Angular Basics:

What is Angular?

Angular is a JavaScript framework which makes you able to create reactive **Single Page Applications** (SPAs). This is a leading front-end development framework which is regularly updated by Angular team of Google. Angular 8 is completely based on components. It consists of several components forming a tree structure with parent and child components.

Angular 7:

Angular7 is a JavaScript **(actually a TypeScript based open-source full-stack web application)** framework which makes you able to create reactive **Single Page Applications** (SPAs). Angular 7 is completely based on components. It consists of several components which forms a tree structure with parent and child components. Angular's versions beyond 2+ are generally known as **Angular** only. The very first version Angular 1.0 is known as **AngularJS.**

"Angular is a complete rewrite of AngularJS by the same team that built AngularJS."

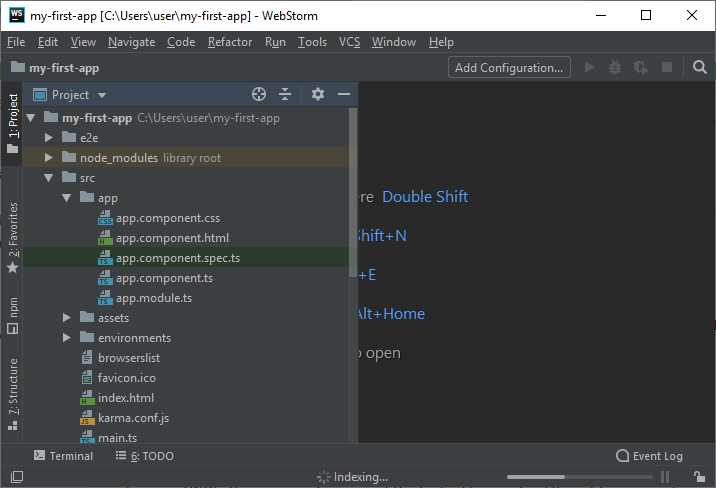
SPA:

A single page application is a web application or a website which provides users a very fluid, reactive and fast experience similar to a desktop application. It contains menu, buttons and blocks on a single page and when a user clicks on any of them; it dynamically rewrites the current page rather than loading entire new pages from a server. That's the reason behind its reactive fast speed.

Installation:

1. Install VS Code.  **https://code.visualstudio.com**
2. Install Node.js **https://nodejs.org/en/**
3. Use npm to install Angular: npm install -g @angular/cli
4. node –v 🡪 To check the version of Node.js
5. npm –v 🡪 To check the version of npm
6. To Create a new project in Angular: 🡪 ng new projectname
7. Ng serve is to run the project in the web server

Folder Structure:



**You will see 5 components there:**

* app.component.css
* app.component.html
* app.component.spec.ts
* app.component.ts
* app.module.ts

Files Used in Angular :

Angular7 App files which are mainly used in your project are given below:

**src folder:** This is the folder which contains the main code files related to your angular application.

**app folder:** The app folder contains the files, you have created for app components.

**app.component.css:** This file contains the 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.

**package.json:** This is npm configuration file. It includes details about your website's package dependencies along with details about your own website being a package itself.

**package-lock.json :** This is an auto-generated and modified file that gets updated whenever npm does an operation related to node\_modules or package.json

**angular.json:** It is very important configuration file related to your angular application. **It defines the structure of your app and includes any settings associated with your application.** Here, you can specify environments on this file (development, production). This is the file where we add Bootstrap file to work with Angular7.

**.gitignore:** This file is related to the source control git.

**.editorconfig:** This is a simple file which is used to maintain consistency in code editors to organize some basics such as indentation and whitespaces.

**assets folder:** This folder is a placeholder for resource files which are used in the application such as images, locales, translations etc.

**environments folder:** The environments folder is used to hold the environment configuration constants that help when building the angular application. The constants are defined in 2 separate .ts files (environment.ts and environment.prod.ts), where these constants are used within the angular.json file by the Angular CLI. For example, if you run the ng build command, it will build the application using the development environment settings, whereas the command ng build ?prod will build the project using the production environment settings.

**browserlist :** This file is used by autoprefixer that adjusts the CSS to support a list of defined browsers.

**favicon.ico**This file specifies a small icon that appears next to the browser tab of a website.

**index.html:** This is the entry file which holds the high level container for the angular application.

**karma.config.js:** This file specifies the config file for the Karma Test Runner, Karma has been developed by the AngularJS team which can run tests for both AngularJS and Angular 2+

**main.ts:** As defined in angular.json file, this is the main ts file that will first run. This file bootstraps (starts) the AppModule from app.module.ts , and it can be used to define global configurations.

**polyfills.ts:** This file is a set of code that can be used to provide compatibility support for older browsers. Angular 7 code is written mainly in ES6+ language specifications which is getting more adopted in front-end development, so since not all browsers support the full ES6+ specifications, pollyfills can be used to cover whatever feature missing from a given browser.

**styles.css:/** This is a global css file which is used by the angular application.

**tests.ts:** This is the main test file that the Angular CLI command ng test will use to traverse all the unit tests within the application and run them.

**tsconfig.json:** This is a typescript compiler configuration file.

**tsconfig.app.json:** This is used to override the tsconfig.json file with app specific configurations.

**tsconfig.spec.json:** This overrides the tsconfig.json file with app specific unit test configurations.

Angular Components:

Components are the key features of Angular. The whole application is built by using different components.

The core idea behind Angular is to build components. They make your complex application into reusable parts which you can reuse very easily.

To Create a New Component:

* ng generate component component\_name   (OR)
* ng g c component\_name

When we create a new component, 4 files gets generated:

* server2.component.css
* server2.component.html
* server2.component.spec.ts
* server2.component.ts

Directives:

Directives are instructions in the DOM. They specify how to place your components and business logic in the Angular.

Directives are js class and declared as @directive. There are 3 directives in Angular.

* Component Directives
* Structural Directives
* Attribute Directives

**Component Directives:** Component directives are used in main class. They contain the detail of how the component should be processed, instantiated and used at runtime.

**Structural Directives:** Structural directives start with a \* sign. These directives are used to manipulate and change the structure of the DOM elements. For example, \*ngIf and \*ngFor.

**Attribute Directives:** Attribute directives are used to change the look and behavior of the DOM elements. For example: ngClass, ngStyle etc.

## Difference between Attribute Directive and Structural Directive

|  |  |
| --- | --- |
| **Attribute Directives** | **Structural Directives** |
| Attribute directives look like a normal HTML Attribute and mainly used in databinding and event binding. | Structural Directives start with a \* symbol and look different. |
| Attribute Directives affect only the element they are added to. | Structural Directives affect the whole area in the DOM. |

We can create our own custom directives also in Angular.

You have seen the attribute directive like ngClass and ngStyle. Now, it's time to create our own attribute directives.

First, create a folder. Let's name it "simple-style". Then, create a file within that folder named as "simple-style.directive.ts"

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

 @Directive( {

  selector: '[appSimpleStyle]'

})

export class SimpleStyleDirective implements OnInit {

  constructor(private elementRef: ElementRef) {

  }

  ngOnInit() {

  this.elementRef.nativeElement.style.backgroundColor = 'green';

  }

Now, you have to inform Angular that you have a new directive. So, you have to add **SimpleStyleDirective to app.module.ts** and also import it.

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

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

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

import {SimpleStyleDirective} from './simple-style/simple-style.directive';

@NgModule({

  declarations: [

    AppComponent,

    SimpleStyleDirective

  ],

  imports: [

    BrowserModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

Now, your directive is created. Let's check it. Open app.component.html and use your created **SimpleStyleDirective**

<p appSimpleStyle>Style me with your created SimpleStyleDirective</p> </p>

**<div** class="container"**>**

**<div** class="row"**>**

**<div** class="col-xs-12"**>**

**<h2>**My Servers**</h2>**

**<button** class="btn btn-primary" (click)="OnAddServer()"**>**Add Server**</button>**

**<br><br>**

**<ul** class="list-group"**>**

**<li**

   class="list-group-item "

   \*ngFor="let server of servers; let i = index"

   (click)="onRemoveServer(i)"**>** {{ server }}

**</li>**

**</ul>**

**<p** appSimpleStyle**>**Style me with your created SimpleStyleDirective**</p>**

**</div>**

**</div>**

**</div>**

\*ngIf Structural Directive:

**component.ts file:**

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

@Component({

  selector: 'app-server2',

  templateUrl: './server2.component.html',

  styleUrls: ['./server2.component.css']

})

export class Server2Component implements OnInit {

 allowNewServer = false;

 serverCreationStatus = 'No server is created.';

  serverName = 'TestServer';

  serverCreated = false;

  /\*constructor() {

    setTimeout(() =**>**{

      this.allowNewServer = true;

    }, 5000);

  }\*/

  ngOnInit() {

  }

  onCreateServer() {

    this.serverCreated = true;

    this.serverCreationStatus = 'Server is created. Name of the server is' + this.serverName;

  }

  OnUpdateServerName(event: Event) {

    this.serverName = (**<HTMLInputElement>**event.target).value;

  }

}

**component.html file:**

**<p>**

  Server2 is also working fine.

**</p>**

**<label>**Server Name**</label>**

<!--**<input** type="text"

       class="form-control"

       (input)="OnUpdateServerName($event)"**>**--**>**

**<input** type="text"

       class="form-control"

[(ngModel)]="serverName"**>**

<!--<p>{{serverName}}</p>-->

**<button**

  class="btn btn-primary"

  [disabled]="allowNewServer"

  (click)="onCreateServer()"**>**Add Server**</button>**

**<p** \*ngIf="serverCreated"**>** Server is created. Server name is {{serverName}}**</p>**

## \*ngIf directive with an Else condition

**<p** \*ngIf="serverCreated; else noServer"**>** Server is created. Server name is {{serverName}}**</p>**

**<ng-template** #noServer**>**

**<p>**No Server is created.**</p>**

**</ng-template>**

Install Bootstrap in Angular:

npm install --save bootstrap@3

Open  .angular.js file. Add the following line:

"styles": [

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

  "src/styles.css"

],

Angular DataBinding:

Angular Databinding is used for communication. It is used to communicate between your TypeScript code (your business logic) and the other component which is shown to the users i.e. HTML layout.

There are 2 types of Data binding:

* 1. One way Databinding
  2. Two Way Databinding

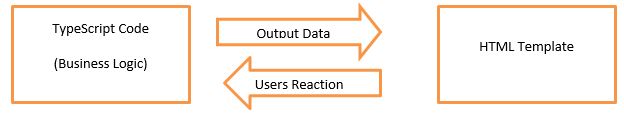
One way databinding is a simple one way communication where HTML template is changed when we make changes in TypeScript code.

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.

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.



ngStyle Attribute:

The ngStyle attribute is used to change or style the multiple properties of Angular. You can change the value, color, and size etc. of the elements.

**component.ts file:**

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

@Component(

  {selector: 'app-server',

    templateUrl: 'server.component.html'})

export class ServerComponent {

  serverID: number = 10;

  serverStatus: string = 'Offline';

  constructor () {

  this.serverStatus = Math.random() **>** 0.5 ? 'Online' : 'Offline';

}

 getServerStatus() {

  return this.serverStatus;

  }

}

**component.html file:**

**<p>**Server with ID {{serverID}} is {{serverStatus}}. **</p>**

Or

**<p** [ngStyle]="{backgroundColor: getColor()}"**>**  Server  with ID {{serverID}} is {{serverStatus}}. **</p>**

Apply styles dynamically with ngClass:

**component.ts file:**

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

@Component(

  {selector: 'app-server',

    templateUrl: 'server.component.html',

    styles: [`

    .Online{

      color: yellow;

    }`]

  })

export class ServerComponent {

  serverID: number = 10;

  serverStatus: string = 'Offline';

  constructor () {

    this.serverStatus = Math.random() **>** 0.5 ? 'Online' : 'Offline';

  }

  getServerStatus() {

    return this.serverStatus;

  }

  getColor() {

    return this.serverStatus === 'Online' ? 'green' : 'red';

  }

}

Component.html

**<p** [ngStyle]="{backgroundColor: getColor()}"

[ngClass]="{Online: serverStatus === 'Online'}"**>**  Server  with ID {{serverID}} is {{serverStatus}}. **</p>**

String Interpolation:

String interpolation is used to display dynamic data on HTML template (at user end). It facilitates you to make changes on component.ts file and fetch data from there to HTML template (component.html file).

Component.ts file:

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

@Component(

  {selector: 'app-server',

 templateUrl: 'server.component.html'})

export class ServerComponent {

  serverID: number = 10;

    serverStatus: string = 'Online';

}

Component.html file:

**<p>**Server with ID {{serverID}} is {{serverStatus}}. **</p>**

**String Interpolation and Property binding** both are used for same purpose i.e. one-way databinding. But the problem is how to know which one is best suited for your application.

String Interpolation and Property Binding doth are about one-way data binding. They both flow a value in one direction from our components to HTML elements.

String interpolation:

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

@Component({

    selector: 'my-app',

    template: `

**<h1>**{{ fullName }}**</h1>**

              `

})

export class AppComponent {

    fullName: string = 'Robert Junior';

}

Property Binding:

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

@Component({

    selector: 'my-app',

    template: `

**<h1** [innerHtml]='fullName'**></h1>**

              `

})

export class AppComponent {

    fullName: string = 'Robert Junior';

}

## Difference between String interpolation and Property Binding:

String Interpolation is a special syntax which is converted to property binding by Angular. It's a convenient alternative to property binding.

When you need to concatenate strings, you must use interpolation instead of property binding.

@Component({

    selector: 'my-app',

    template: `**<div>**

**<h1>**{{citedExample}}**</h1>**

**</div>**`

})

export class AppComponent {

    citedExample: string = 'Interpolation foe string only';

}

FirstApplication:

ngFor and ngIF:

**component.ts file**

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

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

@Component({

selector: 'app-contact-list',

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

styleUrls: ['./contact-list.component.css']

})

export class ContactListComponent implements OnInit {

angForm: FormGroup;

title:string="CTS";

constructor(private fb: FormBuilder) {

this.createForm();

}

createForm() {

this.angForm = this.fb.group({

name: ['', Validators.required ]

});

}

showActions: boolean = true;

contacts:any = [

{name: 'test1', email:'test1@test1.com',result:true},

{name: 'test2', email:'test1@test2.com',result:false},

{name: 'test3', email:'test1@test3.com',result:true},

{name: 'test4', email:'test1@test4.com',result:false}

]

ngOnInit() {

}

}

**component.html**

<!-- <h1>Contact List</h1>

<table border="1">

<thead>

<th>Name</th>

<th>Email</th>

<th>Result</th>

<th \*ngIf="showActions">Actions</th>

</thead>

<tbody>

<tr \*ngFor="let contact of contacts">

<td>

{{contact.name}}

</td>

<td>

{{contact.email}}

</td>

<td \*ngIf="contact.result;then show else notShow">

</td>

<ng-template #show>

<td>

Pass

</td>

</ng-template>

<ng-template #notShow>

<td>

Fail

</td>

</ng-template>

<td \*ngIf="showActions">

<button>Delete</button>

<button>Update</button>

</td>

</tr>

</tbody>

</table> -->

<hr/>

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

<h1>

Welcome to {{title}}!!

</h1>

<form [formGroup]="angForm" novalidate>

<div class="form-group">

<label class="center-block">Name:

<input class="form-control" formControlName="name">

</label>

</div>

<div \*ngIf="angForm.controls['name'].invalid && (angForm.controls['name'].dirty || angForm.controls['name'].touched)" class="alert alert-danger">

<div \*ngIf="angForm.controls['name'].errors.required">

Name is required.

</div>

</div>

</form>

<p>Form value: {{ angForm.value | json }}</p>

<p>Form status: {{angForm.status | json }}</p>

</div>

**Parent.html**

<h3>Parent Component</h3>

<label>Bitcoin price</label>

<input type="text" #pcomponent (keyup)="0"/>

<!-- <app-child [PData]="pcomponent.value"></app-child> -->

<p>Value of child component is:{{CData}} </p>

<app-child [PData]="pcomponent.value"

(childEvent)="CData=$event"></app-child>

**Parent.component.ts**

export class ParentComponent implements OnInit {

public CData: number;

constructor() { }

ngOnInit() {

}

}

**Child.component.html**

<h3>Child Component</h3>

<label>Child Component</label>

<input type="text" #ccomponent

(keyup)="onChange(ccomponent.value)"/>

<p>Value of parent component is:{{PData}} </p>

**Child.component.ts**

export class ChildComponent implements OnInit {

@Input() PData: number;

@Output() childEvent = new EventEmitter();

constructor() { }

ngOnInit() {

}

onChange(value) {

this.childEvent.emit(value);

}

}

**Emloyeelist.html**

<div class="panel panel-primary" \*ngFor="let employee of employees">

<div class="panel-heading">

<h3 class="panel-title">{{employee.name}}</h3>

</div>

<div class="panel-body">

<div class="col-xs-10">

<div class="row">

<div class="col-xs-4">

<img height="100" width="100" [src]="employee.photoPath">

</div>

<div class="col-xs-8">

<div class="row">

<div class="col-xs-6">

Gender

</div>

<div class="col-xs-6">

:{{employee.gender}}

</div>

</div>

<div class="row">

<div class="col-xs-6">

contactPreference

</div>

<div class="col-xs-6">

:{{employee.contactPreference}}

</div>

</div>

<div class="row" \*ngIf="employee.email">

<div class="col-xs-6">

Email

</div>

<div class="col-xs-6">

:{{employee.email}}

</div>

</div>

<div class="row" [hidden]="!employee.phonenumber">

<div class="col-xs-6">

Phone Number

</div>

<div class="col-xs-6">

:{{employee.phonenumber}}

</div>

</div>

<div class="row">

<div class="col-xs-6">

dateOfBirth

</div>

<div class="col-xs-6">

:{{employee.dateOfBirth|date }}

</div>

</div>

<div class="row">

<div class="col-xs-6">

department

</div>

<div class="col-xs-6">

:{{employee.department}}

</div>

</div>

<div class="row">

<div class="col-xs-6">

IsActive

</div>

<div class="col-xs-6">

:{{employee.isActive}}

</div>

</div>

</div>

</div>

</div>

</div>

</div>

**Employeelist.component.ts**

export class EmplistComponent implements OnInit {

employees:Employee[]=[

{

id:1,

name:'Steve',

gender:'Male',

contactPreference:'Email',

email:'steve@gmail.com' ,

dateOfBirth:new Date('01/03/2010'),

department:'Apple',

isActive:true,

photoPath:'assets/steve.jpeg'

},

{

id:2,

name:'Bill Gates',

gender:'Male',

contactPreference:'Phone',

phonenumber:9850048765 ,

dateOfBirth:new Date('01/03/2012'),

department:'Microsoft',

isActive:true,

photoPath:'assets/billgates.jpeg',

},

{

id:3,

name:'Sundar',

gender:'Male',

contactPreference:'Email',

email:'Sundar@gmail.com' ,

dateOfBirth:new Date('05/07/2010'),

department:'Google',

isActive:true,

photoPath:'assets/sundar.jpeg',

}

]

constructor() { }

ngOnInit() {

}

}

**Registration form (Employee add.html)**

<form #employeeform="ngForm" ngNativeValidate (ngSubmit)="saveEmp(employee)">

<div class="panel panel-primary">

<div class="panel-heading">

<h3 class="panel-title"> Create Employee </h3>

</div>

<div class="panel-body">

<div class="form-group" [class.has-error]="name.invalid && name.touched" [class.has-success]="name.valid" >

<label for="name"> Full Name :</label>

<input required type="text" name="name" [(ngModel)]="employee.name"

#name="ngModel" id="name" class="form-control" >

<span class="help-block" \*ngIf="name.invalid">

Name is Required

</span>

</div>

<br/>

<h3> Employee Form level Validations:</h3>

<div> touched: {{employeeform.touched}} </div>

<div> Untouched: {{employeeform.untouched}} </div>

<div> Dirty: {{employeeform.dirty}} </div>

<div> Pristine: {{employeeform.pristine}} </div>

<div> Valid: {{employeeform.valid}} </div>

<div> Invalid: {{employeeform.invalid}} </div>

<div class="form-group" [class.has-error]="gender.touched && gender.invalid">

<label class="control-label"> Gender :</label>

<div class="form-control">

<label class="radio-inline">

<input type="radio" value="male" required #gender="ngModel" name="gender" [(ngModel)]="employee.gender" >Male

</label>

<label class="radio-inline">

<input type="radio" value="female" required name="gender" #gender="ngModel" [(ngModel)]="employee.gender" >Female

</label>

<span class="help-block" \*ngIf="gender.touched && gender.invalid">

Gender is required

</span>

</div>

</div>

<div class="form-group" [class.has-error]="contactPreference.touched && contactPreference.invalid">

<label class="control-label"> Contact Preference :</label>

<div class="form-control">

<label class="radio-inline">

<input type="radio" value="pno" [required]="contactPreference.value=='pno'" name="contact" #contactPreference="ngModel" [(ngModel)]="employee.contactPreference">Phone

</label>

<label class="radio-inline">

<input type="radio" value="email" [required]="contactPreference.value=='email'" #contactPreference="ngModel" name="contact" [(ngModel)]="employee.contactPreference">Email

</label>

</div>

<span class="help-block" \*ngIf="contactPreference.touched && contactPreference.invalid">

ContactPreference is required

</span>

</div>

<div class="form-group [class.has-error]=email.invalid && email.touched">

<label for="email" class="control-label"> Email :</label>

<input type="text" name="email" [(ngModel)]="employee.email" #email="ngModel"

class="form-control" required email>

<span class="help-block" \*ngIf="email.errors?.required && email.touched">

Email is Required

</span>

<span class="help-block" \*ngIf="email.errors?.email && email.touched">

Email is Invalid

</span>

</div>

<h3> Employee Email Validations:</h3>

<div> touched: {{email.touched}} </div>

<div> Untouched: {{email.untouched}} </div>

<div> Dirty: {{email.dirty}} </div>

<div> Pristine: {{email.pristine}} </div>

<div> Valid: {{email.valid}} </div>

<div> Invalid: {{email.invalid}} </div>

<div class="form-group">

<label class="radio-inline"> PhoneNumber :</label>

<input type="text" name="phno" [(ngModel)]="employee.phonenumber" class="form-control">

</div>

<div class="form-group" [class.has-error]="isActive.touched && isActive.invalid">

<div class="form-control" >

<label class="checkbox-inline control-label" >

<input type="checkbox" name="isActive" [required]="employee.isActive==null" #isActive="ngModel" [(ngModel)]="employee.isActive" > IsActive:

</label>

</div>

<span class="help-block" \*ngIf="isActive.touched && isActive.errors?.required">

IsActive is Required

</span>

</div>

<br/>

<div class="form-control">

<label for="department"> Department :</label>

</div>

<br/>

<div>

<select id="department" name="department" [(ngModel)]="employee.department" class="form-control" >

<!-- <option value="1"> Help Desk </option>

<option value="2"> HR </option>

<option value="3"> Payroll </option>

<option value="4"> IT </option> -->

<option \*ngFor="let dept of departments" [value]="dept.id">

{{dept.name}}

</option>

</select>

</div>

<div class="row">

<div class="form-group col-md-3">

<label for="dateOfBirth"> Date Of Birth :</label>

<!-- <input type="text" placement="right" bsDatepicker [bsConfig]=datePickerConfig name="dateOfBirth" -->

<input type="date" name="dateofBirth"

[(ngModel)]="employee.datefBirth" id="dateOfBirth" class="form-control">

</div>

</div>

<div class="form-group">

<label for="photoPath"> Photo Path:</label>

<input type="text" name="photoPath" [(ngModel)]="employee.photoPath" id="photoPath" class="form-control">

</div>

<div>

<button type="button" (click)="togglePhotoPreview()" class="btn btn-primary">

{{previewphoto? "Hide" : "Show "}} Preview

</button>

</div>

<div class="form-group">

<img [src]="employee.photoPath" height="200" width="200" \*ngIf="previewphoto"/>

</div>

</div>

<div style="margin-left:500px;margin-bottom: 20px;">

<button type="submit" class="btn btn-primary" [disabled]="employeeform.invalid"> Save </button>

</div>

</div>

</form>

Angular Generated Form Model: {{employeeform.value | json}}

<br/>

Our Employee Model: {{employee |json }}

**Employee.component.ts**

export class EmployeeAddComponent implements OnInit {

datePickerConfig:Partial<BsDatepickerConfig>;

dob:Date=new Date(2018,0,30);

previewphoto=false;

gender="male";

contact="pno";

isActive=true;

department=2;

employee: Employee={

id:null,

name:null,

gender:null,

contactPreference:null,

isActive:null,

phonenumber:null,

photoPath:null,

dateOfBirth:null,

department:null

};

departments: Department[]=[

{id:1,name:"HelpDesk"},

{id:2,name:"HR"},

{id:3,name:"IT"},

{id:4,name:"Payroll"},

{id:5,name:"admin"}

]

constructor() {

this.datePickerConfig=Object.assign({},{containerClass:'theme-dark-blue',

showWeekNumbers:false,

minDate:new Date(2018,0,1),

maxDate: new Date(2018,11,1),

dateInputFormat:'DD/MM/YYYY'

});

}

togglePhotoPreview()

{

this.previewphoto=!this.previewphoto;

}

ngOnInit() {

}

saveEmp(newEmployee:Employee):void

{

console.log(newEmployee);

}

}

**Employee.ts**

export class Employee

{

id:number;

name:string;

gender:string;

email?:string;

phonenumber?:number;

contactPreference:string;

dateOfBirth:Date;

department:string;

isActive:Boolean;

photoPath?:string;

}

**Department.ts**

export class Department

{

id:number;

name:string;

}

### **8 Lifecycle Hooks**

* ngOnChanges()
  + - Used in pretty much any component that has an input.
    - Called whenever an input value changes
    - Is called the first time before ngOnInit
* ngOnInit()
  + Used to initialize data in a component.
  + Called after input values are set when a component is initialized.
  + Added to every component by default by the Angular CLI.
  + Called only once
* ngDoCheck()
  + Called during all change detection runs
  + A run through the view by Angular to update/detect changes.
* ngAfterContentInit()
  + Called only once after first ngDoCheck()
  + Called after the first run through of initializing content
* ngAfterContentChecked()
  + Called after every ngDoCheck()
  + Waits till after ngAfterContentInit() on first run through
* ngAfterViewInit()
  + Called after Angular initializes component and child component content.
  + Called only once after view is initialized
* ngAfterViewChecked()
  + Called after all the content is initialized and checked. (Component and child components).
  + First call is after ngAfterViewInit()
  + Called after every ngAfterContentChecked() call is completed
* ngOnDestroy()
  + Used to clean up any necessary code when a component is removed from the DOM.
    - Fairly often used to unsubscribe from things like services.
  + Called only once just before component is removed from the DOM.

Forms in Angular:

Handling user input with forms is the cornerstone of many common applications. Applications use forms to enable users to log in, to update a profile, to enter sensitive information, and to perform many other data-entry tasks.

Angular provides two different approaches to handling user input through forms: reactive and template-driven. Both capture user input events from the view, validate the user input, create a form model and data model to update, and provide a way to track changes.

Reactive and template-driven forms process and manage form data differently. Each offers different advantages.

In general:

* Reactive forms are more robust: they're more scalable, reusable, and testable. If forms are a key part of your application, or you're already using reactive patterns for building your application, use reactive forms.
* Template-driven forms are useful for adding a simple form to an app, such as an email list signup form. They're easy to add to an app, but they don't scale as well as reactive forms. If you have very basic form requirements and logic that can be managed solely in the template, use template-driven forms.

## Key differences

The table below summarizes the key differences between reactive and template-driven forms.

|  | REACTIVE | TEMPLATE-DRIVEN |
| --- | --- | --- |
| Setup (form model) | More explicit, created in component class | Less explicit, created by directives |
| Data model | Structured | Unstructured |
| Predictability | Synchronous | Asynchronous |
| Form validation | Functions | Directives |
| Mutability | Immutable | Mutable |
| Scalability | Low-level API access | Abstraction on top of APIs |

## Common foundation

Both reactive and template-driven forms share underlying building blocks.

* [FormControl](https://angular.io/api/forms/FormControl) tracks the value and validation status of an individual form control.
* [FormGroup](https://angular.io/api/forms/FormGroup) tracks the same values and status for a collection of form controls.
* [FormArray](https://angular.io/api/forms/FormArray) tracks the same values and status for an array of form controls.
* [ControlValueAccessor](https://angular.io/api/forms/ControlValueAccessor) creates a bridge between Angular [FormControl](https://angular.io/api/forms/FormControl) instances and native DOM elements.

Computer Example:

Computer.Model.ts

export class computer

{

purpose:string;

cputype:string;

processortype:string;

RAMtype:string;

Brandname:string;

price:number;

}

Computers.component.html

<div class="panel panel-primary">

<h1> List of Computers </h1>

<div>

<table class=table>

<tr \*ngFor="let c of Computers|brandfilterPipe:brandname|purposefilterPipe:searchterm|budgetfilterPipe:searchbudget" style="margin-bottom:10px;">

<td>{{c.purpose}}</td>

<td> {{c.Brandname}}</td>

<td>{{c.price}}</td>

</tr>

</table>

</div>

<div>

Enter the brand which you want

<input type="text" [(ngModel)]="brandname" id=txtbrand #brand>

</div>

<div>

Enter the Purpose for which you need the computer

<input type="text" [(ngModel)]="searchterm" id=txtpurpose #purpose>

</div>

<div>

Enter the range of your budget

<input type="text" [(ngModel)]="searchbudget" id=txtbudget #budget>

</div>

<br/>

<br/><br/>

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

<input type="submit" class="btn btn-primary" value="Suggest" (click)="this.Computers=suggest(this.Computers,brand.value)"/>

<input type="submit" class="btn btn-info" value="Reset" (click)="reset()"/>

</div>

</div>

Computer-file-pipe.ts

import {Pipe,PipeTransform} from '@angular/core';

import {computer} from '../Computer.Model'

@Pipe({

name:'brandfilterPipe'

})

export class brandfilterPipe implements PipeTransform{

transform(CPUs:computer[],brandname:string)

{

if(!CPUs||!brandname)

{

return CPUs;

}

return CPUs.filter((x=>x.Brandname.toLowerCase().indexOf(brandname.toLowerCase())!==-1));

}

}

@Pipe({

name:'purposefilterPipe'

})

export class purposefilterPipe implements PipeTransform{

transform(CPUs:computer[],searchterm:string)

{

if(!CPUs|| !searchterm)

{

return CPUs;

}

return CPUs.filter((x=>x.purpose.toLowerCase().indexOf(searchterm.toLowerCase())!==-1));

}

}

@Pipe({

name:'budgetfilterPipe'

})

export class budgetfilterPipe implements PipeTransform{

transform(CPUs:computer[],searchbudget:number)

{

if(!CPUs|| !searchbudget)

{

return CPUs;

}

return CPUs.filter((x=>x.price<searchbudget));

}

}

App.module.ts

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

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

import {FormsModule} from '@angular/Forms';

import {purposefilterPipe,budgetfilterPipe,brandfilterPipe} from '../app/computers/computer-file-pipe';

import { AppRoutingModule } from './app-routing.module';

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

import { ComputersComponent } from './computers/computers.component';

@NgModule({

declarations: [

AppComponent,

ComputersComponent,

purposefilterPipe,

budgetfilterPipe,

brandfilterPipe

],

imports: [

BrowserModule,

AppRoutingModule,

FormsModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

Computers.component.ts

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

import {computer} from '../Computer.Model'

@Component({

selector: 'app-computers',

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

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

})

export class ComputersComponent implements OnInit {

purpose:string;

budget:number;

searchTerm:string;

searchbudget:number;

searchbrand:string;

Computers:computer[]=

[

{purpose:'General',cputype:'CPU type1',RAMtype:'RAMtype1', processortype:'processortype1',Brandname:'Dell', price:9000 },

{purpose:'Gaming',cputype:'CPU type2',RAMtype:'RAMtype4', processortype:'processortype2',Brandname:'HP', price:12000 },

{purpose:'General',cputype:'CPU type1',RAMtype:'RAMtype3', processortype:'processortype3',Brandname:'Lenovo', price:16000 },

{purpose:'Business',cputype:'CPU type3',RAMtype:'RAMtype2', processortype:'processortype1',Brandname:'Apple', price:88000 },

{purpose:'General',cputype:'CPU type4',RAMtype:'RAMtype2', processortype:'processortype4',Brandname:'Acer', price:94000 },

{purpose:'Gaming',cputype:'CPU type1',RAMtype:'RAMtype2', processortype:'processortype1',Brandname:'Lenovo', price:36000 },

{purpose:'Business',cputype:'CPU type2',RAMtype:'RAMtype1', processortype:'processortype3',Brandname:'HP', price:56000 },

{purpose:'General',cputype:'CPU type3',RAMtype:'RAMtype3', processortype:'processortype4',Brandname:'Acer', price:76000 },

{purpose:'Gaming',cputype:'CPU type1',RAMtype:'RAMtype4', processortype:'processortype2',Brandname:'Lenovo', price:86000 },

{purpose:'Business',cputype:'CPU type2',RAMtype:'RAMtype3', processortype:'processortype4',Brandname:'Dell', price:35000 },

{purpose:'General',cputype:'CPU type4',RAMtype:'RAMtype4', processortype:'processortype4',Brandname:'HP', price:90000 },

]

constructor() { }

ngOnInit() {

console.log(this.Computers)

}

public suggest(Computers:computer[],brand:string):computer[]

{

console.log('button clicked');

console.log(Computers);

// console.log(purpose);

// console.log(budget);

console.log(brand);

return Computers.filter((x=>x.Brandname.toLowerCase().indexOf(brand.toLowerCase())!==-1));

}

public reset(brand:string,purpose:string,budget:number):void{

brand=null;

purpose=null;

budget=0;

}

}

Routing in Angular:

**app.module.ts**

1. import {RouterModule} from '@angular/router';

2. RouterModule.forRoot([

{

path:'test',

component:TestComponent

},

{

path:'sample',

component:SampleComponent

}

**main.component.html**

3. <a routerLink="test"> Click to navigate to test </a>

<hr/>

<a routerLink="sample">Click to navigate to sample </a>

4. <router-outlet></router-outlet>

export class TestComponent implements OnInit {

constructor(private router:Router) { }

ngOnInit() {

}

gosample()

{

this.router.navigate(['/sample']);

}

<input type="button" value="go to sample" (click)="gosample()"/>

<button type="submit" (click)="goback()">Go back</button>

export class SampleComponent implements OnInit {

constructor(private location:Location) { }

ngOnInit() {

}

goback()

{

this.location.back();

}

WebAPI Code:

Employee.cs

public partial class Employee

{

public int eid { get; set; }

public string ename { get; set; }

public Nullable<double> salary { get; set; }

public Nullable<System.DateTime> doj { get; set; }

public string designation { get; set; }

public Nullable<int> deptid { get; set; }

public virtual Department Department { get; set; }

}

}

WebApiConfig.cs

public static void Register(HttpConfiguration config)

{

config.EnableCors(new EnableCorsAttribute("http://localhost:4200", headers: "\*", methods: "\*"));

// Web API configuration and services

// Configure Web API to use only bearer token authentication.

config.SuppressDefaultHostAuthentication();

config.Filters.Add(new HostAuthenticationFilter(OAuthDefaults.AuthenticationType));

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

Global.asax.cs

GlobalConfiguration.Configuration.Formatters.JsonFormatter.SerializerSettings

.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore;

GlobalConfiguration.Configuration.Formatters

.Remove(GlobalConfiguration.Configuration.Formatters.XmlFormatter);

Angular 6

Employees

Employee

Employee.component.html

Employee.component.ts

Employee-List

Employee-list.component.html

Employee-list.component.ts

Employees.component.html

Employees.component.ts

Shared

Employee.Model.ts

Employee.service.ts

Employee.Model.ts

export class Employee {

eid:number;

ename:string;

salary:number;

doj:Date;

designation:string;

deptid:number;

}

Employee.service.ts

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

import { Employee } from './employee.model';

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

import { Observable } from 'rxjs/internal/Observable';

import { getNumberOfCurrencyDigits } from '@angular/common';

@Injectable({

providedIn: 'root'

})

export class EmployeeService {

formData:Employee;

list:Employee[];

readonly rootURL="http://localhost:53981/api"

constructor(private http:HttpClient) { }

postEmployee(formData:Employee)

{

return this.http.post(this.rootURL+'/Employees',formData);

}

refreshList():void

{

this.http.get(this.rootURL+'/Employees')

.toPromise().then(res=>this.list=res as Employee[]);

}

putEmployee(formData:Employee)

{

const headers = new HttpHeaders().set('content-type', 'application/json');

const params = new HttpParams().set('ID', formData.eid.toString());

{

return this.http.put<Employee>(this.rootURL+'/Employees/'+formData.eid,formData,{headers,params});

}

}

deleteEmployee(id:number)

{

return this.http.delete(this.rootURL+'/Employees/'+id);

}

}

Employees-component.html

<div class="Jumbotron">

<h1 class="dipslay-4.text-center">Employee Register</h1>

<hr>

<div class="row">

<div class="col-md-5">

<app-employee></app-employee>

</div>

<div class="col-md-7">

<app-employee-list></app-employee-list>

</div>

</div>

</div>

App.module.ts

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

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

import {FormsModule} from '@angular/forms';

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

import {ToastrModule} from 'ngx-toastr';

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

import { AppRoutingModule } from './app-routing.module';

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

import { EmployeesComponent } from './employees/employees.component';

import { EmployeeComponent } from './employees/employee/employee.component';

import { EmployeeListComponent } from './employees/employee-list/employee-list.component';

import { EmployeeService } from './shared/employee.service';

@NgModule({

declarations: [

AppComponent,

EmployeesComponent,

EmployeeComponent,

EmployeeListComponent

],

imports: [

BrowserModule,

AppRoutingModule,

FormsModule,

HttpClientModule,

BrowserAnimationsModule,

ToastrModule.forRoot()

],

providers: [EmployeeService],

bootstrap: [AppComponent]

})

export class AppModule { }

angular.json

"styles": [

"src/styles.css",

"node\_modules/ngx-toastr/toastr.css"

],

Index.html

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Emplwebapi</title>

<base href="/">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="icon" type="image/x-icon" href="favicon.ico">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

</head>

<body>

<app-root></app-root>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>

</body>

</html>

Employee-list.component.html

<table class="table table-hover">

<tr \*ngFor="let emp of service.list">

<td (click)=Populateform(emp)>[{{emp.eid}}]-{{emp.ename}}</td>

<td (click)=Populateform(emp)>{{emp.salary}}</td>

<td (click)=Populateform(emp)>{{emp.deisgnation}}</td>

<td (click)=Populateform(emp)>{{emp.doj}}</td>

<td (click)=Populateform(emp)>{{emp.deptId}}</td>

<td><button (click)="onDelete(emp.eid)" class="btn btn-sm btn-outine-danger">X</button>

</tr>

</table>

Employee-list.component.ts

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

import { EmployeeService } from '../../shared/employee.service';

import {Employee} from '../../shared/employee.model';

import { ToastrService } from 'ngx-toastr';

@Component({

selector: 'app-employee-list',

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

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

})

export class EmployeeListComponent implements OnInit {

constructor(public service:EmployeeService,public toastr:ToastrService) {

}

ngOnInit() {

this.service.refreshList();

}

Populateform(emp:Employee)

{

this.service.formData=Object.assign({},emp);

}

onDelete(id:number)

{

if(confirm("Are you sure to Delete this record?")){

console.log(id);

this.service.deleteEmployee(id).subscribe(res=>

{

this.service.refreshList();

this.toastr.info("Deleted Successfully","Employee Register");

});

}

}

}

Employee.component.html

<form #form="ngForm" (submit)=OnSubmit(form) autocomplete="off">

<div class="form-group">

<label> Eid :</label>

<input type="hidden" class="form-control" name="eid" required [(ngModel)]="this.service.formData.eid" #name=ngModel>

</div>

<div class="form-group">

<label> Full Name:</label>

<input class="form-control" id="ename" name="ename" [(ngModel)]="this.service.formData.ename" #ename="ngModel">

</div>

<div class="form-group">

<label> Salary:</label>

<input class="form-control" name="salary" [(ngModel)]="this.service.formData.salary" #salary="ngModel">

</div>

<div class="form-group">

<label> Date Of Joining:</label>

<input class="form-control" name="doj" [(ngModel)]="this.service.formData.doj" #doj="ngModel">

</div>

<div class="form-group">

<label> Designation:</label>

<input class="form-control" name="designation" [(ngModel)]="this.service.formData.designation" #designation="ngModel">

</div>

<div class="form-group">

<label> Department ID:</label>

<input class="form-control" name="deptId" [(ngModel)]="this.service.formData.deptId" #deptId="ngModel" >

</div>

<div class="form-group">

<button type="submit" class="btn btn-lg btn-block"> SUBMIT </button>

</div>

</form>

Employee-component.ts

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

import { EmployeeService } from '../../shared/employee.service';

import { NgForm, NgFormSelectorWarning } from '@angular/forms';

import { MockNgModuleResolver } from '@angular/compiler/testing';

import {HttpClient} from '@angular/common/http';

import { Employee } from '../../shared/employee.model';

import { ToastrService } from 'ngx-toastr';

@Component({

selector: 'app-employee',

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

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

})

export class EmployeeComponent implements OnInit {

public empids:number[];

public result:any;

employeeid:number;

emp:Employee;

constructor(private service:EmployeeService,private toastr:ToastrService)

{

this.service.formData={

eid:null,

ename:null,

doj:null,

salary:null,

deptid:null,

designation:null

}

this.result=this.service.list;

// this.result.forEach(element => {

//});

}

// console.log(this.emp.ename);

ngOnInit() {

this.resetForm();

}

OnSubmit(form:NgForm)

{ {debugger}

this.employeeid=form.value.eid;

this.emp=this.service.list.find(e=>e.eid==this.employeeid);

console.log(this.employeeid);

if(this.emp==null)

// if(form.value.eid=this.employeeid)

{

this.InsertRecord(form);

}

else

{

this.UpdateRecord(form);

}

}

InsertRecord(form:NgForm)

{

this.service.postEmployee(form.value).subscribe(res=>{

this.toastr.success("Inserted Successfully","Employee Register");

this.resetForm(form)

this.service.refreshList();

});

}

UpdateRecord(form:NgForm)

{

this.service.putEmployee(form.value).subscribe(res=>{

this.toastr.warning("Updated Successfully","Employee Register");

this.resetForm(form)

this.service.refreshList();

});

}

resetForm(form?:NgForm)

{

if(form!=null)

{

form.resetForm();

this.service.formData={

eid:null,

ename:null,

doj:null,

salary:null,

deptid:null,

designation:null

}

}

}

}

EmployeeProject:

Department.model.ts

export class Department

{

id:number;

name:string;

}

Employee.model.ts

export class Employee

{

id:number;

name:string;

gender:string;

email?:string;

phonenumber?:number;

contactPreference:string;

dateOfBirth:Date;

department:string;

isActive:Boolean;

photoPath?:string;

}

Empreg.component.html

<form #employeeform="ngForm" ngNativeValidate (ngSubmit)="saveEmp(employee)">

<div class="panel panel-primary">

<div class="panel-heading">

<h3 class="panel-title"> Create Employee </h3>

</div>

<div class="panel-body">

<div class="form-group" [class.has-error]="name.invalid && name.touched" [class.has-success]="name.valid" >

<label for="name"> Full Name :</label>

<input required type="text" name="name" [(ngModel)]="employee.name"

#name="ngModel" id="name" class="form-control" >

<span class="help-block" \*ngIf="name.invalid">

Name is Required

</span>

</div>

<!-- <div> touched: {{fullnamecontrol.touched}} </div>

<div> Untouched: {{fullnamecontrol.untouched}} </div>

<div> Dirty: {{fullnamecontrol.dirty}} </div>

<div> Pristine: {{fullnamecontrol.pristine}} </div>

<div> Valid: {{fullnamecontrol.valid}} </div>

<div> Invalid: {{fullnamecontrol.invalid}} </div>

<br/>

<h3> Employee Form level Validations:</h3>

<div> touched: {{employeeform.touched}} </div>

<div> Untouched: {{employeeform.untouched}} </div>

<div> Dirty: {{employeeform.dirty}} </div>

<div> Pristine: {{employeeform.pristine}} </div>

<div> Valid: {{employeeform.valid}} </div>

<div> Invalid: {{employeeform.invalid}} </div> -->

<div class="form-group" [class.has-error]="gender.touched && gender.invalid">

<label class="control-label"> Gender :</label>

<div class="form-control">

<label class="radio-inline">

<input type="radio" value="male" required #gender="ngModel" name="gender" [(ngModel)]="employee.gender" >Male

</label>

<label class="radio-inline">

<input type="radio" value="female" required name="gender" #gender="ngModel" [(ngModel)]="employee.gender" >Female

</label>

<span class="help-block" \*ngIf="gender.touched && gender.invalid">

Gender is required

</span>

</div>

</div>

<div class="form-group" [class.has-error]="contactPreference.touched && contactPreference.invalid">

<label class="control-label"> Contact Preference :</label>

<div class="form-control">

<label class="radio-inline">

<input type="radio" value="pno" [required]="contactPreference.value=='pno'" name="contact" #contactPreference="ngModel" [(ngModel)]="employee.contactPreference">Phone

</label>

<label class="radio-inline">

<input type="radio" value="email" [required]="contactPreference.value=='email'" #contactPreference="ngModel" name="contact" [(ngModel)]="employee.contactPreference">Email

</label>

</div>

<span class="help-block" \*ngIf="contactPreference.touched && contactPreference.invalid">

ContactPreference is required

</span>

</div>

<div class="form-group [class.has-error]=email.invalid && email.touched">

<label for="email" class="control-label"> Email :</label>

<input type="text" name="email" [(ngModel)]="employee.email" #email="ngModel"

class="form-control" required email>

<span class="help-block" \*ngIf="email.errors?.required && email.touched">

Email is Required

</span>

<span class="help-block" \*ngIf="email.errors?.email && email.touched">

Email is Invalid

</span>

</div>

<div class="form-group">

<label class="radio-inline"> PhoneNumber :</label>

<input type="text" name="phno" [(ngModel)]="employee.phonenumber" class="form-control">

</div>

<div class="form-group" [class.has-error]="isActive.touched && isActive.invalid">

<div class="form-control" >

<label class="checkbox-inline control-label" >

<input type="checkbox" name="isActive" [required]="employee.isActive==null" #isActive="ngModel" [(ngModel)]="employee.isActive" > IsActive:

</label>

</div>

<span class="help-block" \*ngIf="isActive.touched && isActive.errors?.required">

IsActive is Required

</span>

</div>

<br/>

<div class="form-control">

<label for="department"> Department :</label>

</div>

<br/>

<div>

<select id="department" name="department" [(ngModel)]="employee.department" class="form-control" >

<!-- <option value="1"> Help Desk </option>

<option value="2"> HR </option>

<option value="3"> Payroll </option>

<option value="4"> IT </option> -->

<option \*ngFor="let dept of departments" [value]="dept.id">

{{dept.name}}

</option>

</select>

</div>

<div class="row">

<div class="form-group col-md-3">

<label for="dateOfBirth"> Date Of Birth :</label>

<input type="text" placement="right" bsDatepicker name="dateOfBirth"

[(ngModel)]="employee.dateOfBirth" id="dateOfBirth" class="form-control">

</div>

</div>

<div class="form-group">

<label for="photoPath"> Photo Path:</label>

<input type="text" name="photoPath" [(ngModel)]="employee.photoPath" id="photoPath" class="form-control">

</div>

<div>

<button type="button" (click)="togglePhotoPreview()" class="btn btn-primary">

{{previewphoto? "Hide" : "Show "}} Preview

</button>

</div>

<div class="form-group">

<img [src]="employee.photoPath" height="200" width="200" \*ngIf="previewphoto"/>

</div>

</div>

<div style="margin-left:500px;margin-bottom: 20px;">

<button type="submit" class="btn btn-primary" [disabled]="employeeform.invalid"> Save </button>

</div>

</div>

</form>

Angular Generated Form Model: {{employeeform.value |json}}

<br/>

Our Employee Model: {{employee |json }}

Empreg.component.ts

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

import{NgForm} from '@angular/forms';

import {Department} from '../empreg/department.model'

import {BsDatepickerConfig} from 'ngx-bootstrap/datepicker'

import {Employee} from '../empreg/Employee.model';

@Component({

selector: 'app-empreg',

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

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

})

export class EmpregComponent implements OnInit {

bsConfig:Partial<BsDatepickerConfig>;

dob:Date=new Date(2018,0,30);

previewphoto=false;

gender="male";

contact="pno";

isActive=true;

department=2;

employee: Employee={

id:null,

name:null,

gender:null,

contactPreference:null,

isActive:null,

phonenumber:null,

photoPath:null,

dateOfBirth:null,

department:null

};

departments: Department[]=[

{id:1,name:"HelpDesk"},

{id:2,name:"HR"},

{id:3,name:"IT"},

{id:4,name:"Payroll"},

{id:5,name:"admin"}

]

constructor() {

this.datePickerConfig=Object.assign({},{containerClass:'theme-dark-blue',

showWeekNumbers:false,

minDate:new Date(2018,0,1),

maxDate: new Date(2018,11,1),

dateInputFormat:'DD/MM/YYYY'

});

}

togglePhotoPreview()

{

this.previewphoto=!this.previewphoto;

}

ngOnInit() {

}

saveEmp(newEmployee:Employee):void

{

console.log(newEmployee);

}

}

App-routing.module.ts

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

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

import {EmpregComponent} from './empreg/empreg.component';

import {EmplistComponent} from './emplist/emplist.component';

const routes: Routes = [

{

path:'Create',

component:EmpregComponent,

},

{

path:'List',

component:EmplistComponent

}

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

app.component.html

<div>

<nav class="navbar navbar-inverse">

<div class="container-fluid">

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed" (click)="isCollapsed=!isCollapsed">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<!-- <a class="navbar-brand" href=#>Employee Management </a> -->

</div>

<div id="navbar" class="navbar navbar-collapse collapse" [collapse]="isCollapsed">

<ul class="nav navbar nav">

<li> <a routerLink="List">List</a></li>

<li> <a routerLink="Create">Create New Employee</a></li>

</ul>

<ul class="nav navbar-nav navbar-right"></ul>

</div>

</div>

</nav>

<router-outlet></router-outlet>

App.module.ts

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

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

import {FormsModule} from '@angular/forms';

import {CollapseModule} from 'ngx-bootstrap';

import { AppRoutingModule } from './app-routing.module';

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

import { EmpregComponent } from './empreg/empreg.component';

import { EmplistComponent } from './emplist/emplist.component';

@NgModule({

declarations: [

AppComponent,

EmpregComponent,

EmplistComponent

],

imports: [

BrowserModule,

AppRoutingModule,

FormsModule

],

providers: [],

bootstrap: [AppComponent],

})

export class AppModule { }

Emplist.component.html

<div class="panel panel-primary" \*ngFor="let employee of employees">

<div id="maindiv">

<div class="panel-heading">

<h3 class="panel-title">{{employee.name}}</h3>

</div>

<div class="panel-body">

<div class="col-xs-12">

<div class="row">

<div class="col-md-2">

<img height="100" width="100" [src]="employee.photoPath">

</div>

<div class="col-md-10">

<div class="row">

<div class="col-xs-6">

Gender

</div>

<div class="col-xs-6">

:{{employee.gender}}

</div>

</div>

<div class="row">

<div class="col-xs-6">

contactPreference

</div>

<div class="col-xs-6">

:{{employee.contactPreference}}

</div>

</div>

<div class="row" \*ngIf="employee.email">

<div class="col-xs-6">

Email

</div>

<div class="col-xs-6">

:{{employee.email}}

</div>

</div>

<div class="row" [hidden]="!employee.phonenumber">

<div class="col-xs-6">

Phone Number

</div>

<div class="col-xs-6">

:{{employee.phonenumber}}

</div>

</div>

<div class="row">

<div class="col-xs-6">

dateOfBirth

</div>

<div class="col-xs-6">

:{{employee.dateOfBirth | date}}

</div>

</div>

<div class="row">

<div class="col-xs-6">

department

</div>

<div class="col-xs-6">

:{{employee.department}}

</div>

</div>

<div class="row">

<div class="col-xs-6">

IsActive

</div>

<div class="col-xs-6">

:{{employee.isActive}}

</div>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

Emplist.component.ts

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

import {Employee} from '../empreg/Employee.model';

@Component({

selector: 'app-emplist',

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

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

})

export class EmplistComponent implements OnInit {

employees:Employee[]=[

{

id:1,

name:'Steve',

gender:'Male',

contactPreference:'Email',

email:'steve@gmail.com' ,

dateOfBirth:new Date('01/03/2010'),

department:'Apple',

isActive:true,

photoPath:'assets/stevejobs.jpeg'

},

{

id:2,

name:'Bill Gates',

gender:'Male',

contactPreference:'Phone',

phonenumber:9850048765 ,

dateOfBirth:new Date('01/03/2012'),

department:'Microsoft',

isActive:true,

photoPath:'assets/billgates.jpeg',

},

{

id:3,

name:'Sundar',

gender:'Male',

contactPreference:'Email',

email:'Sundar@gmail.com' ,

dateOfBirth:new Date('05/07/2010'),

department:'Google',

isActive:true,

photoPath:'assets/sundar.jpeg',

}

]

constructor() {

}

ngOnInit() {

}

}