AngularJS is a popular open-source framework that simplifies web development by creating interactive single-page applications (SPAs). Unlike traditional websites that load new pages for each click, SPAs offer a smoother user experience by updating content on the same page. AngularJS makes this possible by transforming static HTML into dynamic content that adapts to user interactions. Features like data binding and dependency injection streamline development, saving you time and effort. With regular updates and a large community, AngularJS ensures your web applications stay modern and efficient. AngularJS is a JavaScript Framework

MVC Framework The Model-View-Controller is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects. MVC Components: Following are the components of MVC: Model: The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data. View: The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with. Controller: Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.

Differences AngularJS & Angular:

Language:AngularJS: Uses JavaScript. Angular: Uses TypeScript, a superset of JavaScript that offers static typing and object-oriented features, leading to more robust and maintainable code.

Architecture: AngularJS: Employs the Model-View-Controller (MVC) or Model-View-ViewModel (MVVM) architectural pattern. Angular: Utilizes a component-based architecture, where applications are built from self-contained components.

Performance:Angular: Generally faster due to its optimized architecture, one-way data flow, and features like lazy loading and server-side rendering (Angular Universal). AngularJS: Can experience performance limitations, especially in larger applications, due to its two-way data binding and digest cycle. Data Binding: AngularJS: Primarily uses ng-model for two-way binding and ng-bind for one-way binding, relying on the ng directive approach. Angular: Uses []

for property binding and () for event binding, simplifying data flow.

In AngularJS, a component is a special type of directive that is used to create reusable UI elements with their own logic, template, and behavior.A component in AngularJS:

1 Is a simplified version of a directive. 2 Encapsulates template (HTML), controller (logic), and bindings (data inputs/outputs). 3 Promotes reusable, modular, and testable code. There is no fixed number of components you can create as many as your application needs. Each UI section or feature can be built as a separate component for modularity. Module Defines an application.Component Reusable, self-contained UI block. Controller Handles logic (used more in older AngularJS before components). Directive Extends HTML behavior. Service/Factory Provides reusable business logic or data. Filter Formats data for display. Routing Defines navigation between views.

The Angular CLI is a command-line interface tool which allows you to scaffold, develop, test, deploy, and maintain Angular applications directly from a command shell. Angular CLI is published on npm as the @angular/cli package and includes a binary named ng.

install Node and NPM on windows

Step 1: Download Node.js

1 Go to the official Node.js website: "https://nodejs.org". 2 Youll see two versions: LTS (Long Term Support) recommended for most users.Current includes the latest features but may be less stable. For AngularJS or Angular projects, choose LTS. 3 Click Windows Installer (.msi) to download.

Step 2: Install Node.js and npm

1 Run the installer you downloaded. 2 Follow the setup wizard:-> Accept the license agreement. -> Choose the default installation path. -> Leave the default options checked (especially the box that says Install npm package manager). -> Optional: Check Automatically install necessary tools this installs Python and Visual Studio Build Tools (useful for some npm packages). 3 Click Install and wait for it to finish. 4 When done, click Finish.

Step 3: Verify Installation

Open Command Prompt (cmd) or PowerShell, then run:node -v, Then check npm:npm -v

Step 4: Using Node for AngularJS

You dont need the Angular CLI. You just use Node and npm to manage dependencies or run a local server. Example: npm init -y, npm install http-server --save-dev, Then you can start a simple local server: npx http-server.

create your first Angular Project.

Step 1: Install Prerequisites

Youll need: A web browser (Chrome, Firefox, etc.). A text editor (VS Code, Sublime Text, etc.). Node.js and npm (optional) if you want to use a local web server.

Step 2: Create a Project Folder

Create a new folder for your project, for example: C:\Projects\MyFirstAngularJSApp

Step 3: Create Basic Files : index.html,app.js

Step 4: Add AngularJS Library

You can include AngularJS from a CDN.

In index.html, add:

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <meta charset="utf-8">
  <title>My First AngularJS App</title>
```

```
  <!-- Load AngularJS -->
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <script src="app.js"></script>
</head>
<body>
  <div ng-controller="myController">
    <h1>{{ message }}</h1>
  </div>
</body>
</html>
```

Step 5: Create the AngularJS Application

In app.js, define your AngularJS module and controller:

```
var app = angular.module("myApp", []);
app.controller("myController", function($scope) {
    $scope.message = "Hello, AngularJS!";
});
```

Step 6: Run the Application

Option 1 Open directly in browser:

Just double-click index.html AngularJS will run locally.

Option 2 Run via local server (recommended):

If you have Node.js, run a local server: npx http-server.

Step 7: Verify Output

Hello, AngularJS!

Features of Angular:

Build app-like experiences using modern web platform capabilities. High performance, offline, and zero-step installation. You can develop Cordova, Ionic, or NativeScript mobile apps with these strategies. With the addition of Angular to Mac, Windows, and Linux, you can create desktop-installed apps using the same methods as for the web. Furthermore, you can use native OS APIs on Windows and Linux. Due to Angulars advanced JavaScript virtual machine optimizations, hand-written code with the productivity of a framework is achieved without the problems of frameworks. The first view of your application can be served on Node.js, .NET, PHP, and other servers nearinstantaneously in HTML and CSS by also laying the groundwork for SEO-optimized sites. When using the new Component Router, which automatically splits the Angular code into code that is needed for rendering only the part of the page that is requested, users only need to load code necessary for the page to be displayed. UI views can be created quickly with template syntax that issimple and powerful. Quickly build modules and tests, then instantly deploy them using Command Line Tools. In addition to intelligent code completion, instant errors, and other feedback, popular editors and IDEs offer this capability. You can track whether or not youve violated any rules by checking Karma for unit tests. An Angular tutorial using very little code to build complex choreography and animation timelines. Use ARIA-compliant components, developer guides, and built-in a11y test infrastructure to create

accessible applications

Creating Project You develop appsin the context of an Angular workspace. To create a new workspace and initial starter app: 1. Run the CLI command ng new and provide the name my-app, as shown here: ng new my-app 2. The ng new command prompts you for information about featuresto include in the initial app. Accept the defaults by pressing the Enter or Return key. The Angular CLI installsthe necessary Angular npm packages and other dependencies. This can take a few minutes.The CLI creates a new workspace and a simple Welcome app, ready to run.

Running Project The Angular CLI includes a server, for you to build and serve your app locally. 1. Navigate to the workspace folder,such as my-app. 2. Run the following command: cd my-app ng serve --open

Files used in Angular App folder: Angular App files which are mainly used in your project are given below: src folder: This is the folder which contains the main code filesrelated to your angular application. app folder: The app folder contains the files, you have created for app components. app.component.css: This file containsthe cascading style sheets code for your app component. app.component.html: This file contains the html file related to app component. This is the template file which is used by angular to do the data binding. app.component.spec.ts: This file is a unit testing file related to app component. This file is used along with other unit tests. It is run from Angular CLI by the command ng test. app.component.ts: This is the most important typescript file which includes the view logic behind the component. app.module.ts: This is also a typescript file which includes all the dependencies for the website. This file is used to define the needed modules to be imported, the components to be declared and the main component to be bootstrapped.

1

1

1

1

1

1

1

1

1

1

1

Data Binding: Data binding is the core concept of Angular and used to define the communication between a component and the DOM. It is a technique to link your data to your view layer. In simple words, you can say that data binding is a communication between your typescript code of your component and your template which user sees. It makes easy to define interactive applications without worrying about pushing and pulling data

One-way databinding: One way databinding is a simple one-way communication where HTML template is changed when we make changes in TypeScript code. Or in one-way databinding, the value of the Model is used in the View (HTML page) but you can't update Model from the View. Angular Interpolation / String Interpolation, Property Binding, and Event Binding are the example of one-way databinding.

Two-way databinding: In two-way databinding, automatic synchronization of data happens between the Model and the View. Here, change is reflected in both components. Whenever you make changes in the Model, it will be reflected in the View and when you make changes in View, it will be reflected in Model. This happens immediately and automatically, ensures that the HTML template and the TypeScript code are updated at all times. In two- way data binding, property binding and event binding are combined together. Syntax: [(ngModel)] = "[property of your component]"

Types of Binding: String interpolation: String Interpolation is a one-way databinding technique which is used to output the data from a TypeScript code to HTML template (view). It uses the template expression in double curly braces to display the data from the component to the view. For example: {{ data }} String interpolation adds the value of a property from the component. Syntax:

 Name: {{ user.name }}

 Email: {{ user.email }}

Property Binding: Property Binding is also a one-way data binding technique. In property binding, we bind a property of a DOM element to a field which is a defined property in our component TypeScript code. For example: Syntax: <input type="email" [value]="user.email">

Event Binding: In Angular, event binding is used to handle the events raised from the DOM like button click, mouse move etc. When the DOM event happens (eg. click, change, keyup), it calls the

specified method in the component. In the following example, the cookBacon() method from the component is called when the button is clicked. For example: Two-way Data Binding:

Directives The Angular directives are used to manipulate the DOM. By using Angular directives, you can change the appearance, behavior or a layout of a DOM element. It also helps you to extend HTML. Angular directives are DOM elements to interact with your application. Angular directives begin with ngwhere ng stands for Angular and extends HTML tags with @directive decorator. Angular directives can be classified in three categories Component Directives: Component can be used as directives. Every component hasInput and Output option

to pass between component and its parent HTML elements.

<component-selector-name[input-reference]="input-value"> ... </component-selector-name>

EX: <list- item [items]="fruits"> ... </list- item>

Structural Directives: Used to add or remove DOM elements in the current HTML document. <HTMLTag [structural directive]= 'value />. Ex, <div *ngIf="isNeeded"> Only render if the *isNeeded* value has true value.

Structural directives start with a * sign.

*ngIf Directive: The ngIf allows usto Add/Remove DOM Element. *ngSwitch Directive: The *ngSwitch allows us to Add/Remove DOM Element. It is similar to switch statement of C#. *ngFor Directive: The *ngFor directive is used to repeat a portion of HTML template once per each item from an iterable list (Collection)

Attribute Directives: Used to add new attributes forthe existing HTML elements to change itslook and behaviour. <HTMLTag [attrDirective]='value />.

EX: <p [showToolTip]='Tips' />

Attribute directives are used to change the look and behavior of the DOM elements.

ngClass Directive: The ngClass directive is used to add or remove CSS classesto an HTML element. ngStyle Directive: The ngStyle directive facilitates you to modify the style of an HTML

element using the expression. You can also use ngStyle directive to dynamically change the style of your HTML element.

Creating multiple modules: Modules are a great way to organize an application and extend it with capabilities from external libraries.

Angular libraries are NgModules, such as FormsModule, HttpClientModule, and RouterModule. Many thirdparty libraries are available as NgModules such as Material Design, Ionic, and AngularFire2.

Step 1: Create the Main Module. var mainApp = angular.module('mainApp', ['moduleA', 'moduleB']);mainApp is the main (root) module. It depends on two other modules: moduleA and module.

Step 2: Create the First Module. angular.module('moduleA', []).controller('controllerA', function($scope) { $scope.messageA = "Hello from Module A!"; });

Step 3: Create the Second Module. angular.module('moduleB', []).controller('controllerB', function($scope) { $scope.messageB = "Hello from Module B!"; });

Step 4: Include All Script Files in HTML

```
<!DOCTYPE html>
<html ng-app="mainApp">
<head> <script src= "https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <script src="moduleA.js"></script>
  <script src="moduleB.js"></script>
  <script src="mainApp.js"></script> </head>
<body>
  <div ng-controller="controllerA">
    {{ messageA }}
  </div>
  <div ng-controller="controllerB">
```

{{ messageB }}

</body> </html>

ng-model for two way Data Binding process

The ng-model directive in AngularJS is used to bind the value of an HTML form element (like <input>, <select>, or <textarea>) to a variable defined in the AngularJS scope ($scope).

It establishes a two-way data binding meaning: -When the user changes the value in the view (HTML input), it automatically updates the model (JavaScript variable). -When the model changes in the controller, it automatically updates the view.

How Two-Way Data Binding Works.. 1.The ng-model directive connects the view and the model. 2.AngularJS constantly monitors both the model and the view for changes. 3.Any change in one is instantly reflected in the other hence two-way binding.

Explanation: ng-model="name" binds the input field to $scope.name. When you type in the input box, the value of $scope.name updates automatically. The change in $scope.name is immediately reflected in {{ name }} in the view.

Benefits of ng-model: 1.Enables real-time synchronization between view and model. 2.Reduces boilerplate code for event handling. 3.Makes form inputs and validation simpler.

Ex:- <!DOCTYPE html>

<html ng-app="myApp">

<head>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body ng-controller="myCtrl">

<h2>Two-Way Data Binding Example</h2>

<input type="text" ng-model="name" placeholder="Enter your name">

Hello, {{ name }}!

<script>

```
    angular.module('myApp', []).controller('myCtrl', function($scope) {$scope.name = "Alice";  // initial model value});
  </script>
</body>
</html>
```

Formmodule and inputs & variables to components.

FormsModule is an Angular module that provides tools and directives to build and manage template-driven forms. It allows you to: Use two-way data binding with [(ngModel)]. Capture and validate user inputs. Work easily with form controls like <input>, <select>, and <textarea>.

Step 1: Import FormsModule:- Before using form features, you must import FormsModule in your app module.

```
import { NgModule } from '@angular/core'; import { BrowserModule } from '@angular/platform-browser'; import { FormsModule } from '@angular/forms'; import { AppComponent } from './app.component';
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, FormsModule],
  bootstrap: [AppComponent] })
export class AppModule { }
```

Step 2: Use [(ngModel)] for Two-Way Data Binding

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html' })
export class AppComponent { username: string = ''; }
<h2>Template-driven Form Example</h2>
<input type="text" [(ngModel)]="username" placeholder="Enter your name">
```

Hello, {{ username }}!

How it works: [(ngModel)] binds the <input> value to the username variable. Changes in the input update the variable, and vice versa two-way binding.

Passing Inputs and Variables to Components: Angular components communicate through @Input() and @Output() decorators.

Step 1: Define a Child Component

```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'app-child',
  template: `<p>Message from parent: {{ message }}</p>` })
export class ChildComponent {
  @Input() message!: string; // variable received from parent }
```

Step 2: Use the Child Component in the Parent Template

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-parent',
  template: `
    <h2>Parent Component</h2>
    <input [(ngModel)]="parentMessage" placeholder="Type a message">
        <app-child  [message]="parentMessage"></app-child>  `  }) export  class  ParentComponent  {
parentMessage: string = 'Hello Child!'; }
```

ngif

The ngIf Directives is used to add or remove HTML Elements according to the expression. The expression must return a Boolean value. If the expression is false then the element is removed, otherwise element isinserted. It is similar to the ng-if directive of AngularJS. ngIf Syntax: <p *ngIf="condition">condition is true and ngIf is true.

<p *ngIf="!condition"> Condition is false and ngIf is false

The *ngIf directive form with an "else" block

ngFor

The *ngFor directive is used to repeat a portion of HTML template once per each item from an iterable list (Collection). The ngFor is an Angular structural directive and is similar to ngRepeat in AngularJS. Some local variables like Index, First, Last, odd and even are exported by *ngFor directive. See the simplified syntax for the ngFor directive: <li *ngFor="let item of items;">....

Ex: In .ts export class AppComponent { Fruits[]=[{Id:1 ,Name:'Apple'},{Id:1 ,Name:'Mango'},{Id:1 ,Name:'Banana'}]

In Html <div *ngFor="let fruit of Fruits;"> <h1>{{fruit.Name}}</h1>

2

2

2

2

2

2

2

2

2

2

Dependency Injection (DI): When you develop a smaller part of yoursystem, like a module or a class, you may need to use features from other classes. For example, you may need an HTTP service to make backend calls. Dependency Injection.Dependency injection, or DI, is one of the fundamental concepts in Angular. DI is wired into the Angular framework and allows classes with Angular decorators, such as Components, Directives, Pipes, and Injectables, to configure dependencies that they need.Dependency injection (DI) is the part of the Angular framework that provides components with access to services and other resources. The @Injectable() decorator defines a class as a service in Angular and allows Angular to inject it into a component as a dependency.

Services: Services providesspecific functionality in an Angular application. In a given Angular application, there may be one or more services can be used. Similarly, an Angular component may depend on one or more services. Service is a broad category encompassing any value, function, or feature that an application needs. A service istypically a class with a narrow, well-defined purpose. Itshould do something specific and do it well.

Use Of Service

1.Code Reusability: Write once, use everywhere. 2.Separation of Concerns: Keeps controllers focused on presentation logic. 3.Singletons: One shared instance ideal for shared state or caching. 4.Easy Testing: Mock or stub services in unit tests.

Creating a Service: Basically, Service helps to organize & share the business logic, data model & functions with various components in the application, & it gets instantiated only once in the lifecycle of the application. For this, Services is written only once & can be injected into the different components, which use that particular Services. Services and other dependencies are injected directly into the constructor of the component like this: constructor(private _myService: MyService) { }

By doing this, we are actually creating an instance of the service, which means we have to access all the public variables and methods of the service. To create a new service, we can use the below command: ng generate service my-custom-service

import { Injectable } from '@angular/core'; @Injectable() export class DebugService { constructor() { } }

Injecting the service into components is pretty straightforward. For instance,suppose we have a service called

MyCustomService. This is how we can inject it into a component: MyCustomComponent.ts

```
import {... } from "@angular/core";
import { MyCustomService } from "../...PATH";
@Component({
selector: "...",
templateUrl: "...",
styleUrls: ["..."],
})
export class MyCustomComponent {
// INJECTING SERVICE INTO THE CONSTRUCTOR
constructor(private _myCustomService:
MyCustomService) { }
// USING THE SERVICE MEMBERS
this._myCustomService.sampleMethod();
```

Injecting a Service into otherservices: Lets say that you have a Service1 and Service2 in an regular Angular application nothing fancy at all. Lets say now that this Service2 depends on Service1. The first thing is to add the @Injectable decorator on each services you want to inject. In fact, the Injectable name is a bit insidious. It doesn't mean that the class will be "injectable" but it will decorate so the constructor parameters can be injected. .In fact, the Injectable name is a bit insidious. It doesn't mean that the class will be "injectable" but it will decorate so the constructor parameters can be injected. When setting an @Injectable decorator for a class, Angular will try to create or get instances for corresponding types in the injector for the current execution chain. In fact, there is not only one injector for an Angular2 application but a tree of injectors

Dependency Hierarchical Injector

Dependency Injection (DI) is one of the core features it helps manage how components (like controllers, services, directives, filters, etc.) get the objects they depend on. But when people talk about Dependency Hierarchical Injector, theyre referring to how AngularJS structures its injectors in a hierarchical or nested manner rather than having just one global injector. A dependency injector is the system that creates and delivers the services or objects a component needs. EX: app.controller('MyController', function($scope, myService) { // myService is injected automatically});

How the Hierarchy Works:

```
  <div ng-controller="ParentCtrl">
    <div ng-controller="ChildCtrl"></div>
  </div>
</div>
```

The root injector is created by ng-app. When AngularJS compiles the DOM, it creates child injectors for nested components like controllers or directives. Each child injector can:->Use services from the parent (inherited),->Or define its own services (which override parent ones).

Hierarchical Injector is A tree-like structure of injectors each component can have its own injector inheriting from its parent.

Rest Calls with HttpClient

Http client programming is a must needed feature in every modern web application. Nowadays, lot of application exposes their functionality through REST API (functionality over HTTP protocol). With this in mind, Angular Team provides extensive support to access HTTP server. Angular provides a separate module, HttpClientModule and a service, HttpClient to do HTTP programming. The prerequisite to do Http programming is the basic knowledge of Http protocol and REST API technique. Http programming involves two part, server and client. Angular provides support to create client side application. Express a popular web framework provides support to create serverside application.

Lets go step by step so you see the complete setup...1. Import the HttpClientModule 2. Inject

HttpClient into a Service 3. Use the Service in a Component 4. Adding Headers or Parameters 5. Handling Errors with RxJS catchError

Summary

Import module: HttpClientModule in app.module.ts

Inject service: constructor(private http: HttpClient) {}

GET request: this.http.get(url)

POST request: this.http.post(url, data)

PUT request: this.http.put(url, data)

DELETE request: this.http.delete(url)

Add headers/params:{ headers, params} options

Handle errors: .pipe(catchError(...))

$inject in AngularJS (v1.x) is a key concept related to Dependency Injection (DI) and how Angular knows which dependencies to inject into your components (controllers, services, directives, etc.). $inject is a static property you attach to a function (like a controller or service constructor) that explicitly lists the names of the dependencies to be injected. It helps AngularJS understand which services or objects to pass into your function especially when your code gets minified.

How $inject Works

Its an array of strings that lists dependency names in the same order as the function parameters. AngularJS reads $inject before calling your function and injects the matching services.

3

3

3

3

3

3

3

Router: Navigation is one of the important aspect in a web application. Even though a single page application (SPA) does not have multiple page concept, it does moves from one view (list of expenses) to another view (expense details). Providing clear and understandable navigation elements decides the success of an application. Angular provides extensive set of navigation feature to accommodate simple scenario to complex scenario. The process of defining navigation element and the corresponding view is called Routing. Angular provides a separate module, RouterModule to set up the navigation in the Angular application.

keys/Components of Router:

1.Router: Displays the application component for the active URL. Manages navigation from one component to the next. 2.RouterModule: A separate NgModule that provides the necessary service providers and directives for navigating through application views. 3.Routes: Defines an array of Routes, each mapping a URL path to a component. 4.Route: Defines how the routershould navigate to a component based on a URL pattern. Most routes consist of a path and a component type. 5.RouterOutlet: The directive (<router-outlet>) that marks where the router displays a view. 6.RouterLink: The directive for binding a clickable HTML element to a route. Clicking an element with a routerLink directive that's bound to a string or a link parameters array triggers a navigation. 7.RouterLinkActive:The directive for adding/removing classes from an HTML element when an associated routerLink contained on or inside the element becomes active/inactive. It can also set the aria-current of an active link for better accessibility. 8.ActivatedRoute: A service that's provided to each route component that contains route specific information such as route parameters, static data, resolve data, global query parameters, and the global fragment. 9.RouterState: The current state of the router including a tree of the currently activated routes together with convenience methods for traversing the route tree. 10.Link parameters array: An array that the router interprets as a routing instruction. You can bind that array to a RouterLink or pass the array as an argument to the Router.navigate method. 11.Routing component: An Angular component with a RouterOutlet that displays views based on router navigations.

Router.navigate

In Angular, both Router.navigate() method of the Router service used for programmatic navigation (navigating via code instead of template links).Used to navigate using an array of route commands (relative or absolute). This method is path-segment aware, meaning it works with route parameters and relative paths easily. Syntax:this.router.navigate(commands: any[], extras?: NavigationExtras).. Parameters: commands - An array of route segments. extras (optional) - An object for navigation options (e.g. queryParams, fragment, relativeTo, etc.). Example: Absolute Navigation -> this.router.navigate(['/home']); Navigation with Route Parameters -> this.router.navigate(['/product', productId]); Relative Navigation -> this.router.navigate(['details'], { relativeTo: this.route });

Router.navigateByUrl()

In Angular, both Router.navigateByUrl() method of the Router service used for programmatic navigation (navigating via code instead of template links). Used to navigate directly using a full URL string. It does not split the URL into segmentsit just treats the provided string as a full path. Syntax: this.router.navigateByUrl(url: string | UrlTree, extras?: NavigationBehaviorOptions).. Example: Simple Absolute URL-> this.router.navigateByUrl('/home');

URL with Query Parameters-> this.router.navigateByUrl('/search?category=books&sort=asc'); Using a UrlTree-> const urlTree = this.router.createUrlTree(['/product', 123]); this.router.navigateByUrl(urlTree);

Differences Router.navigate()/Router.navigateByUrl()

Feature

(Input type) 1.Array of route commands, 2.String or UrlTree. (Relative navigation) 1.Supports relativeTo, 2.Always absolute. (Query params / extras) 1.Easier with NavigationExtras, 2.Possible but less flexible

(Use case) 1.When building routes dynamically or relatively, 2.When you already know the full URL

(Example) 1.this.router.navigate(['../login'], { relativeTo: this.route }), 2.this.router.navigateByUrl('/login')

$routeProvider is used to configure routes in a Single Page Application (SPA). It defines which template (HTML view) and controller should load for a specific URL. Configured inside the .config()

block of a module.

Syntax app.config(function($routeProvider) {

$routeProvider.when('/path', {

templateUrl: 'template.html',

controller: 'ControllerName }).otherwise({

redirectTo: '/defaultPath}); });

In short $routeProvider defines routes (URL template + controller)

$routeParams is used to access route parameters (values passed in the URL). Used inside a controller to read dynamic parts of the URL. Example: // Route $routeProvider.when('/user/:id', { templateUrl: 'user.html', controller: 'UserCtrl' });

// Controller app.controller('UserCtrl', function($scope, $routeParams) { $scope.userId = $routeParams.id; }); In short $routeParams retrieves values from the route URL (like /user/:id)

A wildcard route is used to catch all undefined or unmatched routes and usually redirect the user to a Not Found page or a default route. It works like a safety net for your routing configuration. Syntax

const routes: Routes = [

{ path: '', component: HomeComponent },

{ path: 'about', component: AboutComponent },

{ path: '**', component: PageNotFoundComponent } // wildcard route];

Key Points

1.path:'**' ->The double asterisk ** matches any URL that hasnt been matched by previous routes. ->Always put the wildcard route last in the routes array because Angular checks routes in order. 2.Purpose : ->Show a 404 Not Found page. ->Redirect to a default page if the URL is invalid.

Summary:

(**)->Wildcard path that matches all unmatched routes. (component)->Route to display (like PageNotFoundComponent). (redirectTo)->Optional: redirect unmatched routes. (Order)->Must always be the last route in the array.

NgRoute Module

ngRoute is an AngularJS module used to implement routing in Single Page Applications (SPAs). It allows you to navigate between different views (pages) within a single HTML page without reloading the whole page.

Purpose: To switch views (HTML templates) based on the URL. To make web applications faster and more dynamic. To improve user experience by avoiding full-page reloads.

How Routing Works:

1.The app uses $routeProvider to define routes.

2.Each route is linked to:

A URL path

A template (HTML file)

A controller (JavaScript logic)

3.The ng-view directive is used in HTML to display the view dynamically.

RouterModule.forRoot()

Definition: RouterModule.forRoot() is used once in the root module (AppModule) to set up the main application routes.

Purpose: It initializes the Angular router. It creates and configures the router service, navigation guards, and directives that Angular needs to manage routes. Used for top-level routes.

EX: @NgModule({ imports: [RouterModule.forRoot(routes)],exports: [RouterModule]})

forRoot() Used once at the root of the app to set up global routes.

Difference RouterModule.forRoot()/ RouterModule.forChild()

Feature

(Usage Location) 1.Used in the root module (AppModule) 2.Used in feature modules (Purpose) 1.Sets up the main router and its configuration 2.Adds child or feature-specific routes (Router Service Creation) 1.Creates a new router service instance 2.Does not create a new router service

(Number of Times Used) 1.Only once in the whole app 2.Can be used multiple times in feature modules

(For Lazy Loading) 1.Not used 2.Commonly used for lazy-loaded modules (Example Module)

1.AppRoutingModule 2.UserRoutingModule, AdminRoutingModule, etc.

(Imports Needed) 1.RouterModule.forRoot(routes) 2.RouterModule.forChild(routes)

RouterModule.forChild()

Definition: RouterModule.forChild() is used in feature modules to configure child routes or module-specific routes.

Purpose: Used to add additional routes to the router configuration. It does not re-initialize the router service. Used in lazy-loaded or feature modules.

Ex: @NgModule({ imports [RouterModule.forChild(studentRoutes)],exports: [RouterModule] })

forChild() Used in feature modules to define their own routing without reconfiguring the main router.

4

4

4

4

4

4

4

4

4

4

4

4

4

Form handling in Angular is a core part of building interactive web applications. Forms are used to handle user input data. Angular supports two types of forms. They are Template driven forms and Reactive forms. Template driven forms: Template driven forms is created using directives in the template. It is mainly used for creating a simple form application. Reactive Forms: Reactive Forms is created inside component class so it is also referred as model driven forms. Every form control will have an object in the component and this provides greater control and flexibility in the form programming. Reactive Form is based on structured data model.

template driven forms: Before understanding forms, let us learn how to configure forms in an application. To enable template driven forms, first we need to import FormsModule in app.module.ts. After the import FormsModule, the application will be ready for form programming. Now Configure FormsModule in our Component. After that we used ngModel attribute in input text field.

Key features: Uses ngModel for two-way data binding. Best for simple forms. Automatically tracks form and control states (valid, dirty, etc.)

Reactive forms: To enable reactive forms, first we need to import ReactiveFormsModule in app.module.ts.

Key features: Form logic is handled entirely in the component class. Easier to test and scale for complex forms. Offers powerful APIs for validation, dynamic form controls, and observables

Before moving to create Reactive forms, we need to understand about the following concepts,

- FormControl Define basic functionality of individual form control
- FormGroup Used to aggregate the values of collection form control
- FormArray Used to aggregate the values of form control into an array
- ControlValueAccessor Acts as an interface between Forms API to HTML DOM elements.

ng-submit in AngularJS

The ng-submit directive is used to handle form submission in AngularJS. It executes a specified function in the controller when the user submits the form (by clicking submit or pressing Enter). It helps prevent the default page reload of HTML forms and allows AngularJS to handle the logic instead.

Syntax:<form ng-submit="submitForm()">

<button type="submit">Submit</button>

</form>

When the user submits the form, the function submitForm() is called. This allows you to process or validate form data using AngularJS without reloading the page.

$untouched

True if the user has not focused (entered) the input field yet. Use: To check if the field has never been interacted with.

Example:<input name="username" ng-model="user" required> Field not touched yet!

$touched

True when the user has focused and left the input field. Use: To show validation errors only after the user interacts with the field.

Example:<input name="email" ng-model="userEmail" required> Invalid email!

$pristine

True if the input field has not been modified by the user. Use: To detect untouched/unchanged fields.

Example: <input name="age" ng-model="userAge"> You haven't changed this field yet.

Built-in Validators in Angular Forms

Angular provides several built-in validators to perform common form validations automatically such as checking if a field is required, an email, or within a range.

required: Ensures the field is not empty.

Ex: <input name="name" ngModel required>, name: new FormControl('', Validators.required)

minlength: Requires the input to have a minimum number of characters.

Ex:<input name="pwd" ngModel minlength="5">, pwd: new FormControl('', Validators.minLength(5))

maxlength: Ensures the input has a maximum number of characters.

Ex: <input name="pwd" ngModel maxlength="10">, pwd: new FormControl('', Validators.maxLength(10))

pattern: Checks if the input matches a regular expression.

Ex: <input name="phone" ngModel pattern="[0-9]{10}">, phone: new FormControl('', Validators.pattern('[0-9]{10}'))

email: Validates if the input is a valid email address.

Ex: <input type="email" name="email" ngModel email>, email: new FormControl('', Validators.email)

min: Ensures numeric input is greater than or equal to a value.

Ex: <input type="number" name="age" ngModel min="18">, age: new FormControl('', Validators.min(18))

max: Ensures numeric input is less than or equal to a value.

Ex: <input type="number" name="age" ngModel max="60">, age: new FormControl('', Validators.max(60))

Mouse Event

ng-click Executes a function when the user clicks an element; commonly used for buttons, links, and triggering custom actions or event handlers. Syntax: <button ng-click="showMsg()">Click</button>

ng-dblclick Invoked when an element is double-clicked by the user, useful for toggling views, editing fields, or performing repeated quick actions. Syntax: Double Click

ng-mousedown Triggered when the mouse button is pressed down on an element, often used to start drag operations or visual effects. Syntax: Press Down

ng-mouseup Fires when the mouse button is released after being pressed, commonly paired with ng-mousedown to complete mouse actions. Syntax: Release

ng-mouseenter Activated when the mouse pointer enters an elements boundary, often used for hover effects or showing tooltips. Syntax: Hover In

ng-mouseleave Triggered when the mouse pointer exits an elements area, typically used to hide

tooltips or reverse hover effects. Syntax: Hover Out

ng-mousemove Continuously fires while the mouse pointer moves over an element, useful for tracking movement or implementing drag-and-drop. Syntax: Move Mouse

ng-mouseover Similar to ng-mouseenter, but also triggers when entering child elements; often used for highlighting nested content. Syntax: Mouse Over

Form validation is an important part of web application. It is used to validate whether the user input is in

correct format or not. The following can be used to track error.

$dirty True when the user changes the input field value. statesthat value has been changed.

Ex: <input name="user" ng-model="uName" required>

Field modified!

$invalid Becomes true if the input field fails validation (e.g., empty or wrong format). statesthat value entered is invalid.

Ex: <input type="email" name="email" ng-model="uEmail" required>

Invalid email!

$error Holds specific error details like required, email, or pattern. statesthe exact error.

Ex: <input name="age" ng-model="uAge" required>

Age is required!

Validation Diretive

ng-minlength: The ng-minlength Directive in Angular is used to set the minimum length for an input field i.e it

adds the restriction for an input field. It is different from minlength attribute in HTML because the former

prevents users from entering less than the specified limit whereas the later doesnt do that. It makes the form

invalid if the entered input is less than the specified limit.syntax:

<element ng-minlength="expressiOn"> cOntent...

</element>

ng-maxlength: The ng-maxlength Directive in Angular is used to set the maximum length for an input field i.e

it adds the restriction for an input field. It is different from maxlength attribute in HTML because the former

prevents users from exceeding the limit whereas the later doesnt do that. It will make the form invalid if the

input limit exceeds it. syntax:

<element ng-maxlength="expressiOn">

COntents...

</element>

ng-pattern: The ng-pattern Directive in AngularJS is used to add up pattern (regex pattern) validator to ngModel on an input HTML element. It is used to set the pattern validation error key if input field data does not

match a RegExp that is found by evaluating the Angular expression specified in the ng-pattern attribute value. syntax:

<element ng-pattern="expressiOn">

COntents...

</element>

ng-required: The ng-required Directive in AngularJS is used to specify the required attribute of an HTML

element. The input field in the form is required only if the expression inside the ng-required directive returns

true. It is supported by <input>, <select> and <textarea> tags. syntax:

<element ng-required="expressiOn">

COntents...

</element>

5

5

5

5

5

5

5

5

5