used for **data fetching, business logic, and state management**.

**Creating a new service:**

ng g service services/todo

Example:

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class TodosService {

constructor() { }

}

Explanation:

* **Injectable decorator** from Angular's core module.
* This decorator is required for **dependency injection**.
* It tells Angular that this class can be **injected into other components or services**.
* The @Injectable decorator **marks this class as an Angular service**.
* { providedIn: 'root' } makes this service a **singleton**, meaning:
* A **single instance** of TodosService is created **and shared** across the application.
* No need to manually add it in providers inside app.module.ts.

Injecting in a component:

export class TodosComponent {

todoService = inject(TodosService);

}

Additional notes:

For loop

@for (todo of todoItems(); track todo.id) {

<p>{{todo.title}}</p>

}

**Making Http calls with angular service**

Inside app.config.ts, provideHttpClient:

export const appConfig: ApplicationConfig = {

providers: [provideHttpClient(), provideZoneChangeDetection({ eventCoalescing: true }), provideRouter(routes)]

};

export class TodosService {

http = inject(HttpClient);

getTodosfromApi() {

const url = `https://jsonplaceholder.typicode.com/todos`

return this.http.get<Array<Todo>>(url);

}

}

export class TodosComponent implements OnInit{

todoService = inject(TodosService);

todoItems = signal<Array<Todo>>([]);

ngOnInit(): void {

this.todoService.getTodosfromApi().pipe(catchError((err) => {

console.log(err);

throw err;

})).subscribe((todos) => {

this.todoItems.set(todos)

})

}

}

* Declares the TodosComponent **class**.
* Implements OnInit, which is an **Angular lifecycle hook** that runs **after** the component has been initialized.
* **Purpose of OnInit**:
* It is used for **initialization tasks**, like **fetching data from an API**.
* Uses the .pipe() method to apply **RxJS operators** before subscribing to the response.
* **Catches errors** if the API request fails.
* **catchError** is an RxJS operator that:
* Handles errors **inside the observable stream**.
* Logs the error (console.log(err)) for debugging.
* **Throws the error again** so it can be handled elsewhere.
* **Subscribes** to the observable returned by getTodosfromApi().
* When data is received, the callback function (todos) => {} runs.

**Angular Directives**

To add additional behaviour to elements in Angular application.

To create a new directive:

ng g directive directives/highlight-completed-todo