A javascript framework for developing SPA.

Developed by Google

Uses typescript/javascript/DART

**Features**

Modularity - can use individual modules and can combine to develop large Apps

Easy

Dependency Injection support

Bindings

Routing

REST Api Support

Event handling

Template Support

Angular CLI used to create templates and components

Angular Supports many IDE like Visual studio and so on

Tools – Angular CLI , IDE

Why Angular

* React is not dedicated for SPA, Uses lots of Third Party tools , not suitable for large apps ,
* As angular provide many built in features for routing , binding , dependency injection and etc

Requirements - NodeJS, TypeScript , Angular CLI

Npm install –g @angular/cli

Once intall the angularCLI we wil get a command ng command

Ng new ,Ng generate

Angular1 is purely uses Javascript , uses MVVM patter , and contains scopes , filters, controller, Directives

**Angular2**

Components(Replacement of scope and controllers)

Pipes (Replacement of filters )

Services

Directives

***Architecture***

Module is the main building block ,

Application is combination of different modules

Module = collection of classes . (its package or namespace )

**An application can have more than one module**

Must have only one main module

**Angular Module**

**Component**

Code{}

Html<>

**Pipes**

|currency

**Services**

Class data{

}

**Directives**

Code{}

Other Modules

Built in , Routing,..

**AppModule :**

---------------------------------------------

@ngModule (

Declaration:[AppComponent,HomeComponent, AboutComponent,.. Components, pipes, directives ],

Providers :[MovieService,UserService, Services],

Imports:[UserModule,BuiltinModules,BrowserModule, all the modules],

Exports :[ModuleNames to be exported ,..],

entryComponets :[dynamic Components ]

Bootstrap:[AppComponent]// setting the startup

)

Export class AppModule {

}

Let us consider in a project which is having first, second , third Module

@ngModule for FirstModule

Declarations – Compoenenets for First Module

Imports – secondmodule , third Module , other module

Exports – Exports the modules to other modules which is consuming appmodule

Importing some module , adding some more functionality and export

Convention

Module names should ends with modules

Service names should ends with service

There must be one Root Component , generally it will be with Name APP

Main module should contain Root Component

Bootstrap Sequence

Index.html - <app></app> -🡪 main.ts 🡪 appModule 🡪 finds app and display

**Component**

@component({

Selector:home,

Template:`html code tags`, // template is used when we want render some inline html

Templateurl:’/htmlpageurl.html’, // hmtl file url

Styles:`[ all styles can be written ]`,

StylesUrls:[styleFilename.css,home.css,..All the css file required ]

})

Export class Home{

Constructor(){}

doClick(){}

}

On home page u can have

<div>

<button id=”btn” (click)=’doClick’ />

</div>

* Angular Events ie click must be in ().

**Ex: to Create a new component**

Create a new file in app folder >> name it as sample.component.ts

import {Component} from '@angular/core'

@Component({

selector:'sample-comp',

template:'<h1>Sample Component</h1>',

styles:["h1{color:red}"]

})

export class SampleComponent{

title='Sample Component'

}

**Now at appModule.ts we need to register the component . write the following**

import {SampleComponent } from './sample.component'

@NgModule({

declarations: [

AppComponent,SampleComponent

],

})

**To Dispaly the sample component we need to use the selector**

At appcomponent.html include the selector tag for samplecomponent

<div style="text-align:center">

<h1>

Welcome to {{ title }}!

</h1>

</div>

<h2>Here are some links to help you start: </h2>

<ul>

</ul>

**<sample-comp></sample-comp>**

**Bindings**

Rendering the data onto html controls is called binding

**Types of Bindings :**

One way binding

Interpolation

Property binding

Two way binding

Event binding

**One way Binding :**

Just Display the data on to the page . Changes in the value will not reflect

Ex: Interpolation binding , Property Binding

**Interpolation**  ( Expression binding) - one Way Binding

Use {{ }} Ex: <p>{{name}}</p> wil print the value of name variable in paragraph

Ex:

export class HomeComponent implements OnInit {

private name : string ="Rama";

constructor() { }

ngOnInit() {

}

}

Html View

<p>

Hi Mr. {{name}}

</p>

**Property binding**

Mostly used for binding form controls like textboxes , list and etc

Use [] ex : [Property] = “nameofVar”

Ex:

export class HomeComponent implements OnInit {

private name : string =”Rama”;

constructor() { }

ngOnInit() {

}

}

Html view

<p>

User name : <input type=”text” [value] = “name” />

</p>

**Event Binding**

Used to bind the events of html controls. Use () to bind the event

Syn : (eventName ) = ‘functionName()’

Ex:

export class HomeComponent implements OnInit {

constructor() { }

btnClick(){

alert('Hi Button Clicked...')

}

ngOnInit() {

}

}

Html view

<button (click)="btnClick()">Click Me </button>

**Two way binding**

After binding data , if any change in the source will be reflect at target

We use [(ngModel)] to bind the data

* If we are using ngModel , we need to import

Another way of implementing two way binding by refering the html element by using #

Use keyup event to send the data to the component variable and bind that in html

Ex:

export class HomeComponent implements OnInit {

private twb: string=""

constructor() { }

orgPress(orgVal)

{

this.twb= orgVal.value;

}

}

Html View

<input type="text" id="txtOrgName" #org (keyup)="orgPress(org)" />

<h1 [style.color]="txtColor">MANIDEEP</h1>

**Directive**

A directive is a customTag or a attribute

Structural Directives - used to adding elements to the dom basing on condition

\*ngIf and, \*ngFor, \*ngSwitch

Behavioural directive – not changing the dom , but changing the behavior of the element

If we add click event handler to a button , that means we adds something for

Button element

Ex: highlight , click , ..

Component Directive - Directives with template

A custom tag which is associated with additional view . generally component we are creating are component directives

**\*ngif** - conditionally display the element

Ex:

<p \*ngIf="dept==='IT'">

Happy teachers day

</p>

The above paragraph will appear only if the dept value is IT , and disappear if false

\*ngforEx:

Used to repeat the loop for number of times . we can iterate through each item of the source

Class

{

Private courses : string[];

Constructor()

{

This.courses =[‘Angulrar’,’Java’,’React’,’Mongo’];

}

}

<ul>

<li \*ngfor=”let cr of courses”>{{cr}}</li>

</ul>

**Note :**

**Ng generate component components/ComponentName**

**Will generate a component with complete .ts, .css, .html files**

**We can generate pipes, service, component and etc with ng generate command**

**ngStyle** – use to apply dynamic style for html elements

<p [ngStyle]={'color':'colorName'} >

If we use ngStyle we can provide css properties with variable value

Ex:

Component.html

Enter U r color : <input type="text" #clrName (keyup)="kuClr(clrName)"/>

<h1 [ngStyle]="{'color':ColorName}">ANGULAR2</h1>

Componenet.ts

export class HomeComponent implements OnInit {

private ColorName: string;

constructor() { }

kuClr(cl)

{

this.ColorName = cl.value;

}

ngOnInit() {

}

}

**ngClass**

Apply the styles based on a condition

Component.ts

export class ExampleComponent implements OnInit {

private Employees:any[];

constructor() {

this.Employees=[{"id":"1001","name":"Rama","city":"Hyd"},{"id":"1002","name":"Raja","city":"Bang"},{"id":"1003","name":"Rana","city":"Hyd"}]

}

ngOnInit() {

}

}

Component.css

.highlight

{

color: black;

background-color: yellow

}

Component.html

<div \*ngFor="let em of Employees" >

<p>Id:{{emid}}</p>

<h3>Name:{{em.name}}</h3>

<p [ngClass]="{'highlight':em.city==='Hyd'}">City:{{em.city}}</p>

</div>

**ngSwitch**

* **Decides which element to be displayed in the view**
* **We need to define the styles first at .css file**

component.html

<div \*ngFor="let em of Employees" [ngSwitch]="em.city" >

<p>Id:{{emid}}</p>

<h3>Name:{{em.name}}</h3>

<p class="hc" \*ngSwitchCase="'Hyd'">{{em.city}}</p>

<p class ="bc" \*ngSwitchCase="'Bang'">{{em.city}}</p>

<p \*ngSwitchDefault>{{em.city}}</p>

</div>

Component.css

.hc

{

color: black;

background-color: yellow

}

.bc

{

color: black;

background-color: red

}

**ngNonBindable** - to disable the binding for any element we can use ngNonBindable

**Sample Example** of EventManagement Example

.row-display-flex{

Display:flex;

Flex-wrap:wrap;

}

.thumbnail{

Height:100%

\}

.event-title{

// for showing tool tip of a heading of

White-space : nowrap;

Overflow:hidden;

Text-overflow: ellipsis

}

**Pipes**  - the pipes are formatters , used to format the data

Uppercase

Lowercase

Date

Number

Currency

**Working with Forms**

**Reactive Forms :**

A form building technology for angular Applications

Uses the **ReativeFormsModule** from **@angular/forms Library**

With reactive forms, you create a tree of Angular form control objects in the component class and bind them to native form control elements in the component template,

Advantage is that value and validity updates are [always synchronous and under your control](https://angular.io/guide/reactive-forms#async-vs-sync)

**Steps to work with reactive forms**

1. Import ReactiveFormsModule at appModule.ts from @angular/forms library
2. And specify that ReactiveFormsModule at imports section of @ngModule({})
3. At Component import FormGroup, FormBuilder, Validators from a@angular/forms
   1. import {FormGroup, FormBuilder, Validators} from '@angular/forms'
4. Create a FormGroup variable at .ts file and add the controls using FormBuilder

this.frmGrp = this.fb.group({

fname: ['', Validators.required],

lname: ['', Validators.required]})

1. ControlName at FormBuilder in .ts file should match the formControlName at html file

<input t type="text" formControlName="fname" />

1. Bind the formGroup property of <form> with FormGroup Variable
   1. <form [formGroup]= “frmGrp” >
2. all validations can be checked by using formGroup.valid
3. all the values can get by using formGroup.value | json or formGroup.value.Controlname

**To validate an individual control**

We can use formcontrol validation properties

Valid

Invalid

Dirty

Prestine

Each formcontrol accessed using formGroup

* this.frmGrp.controls["fname"].invalid – return true if any of the validations on that formcontrol evaluates false

Ex:

**At appModule.ts**

import { ReactiveFormsModule, FormGroup} from '@angular/forms'

@NgModule({

imports: [

BrowserModule, ReactiveFormsModule

],

})

export class AppModule { }

At Component.ts

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

import {FormGroup, FormBuilder, Validators} from '@angular/forms'

@Component({

selector: 'app-employee',

templateUrl: './employee.component.html',

styleUrls: ['./employee.component.css']

})

export class EmployeeComponent implements OnInit {

private frmGrp : FormGroup;

private frmBdr : FormBuilder;

private venomsg : string

constructor() {

this.venomsg="";

this.frmBdr = new FormBuilder();

}

btnSubmit()

{

let veno = this.frmGrp.controls["eno"];

if(veno.invalid==true || veno.untouched == true || veno.dirty== true)

{

this.venomsg = "Plz Check Eno .."

}

else

{

this.venomsg="";

}

//alert(this.frmGrp.valid)

}

ngOnInit() {

this.frmGrp = this.frmBdr.group({

eno:['',Validators.required],

ename:['', Validators.required],

deptId:['', Validators.required],

mngrId:['', Validators.required],

city :[''],

salary:['', Validators.required]

})

}

}

**Component.html**

<div class = "container">

<div class="row">

<h2>Employee Register</h2>

</div>

<div class ="row">

<div class="col-md-offSet-3 col-md-6">

<form [formGroup] = "frmGrp">

<div class="form-group">

Eno:<input type="text" class="form-control" formControlName="eno" /><span style ="color:red">{{venomsg}}</span>

</div>

<div class="form-group">

Name:<input type="text" class="form-control" formControlName="ename" />

</div>

<div class="form-group">

DepartmentID:<input type="text" class="form-control" formControlName="deptId" />

</div>

<div class="form-group">

ManagerId:<input type="text" formControlName="mngrId" class="form-control" />

</div>

<div class="form-group">

City:<input type="text" formControlName="city" class="form-control"/>

</div>

<div class="form-group">

Salary:<input type="text" formControlName="salary" class="form-control"/>

</div>

<div class="form-group">

<button class="btn btn-primary" (click)="btnSubmit()">Save</button>

</div>

</form>

</div>

</div>

</div>

**To Apply Bootstrap in Agular2 Application**

First install bootstrap and jquery

Npm install jquery –-save

Npm install bootstrap –-save

At anglar.json File include

**"styles": [**

**"./node\_modules/bootstrap/dist/css/bootstrap.min.css",**

**"src/styles.css"**

**],**

**"scripts": ["./node\_modules/jquery/dist/jquery.min.js",**

**"./node\_modules/bootstrap/dist/js/bootstrap.min.js"]**

**Routing in Angular**

We use Router Service

A routed Angular application has one singleton instance of the Route service.

When the browser's URL changes, the router looks for a corresponding Route from which it can determine the component to display.

A router has no routes until you configure it.

**Steps to Implement Routing**

1. Import RouterModule and Routes at appmodule.ts from @angular/Router
2. Create a array of routes with path and component
   1. {path:’home’, component:HomeCompoenet}
3. Specify at import of AppModule.ts - RouterModule.forRoot(u r RoutesArray)
4. At aappComponent.html Page navbar specify the hyper link with routerLink=”path”
5. Should specify the routerLink = path which is specified at RoutesList

<router-outlet > element work as a content place holder for the components on the page

Ex:

**Appmodule.ts File**

import { RouterModule , Routes} from '@angular/router'

const routArray : Routes= [

{path:'', component:HomeComponent},

{path:'contact', component : ContactUsComponent},

{path:'about',component:AboutUsComponent}

];

@NgModule({

imports: [

RouterModule.forRoot(routArray)

],

})

**AppComponent.html**

<a routerLink="">Home</a>

<a routerLink="contact">About</a>

<a routerLink="about">Contact</a>

<router-outlet></router-outlet>

**Bootstrap 4 Appcomponet with Nav Bar Ex:**

<nav class="navbar navbar-expand-sm bg-primary navbar-dark">

<ul class="navbar-nav">

<li class="nav-item ">

<a class="nav-link" routerLink="home">Home</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="addEmployee">AddEmployee</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="login">Login</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="login">Update Employee</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="register">Register</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="searchEmployee">SearchEmployee</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="updateEmployee">updateEmployee</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="ShowAllEmployee">ShowAllEmployee</a>

</li>

</ul>

</nav>

<div>

<router-outlet></router-outlet>

</div>

<nav class="navbar navbar-expand-sm bg-primary navbar-dark">

<h6>

<font color="white">Developped by OssBatch1 @ Unic Solutions</font>

</h6>

</nav>

**Working with LifeCycle Hooks**

An angular component under goes different stages in its life cycle . Angular will maintain the lifecycle of a components

Create the compoennt

Render the component

Create and render the Child Components for a component

Checks when the component binding properties values changes

Destroy the component before removing it from the DOM

All the life cycle events were written as interfaces , if we want to check the events occurance and execute some logic based on the lifecycle event, we need to inherit the interface and implement the hook method of the interface

* **ngOnChanges** − When the value of a data bound property changes, then this method is called.
* **ngOnInit** − This is called whenever the initialization of the directive/component after Angular first displays the data-bound properties happens.
* **ngDoCheck** − This is for the detection and to act on changes that Angular can't or won't detect on its own.
* **ngAfterContentInit** − This is called in response after Angular projects external content into the component's view.
* **ngAfterContentChecked** − This is called in response after Angular checks the content projected into the component.
* **ngAfterViewInit** − This is called in response after Angular initializes the component's views and child views.
* **ngAfterViewChecked** − This is called in response after Angular checks the component's views and child views.
* **ngOnDestroy** − This is the cleanup phase just before Angular destroys the directive/component.

**Dependecy Injection**

Service to client class directly will be type dependency

To overcome type dependency , IOC is introduced (inversion f control )

Passing the object from out side and sending it to the consumer class constructor and aborting the control of the consumers class on the object is called IOC

So here dependency was created indirectly through other is called dependency injection

IOC is implemented by DI or DI can be achievd by IOC

Activity : Dependency Injection

**Working with Services**

To create service

Ng g service services/ serviceName

To make it as an injectable service we can use @Injectable. Inorder to achieve the dependency injection we will inject the service into a component constructor method

After creating service we need to register the service

At a component level

@Component({

selector: 'app-event-list',

templateUrl: './event-list.component.html',

styleUrls: ['./event-list.component.css'],

providers:[serviceName]

})

**Or we can register service globally**

App.module.ts – We need to import the service and then

specify the service at providers:[service List ]

**CRUD Operation in Angular 4.3**

HttpClientModule is the base library from @angular/common/http

Use HttpClient to communicate with rest Api

With httpClinet we can make get , put , post and delete , request . the previous http developer need to map the data into json format .

With httpclient default expected response data is json . so no need to convert again into json

All the get , put, post methods will returns Observable type .. so need to import Observable from rxjs

Ex:

Create new Application

At appModule.ts

import { BrowserModule } from '@angular/platform-browser';

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

import {ReactiveFormsModule } from '@angular/forms'

import { RouterModule , Routes} from '@angular/router'

**import { HttpClientModule} from '@angular/common/http'**

import other User defined components

//Speciying Routes

const myRoutes : Routes=[

{path:'home' ,component:HomeComponent},

{path:'addEmployee', component:EmployeeComponent},

{path:'login', component:LoginComponent},

{path :'register', component: RegisterComponent},

{path :'searchEmployee' , component:SearchEmployeeComponent},

{path : 'showAllEmployee', component: DispalyAllEmpComponent},

{path:'updateEmployee', component:UpdateEmpComponent},

{path:'editEmployee/:id', component:EditEmployeeComponent}

]

@NgModule({

declarations: [

user defined Components

],

imports: [

BrowserModule,

ReactiveFormsModule,

RouterModule.forRoot(myRoutes),

**HttpClientModule**

],

providers: [EmployeeService],

bootstrap: [AppComponent]

})

export class AppModule { }

**AppComponent.Html**

<nav class="navbar navbar-expand-sm bg-primary navbar-dark">

<ul class="navbar-nav">

<li class="nav-item ">

<a class="nav-link" routerLink="home">Home</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="addEmployee">AddEmployee</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="login">Login</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="updateEmployee">UpdateEmployee</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="register">Register</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="searchEmployee">SearchEmployee</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="showAllEmployee">ShowAllEmployee</a>

</li>

</ul>

</nav>

<div>

<router-outlet></router-outlet>

</div>

<nav class="navbar navbar-expand-sm bg-primary navbar-dark">

<h6>

<font color="white">Developped by OssBatch1 @ Unic Solutions</font>

</h6>

</nav>

**Creating Models**

We need maintain TypeSafety of the dta we are sending or retrieving from DB. We use a class to maintain the schema as well as type safety

Add a folder in app folder and name it as models >> add a file and name its as employee.ts

export class employee{

eno: number;

ename:string;

deptid:number;

mngrid: number;

city: number;

salary : string;

}

Our model class should represent the table from our Database

Creating Components for CRUD

**Creating Services**

Inodrer to communicate with db we use service class so that all the database communication logic will be in a service class. Which give a clear seperation of logic from component

Ng generate service services/employee service

Will generate a service in services folder with name employee service

At appmodule.ts file we need to import this service and also specify at providers :[servicename] array

Implement all the CRUD operation methods in this service Operation

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

import { employee } from '../models/employee'

import { HttpClient, HttpHeaders } from '@angular/common/http'

import { Observable } from 'rxjs'

@Injectable({

providedIn: 'root'

})

export class EmployeeService {

constructor(private http: HttpClient) {

}

GetAllEmployees(): Observable<any> {

return this.http.get('http://localhost:4500/employee',{responseType:'json'});

}

GetEmployeeById(n: number): Observable<any> {

return this.http.get('http://localhost:4500/employee/' + n, { responseType: 'json' });

}

SaveEmployee(emp: employee) {

const httpOptions = {

headers: new HttpHeaders({ 'Content-Type': 'application/json' })

};

this.http.post('http://localhost:4500/employee', JSON.stringify(emp), httpOptions).subscribe(res => console.log((res)));

alert("Saved Success.......");;

}

DeleteEmployee(n: number): Observable<any>{

console.log(n);

return this.http.delete('http://localhost:4500/employee/' + n, { responseType: 'json' })

}

}

Now u can create components to consume the service and communicate with database

**Add Employee Component**

Ng generate component Addemployeecomponent- component will be generated

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

import { FormGroup, FormBuilder, Validators } from '@angular/forms'

import { employee} from '../models/employee'

import {EmployeeService} from '../services/employee.service'

@Component({

selector: 'app-employee',

templateUrl: './employee.component.html',

styleUrls: ['./employee.component.css']

})

export class EmployeeComponent implements OnInit {

private frmGrp: FormGroup;

private frmBdr: FormBuilder;

private venomsg: string

constructor(private empService:EmployeeService) {

this.venomsg = "";

this.frmBdr = new FormBuilder();

}

btnSubmit() {

let emp = new employee();

emp.eno = this.frmGrp.controls["eno"].value;

emp.ename = this.frmGrp.controls["ename"].value

emp.deptid = this.frmGrp.controls["deptId"].value

emp.mngrid = this.frmGrp.controls["mngrId"].value

emp.city = this.frmGrp.controls["city"].value

emp.salary = this.frmGrp.controls["salary"].value

this.empService.SaveEmployee(emp);

}

ngOnInit() {

this.frmGrp = this.frmBdr.group({

eno: ['', Validators.required],

ename: ['', Validators.required],

deptId: ['', Validators.required],

mngrId: ['', Validators.required],

city: [''],

salary: ['', Validators.required]

})

}

}

**Add Employeecomponent.Html**

<div class = "container">

<div class="row">

<h2>Employee Register</h2>

</div>

<div class ="row">

<div class="col-md-offSet-3 col-md-6">

<form [formGroup] = "frmGrp">

<div class="form-group">

Eno:<input type="text" class="form-control" formControlName="eno" /><span style ="color:red">{{venomsg}}</span>

</div>

<div class="form-group">

Name:<input type="text" class="form-control" formControlName="ename" />

</div>

<div class="form-group">

DepartmentID:<input type="text" class="form-control" formControlName="deptId" />

</div>

<div class="form-group">

ManagerId:<input type="text" formControlName="mngrId" class="form-control" />

</div>

<div class="form-group">

City:<input type="text" formControlName="city" class="form-control"/>

</div>

<div class="form-group">

Salary:<input type="text" formControlName="salary" class="form-control"/>

</div>

<div class="form-group">

<button class="btn btn-primary" (click)="btnSubmit()">Save</button>

</div>

</form>

</div>

</div>

</div>

**DisplayAllemployeeComponent.ts**

Ng generate component displayallemployee – component will be generated

Displayalallemployee.ts

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

import { EmployeeService } from '../services/employee.service'

import {Router} from '@angular/router'

@Component({

selector: 'app-dispaly-all-emp',

templateUrl: './dispaly-all-emp.component.html',

styleUrls: ['./dispaly-all-emp.component.css']

})

export class DispalyAllEmpComponent implements OnInit {

private EmpList : any[]

constructor(private empService: EmployeeService, private router : Router) {

}

btnDelete(n:number){

//alert(n);

this.empService.DeleteEmployee(n).subscribe(data=>console.log(data));

location.reload();

}

btnEdit(n: number)

{

this.router.navigate(['/editEmployee/' + n])

}

ngOnInit() {

this.empService.GetAllEmployees().subscribe((data)=>{this.EmpList= data;});

console.log(this.EmpList)

}

}

**displayAllemployee.html**

<h4>Employee List </h4>

<div \*ngIf='EmpList'>

<table class="table" >

<tr \*ngFor="let em of EmpList">

<td>{{em.eno}}</td>

<td>{{em.ename}}</td>

<td>{{em.city}}</td>

<td>

<button class ="btn btn-primary">Details</button>

<button (click)="btnEdit(em.eno)"class ="btn btn-success" >Edit</button>

<button (click) ="btnDelete(em.eno)"class ="btn btn-danger">Del</button>

</td>

</tr>

</table>

</div>