**ANGULAR**

🡪Angular: Angular is a framework to build single page applications.

Framework: It is combination of languages and libraries.

🡪1.HTML

🡪2.CSS

🡪3.Typescript

🡪4.Rxjs

🡪5.Zone.js

**SPA:** (Single Page Application)

Only one HTML page will be there, port of the page will change WRT (with respect to) user activity.

**Project setup:**

1). Download and install node.js

To check:

node -v

npm -v

2). To Install angular

npm install -g @angular/cli

check: ng –version

3. To create the new project in angular:

ng new filename

4). To run the server

ng serve

**Angular topics:**



CLI



(Routing) (HTTP) (RXjs) (Forms)



Architecture



**Routing:** Angular is a SPA.

Routing helps us to navigate from one view to another view.

🡪There are 4 types of routings in angular.

1. **General Route:**

{path:’login’, component: login component}

2. **Child route:**

{path: ‘dashboard’, component: DashboardComponent,

Children: [{path:’bmi’, component: BMIComponent}

]}

**3.Empty Route:**

{path:’ ’, component: loginComponent}

**4. Wild card Route:**

{path:’\*\*’, component: PagenotfoundComponent}

**Routing Implementation:**

1. Create route in app-routing.module.ts files routes array.

2. Mention <router-outlet> tag in app.component.html file.

**Navigate between views:**

1.From HTML

<button routerlink = “/dashboard”>  
 <<button routerlink = “calculator”>

4200/abc

/xyz xyz



4200/xyz 4200/abc/xyz



**Data Binding:**

🡪There are 4 types of Data Bindings

1.**Interpolation:** Ts 🡪 HTML

To display variable values in template.

<p>{{age}}</p>

2.**Property Binding:** Ts 🡪 HTML

To bind the variable value to html element.

<button[disabled]=” true”>

3. **Event Binding:** HTML 🡪 Ts

To capture user events in template.

<button (click) = “submit()”>submit</button>

4**.Two-way Binding:** Ts 🡨🡪 HTML

To exchange the data in two ways.

<input [(ngModel)]= “age”>

🡪To implement two way binding we need to use FormsModule in app.module.ts

**Variable Structure:**

**Public** age: number = 20;

Public🡪 Access Specifier

age 🡪name

number🡪Data type

20 🡪 value

**Variable names rules:**

1. It should start with alphabet

Ex: 1name 🡪 wrong declaration

name 🡪 Correct declaration

2. Cannot use space in name

3. Should use camelCase names

firstname 🡪 wrong declaration

first-name 🡪 wrong declaration

first\_name🡪 wrong declaration

FirstName🡪Correct declaration

4.Boolean names should start with “is”

isValid, isMale.

5. Array names should always in plural

Users, names

6. Name should be descriptive

fn🡪 wrong declaration

FirstName🡪Correct declaration

**Directives:** Useful to alter the DOM (Document object module)

🡪There are 3 types of Directives

1.**Structural Directives:** Useful to alter the DOM structure

a. ngFor :- To iteratively add element to DOM.

<tr \*ngFor = “let user of users”>

b. ngIf :- To conditionally show and hide the elements in DOM.

<p \*ngIf =”true”>

2.**Attribute Directives:-** Useful to alter attribute value dynamically.

**a). ngstyle:**

<td [ngstyle]=” {‘color’: isvalid? ‘green’ :’ red’}”>

**b). ngClass:**

**<**td [ngclass]= “{‘red’:marks<35, ‘green’: marks>35}”>

**3. Component Directive:**

🡪 In angular, every component is directive. Directives are superset, components are subset.

**Pipes:** Useful to transform data from one format to another format.

**Ex:** {{user.name | uppercase}}

{{user.name | lowercase}}

{{product | Json}}

{{today | date: ‘yyyy-MM-dd}}

**API integration:**

**🡪**Data flows to API through Forms.

**Html🡪🡪🡪🡪Ts🡪🡪🡪🡪servive🡪🡪🡪🡪API**

🡪Data comes from API to browser through table

**HTML🡨🡨🡨🡨Ts🡨🡨🡨🡨service🡨🡨🡨🡨API**

**RxJs:-**Responsible for all asynchronous operations in Angular.

🡪It has 3 features:

1.Promise

2.Observable

3.Subject

**Promise Observable**

1.Single callback. 1.Multiple callbacks.

2.Cannot stop in between. 2.We can stop n between using

Unsubscribe method.

3.Supports less Rxjs operator. 3.Support more Rxjs operators.

**Observable Subject**

1.Unicasting. 1.Multicasing/Broadcasting.

2.State (Not available). 2.State (available).

**API Integration Steps:**

**1.**Import HTTPClientModule in app.module.ts

**2.** Inject HttpClientService in Userservice(which is created for component).

🡪constructor (private \_httpClient(variable):HttpClient){}

**3.** Create **getusers** methods to load data

🡪getusers(): observable<any>{

Return this.\_httpClient.get(“URL”);

}

**4.** Inject UserService in UserComponent.ts(which is created).

**5.** Subscribe to the observable and store data.

🡪 Public users (variable to store the data received from API).

🡪Constructor (private \_userservice(variable):UserService){

\_userservice.**getusers().**subscribe(

(data:any)=>{this.users =data},

(error:any)=>{}) } 🡪for error message.

**ALL Possible API Calls**

**Data API Method API-Call**

1.AllData get get(url)

2.Specific Data get get(url/id)

3.Filter get get(url?filter=\*\*\*\*\*\*)

4.Pagination get get(url?limit=10&page=1)

5.Sorting get get(url?sortBy=”column

Name&order=asc/desc)

6.Create post post(url/,data)

7.Update put put(url/id,data)

**Forms:**

🡪There are two types of forms in Angular

1.Template Driven form (ngModel)

2.Reactive form.

**Reactive Forms:**

1.Import ReactiveformsModule.

2.Create FormGroup in class

3.create form in Html

4.Map the controls

5.Check the form output in controls.

**Reactive Forms features**

1.FormGroup.

2.NestedFormGroup.

3.FormArray.

4.Dynamic form

5.Form Validations.

**Form Validators:**

**TS:**

Age: new formcontrol[null, [validators.required]]

**HTML:**

**<div** \*ngIf=”control?.touched || control?.dirty”**>**

<p \*ngIf=”control?.errors.[‘required’]”>error\_message</p>

**</div>**

**Controls:**

**FormGroup:-** Userform.get(‘age’)

**NestedFormgroup:** userform.get(‘address’).get(‘pincode’)

**FormArray:-** userformArray.control[i]?.get(‘cvv’)

**Communication in Angular**

1.Page Communication.

2.Component Communication.

3.Module Communication

**2.Component Communication: -**

**Parent to Child:**

**Parent.html**

**<app-child** [ac]=”10” (bEvent)=”cathch($event)”**></app-chlid>**

**Child.ts**

@input() public ac:string= “ ”;

@output() public bEvent:EventEmitter<an> = new

EventEmitter();

bEvent.emit(20);