ANGULAR

Basics
Reference:agularyoutube/basics-app
Link : https://youtu.be/RmKnHsUGpPY?si=sk2-rhY0smo9OykI

Component selection

Selector can be found in component.ts file.
Components can be selected in four ways

Html tag

Selector : ‘ app-comp-selector’, then in html file we can use <app-comp-selector></app-comp-selector>

Html attribute

Selector : ‘ [app-comp-selector]’, then in html file we can use <div app-comp-selector>

Html css class

Selector : ‘ .app-comp-selector’, then in html file we can use

Id

Selector : ‘ #app-comp-selector’, then in html file we can use

Example

Ts file

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

@Component({
 // selector: 'app-comp-selector', // html  tag selector
 // selector: '[app-comp-selector]', // html attribute
 // selector: '.app-comp-selector', // html css class
  selector: '#app-comp-selector', // html css id
  templateUrl: './comp-selector.component.html',
  styleUrl: './comp-selector.component.css'
})
export class CompSelectorComponent {

}
```

Html file

```
<!-- <app-comp-selector></app-comp-selector>--> <!--- html tag selector -->
<!-- <div 'app-comp-selector'></div> --> <!--- html attribute selector -->
<!-- <div class='app-comp-selector'></div> --> <!--- html class selector -->
<div id='app-comp-selector'></div> <!--- html id selector -->
```

Data binding

Communication between template class (ts file) and view template (html file)

2 types

One way data binding -
Data can move from ts to html file or html to ts file

From ts to html

String interpolation
Ts file - define data
Html file {{ data }}

Example:
Ts file

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

@Component({
  selector: 'app-one-way-data-binding',
  templateUrl: './one-way-data-binding.component.html',
  styleUrl: './one-way-data-binding.component.css'
})
export class OneWayDataBindingComponent {
// srting interpolation
name:string = 'John';
product = {
name: 'iPhone',
price: 50000,
discount: 8.5,
inStock : 0
}
getDiscountedPrice(){
  return this.product.price - (this.product.price * this.product.discount  / 100);
}
}
```

Html file

```
<h4>String interpolation</h4>
<p>The name is {{ name }}</p>
<p>Product: {{ product.name }}</p>
<p>Price : {{ product.price }}</p>
<p>other options</p>
<p>Price with dollor symbol: {{ "$" + product.price }}</p>
<p>
  Price after discount :
  {{ product.price - (product.price * product.discount) / 100 }}
</p>
<p>Discounted price using method: {{ getDiscountedPrice() }}</p>
<p>
  {{
    product.inStock > 0
      ? "Only " + product.inStock + " left in stock"
      : "Not in Stock"
  }}
</p>
```

Property binding

Ts file - define data
Html file [property] = data

Html file

```
<button [disabled]="!(product.inStock > 0)">Buy Now</button>
```

Note:
Attribute binding - for this we write as below
[attr.attributename] = data

Output:

Component selector

comp-selector works!

One way data binding (string interpolation)

String interpolation

The name is John

Product: iPhone

Price : 50000

other options

Price with dollor symbol: $50000

Price after discount : 45750

Discounted price using method: 45750

Only 10 left in stock

Buy Now

Another example for property binding

John

Another example for property binding using bind

John

From html to ts

Event binding
(data) = “ experssion ”

Two way data binding

Data can move in both directions at the same time
If we change data in ts file it reflects in html file and vice versa
Two way data binding is a combination of property binding and event binding

[ngModel]

Create a new component two-way-data-binding

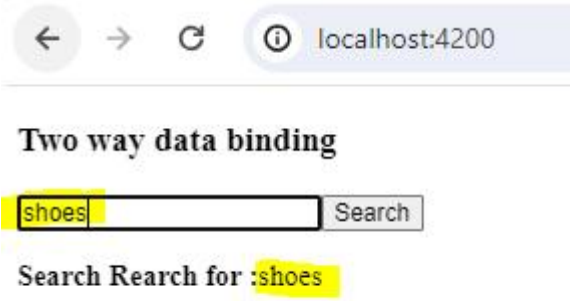
two-way-data-binding.component.html

```
<div>
    <input type="text" [value]=”searchText"
    (input)= ”onUpdateSearchText($event)”>
    <button >Search</button>
</div>
<div>
    <p><strong>Search Rearch for :</strong>{{searchText}}</p>
</div>
```

two-way-data-binding.component.ts

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

@Component({
  selector: 'app-two-way-data-binding',
  templateUrl: './two-way-data-binding.component.html',
  styleUrl: './two-way-data-binding.component.css'
})
export class TwoWayDataBindingComponent {
searchText = 'Means Wear';
onUpdateSearchText(event:any){
  this.searchText = event.target.value;
}
}
```

Output:



Actual way of two way binding

We can use ngModel to achieve 2 way data binding as follows. In order to use mgModel we should inport FormsModule in app.module.ts file

Ts file is same as above - we can delete onUpdateSearchText function
Html file

```
<div>
    <input type="text" [(ngModel)]="searchText">
    <button >Search</button>
```

```
</div>
<div>
    <p><strong>Search Rearch for :</strong>{{searchText}}</p>
</div>
```

Directives

Directive is an instruction to DOM - manipulate DOM, change behaviour, add/remove DOM elements
Component directive - is an angular component which has a template
Attribute directive - used to change the behaviour or appearance of DOM - ngStyle and ngClass
Structural directive - used to add or remove DOM element - ngIf, ngFor, ngSwitch, it starts with *

ngIf

It is used to completely add or reove a DO element based on a given condition

Create a new component ng-if

Html file

```
<div>
    <input type="text" [(ngModel)]="searchText">
    <button >Search</button>
</div>
<div>
    <p *ngIf = "searchText != ''">
      <strong>Search Rearch for : </strong>{{searchText}}</p>
</div>
```

Ts file

```
@Component({
  selector: 'app-directive-ngif',
  templateUrl: './directive-ngif.component.html',
  styleUrl: './directive-ngif.component.css'
})
export class DirectiveNgifComponent {
  searchText='';
}
```

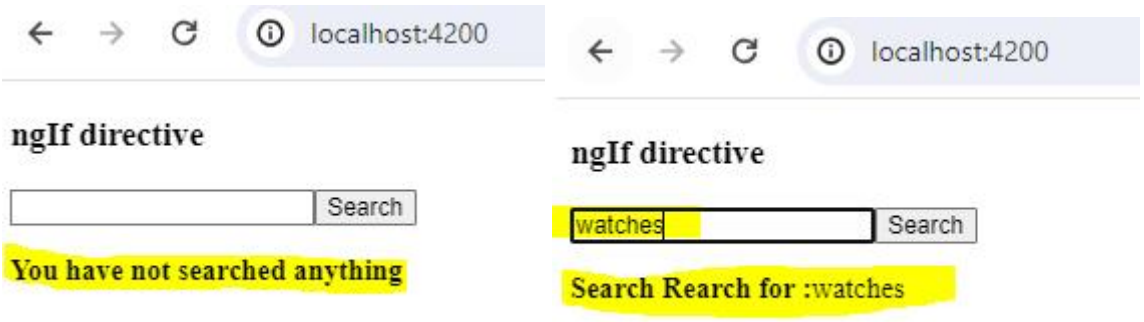
In the above example <p> tag will be rendered only if we enter something in search box

Adding else part

Html file

```
<div>
  <input type="text" [(ngModel)]="searchText" />
  <button>Search</button>
</div>
<div>
  <p *ngIf="searchText != ''; else elsepart">
    <strong>Search Rearch for :</strong>{{ searchText }}
  </p>
  <ng-template #elsepart
    ><p><strong>You have not searched anything</strong></p></ng-template
  >
</div>
```

Output:



ngStyle

It is an attribute directive which allows us to set many inline styles of an html element using expressions. Advantage of this is we can apply styles dynamically

Ts file

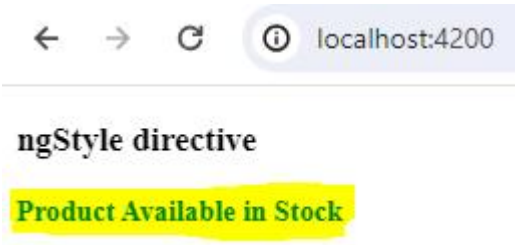
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-directive-ngstyle',
  templateUrl: './directive-ngstyle.component.html',
  styleUrl: './directive-ngstyle.component.css'
})
export class DirectiveNgstyleComponent {
productInStock = 10;
}
```

Html file

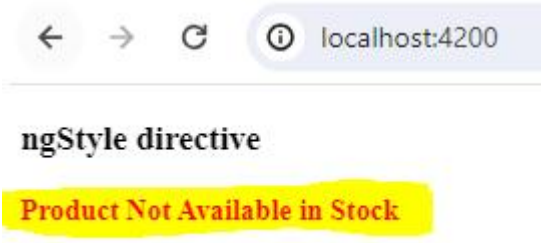
```
<div [ngStyle]="{fontWeight:'bold', color: productInStock? 'green':'red'}">
    <p>Product {{productInStock?'Available in Stock':'Not Available in
Stock'}}</p>
</div>
```

Output:

When productInStock is > 0



When productInStock is = 0



ngClass

It is an attribute directive which allows us to add o remove css classes to or from an html element dynamically based on some typescript expression

Ts file

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

@Component({
  selector: 'app-directive-ngclass',
  templateUrl: './directive-ngclass.component.html',
  styleUrl: './directive-ngclass.component.css'
})
export class DirectiveNgclassComponent {
  searchText ='';
}
```

Css file

```
.btn{
    color: red;
}
.search-btn{
    background-color: blue;
    color: azure;
}
```

Html file

```
<div>
  <input type="text" [(ngModel)]="searchText" />
  <button
```

```
    [ngClass]="{ btn: true, 'search-btn': searchText }"
    [disabled]="!searchText"
  >
    Search
  </button>
</div>
<div>
  <p><strong>Search Rearch for :</strong>{{ searchText }}</p>
</div>
```

In the above example css class btn will we always applied and css class search-btn will be applied when ever there is some value in searchText

Output

@Input and custom property binding

Create a new component product-list
Create a new component product under product-list
In this example product-list is parent and product is child
Change “strict” to false in tsconfig.json file

Product-list.ts file

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

@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html',
  styleUrl: './product-list.component.css',
})
export class ProductListComponent {
  productsList = [
    { id: 1, name: 'Redmi', desc: 'it is a phone' },
    { id: 2, name: 'LG', desc: 'it is a TV' },
    { id: 3, name: 'Voltas', desc: 'it is an AC' },
  ];
```

```
}
```

Product-list.html file

```
<h3>Product List</h3>
<app-product *ngFor="let prod of productsList" [product]="prod"></app-product>
```

Product.ts file

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

@Component({
  selector: 'app-product',
  templateUrl: './product.component.html',
  styleUrl: './product.component.css'
})
export class ProductComponent {
  @Input()
  product:{
    id:number,
    name:string,
    desc:string
  };
}
```

Product.html file

```
<p>{{product.name}}</p>
<p>{{product.desc}}</p>
```

In the above example we created a variable product in child component (product) .ts file using @Input() decorator. By doing this we can use product as a custom property and use it in parent component and assign value to it. By this way we can send data from parent to child.

@Output() &custom event binding

It is used to send the data from child to parent.

Create a component student-list
Create a component add-student inside student-list

Add-student.html file

```
<p>add-student works!( child component of student-list)</p>
<input [(ngModel)]="studentName">
<button (click) = "onStudentAdded()">Add Student</button>
<hr>
```

Add-student.ts file

```
import { Component, EventEmitter, Output } from '@angular/core';
```

```
@Component({
  selector: 'app-add-student',
  templateUrl: './add-student.component.html',
  styleUrl: './add-student.component.css'
})
export class AddStudentComponent {
studentName = '';
@Output() studentAdded = new EventEmitter<string>();
onStudentAdded(){
 this.studentAdded.emit(this.studentName)
}
}
```

In the above ts file studentName will receive the value from html file via two way data binding. This value we want to send to the parent component. To achieve this we use @Output() decorator and make studentAdded as a custom event emitter. So whenever the Add Student button is clicked onStudentAdded() function gets executed and will emit new event and pass the studentName value.
Any variable decorated with @Output() can be used as an attribute along with the respective component selector.

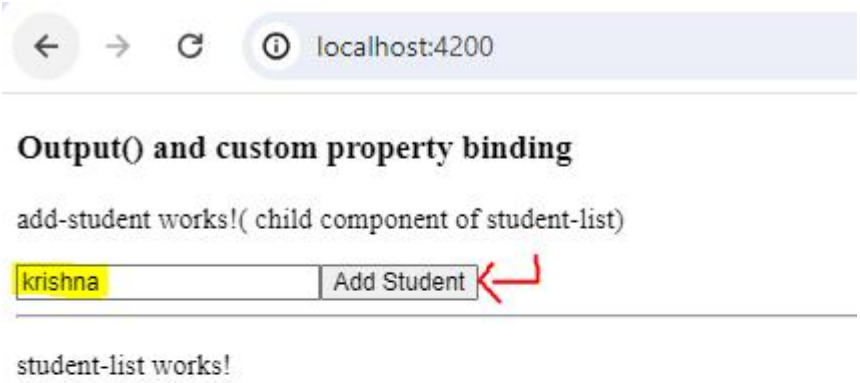
Student-list.html file

```
<app-add-student
  (studentAdded)="onStudentAddedParent($event)"
></app-add-student>
<p>student-list works!</p>
```

Student-list.ts file

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

@Component({
  selector: 'app-student-list',
  templateUrl: './student-list.component.html',
  styleUrl: './student-list.component.css'
})
export class StudentListComponent {
onStudentAddedParent(event:string){
  console.log("Data received from Child in Parent : " +event);
}
}
```

Output:



If we want to show the student in the ui then

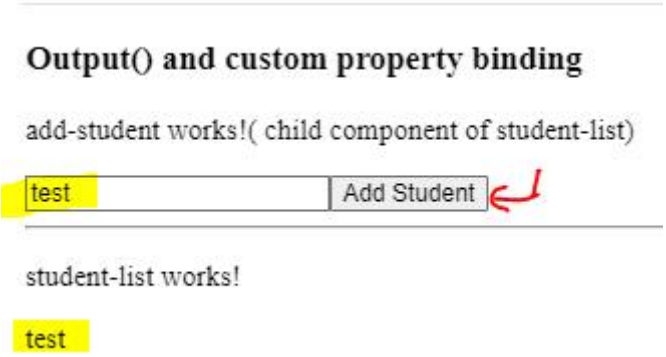
Student-list.ts file

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

@Component({
  selector: 'app-student-list',
  templateUrl: './student-list.component.html',
  styleUrl: './student-list.component.css'
})
export class StudentListComponent {
 studentName = '';
onStudentAddedParent(event:string){
  this.studentName = event;
}
}
```

Student-list.html file

```
<app-add-student
  (studentAdded)="onStudentAddedParent($event)"
></app-add-student>
<p>student-list works!</p>
<p>{{studentName}}</p>
```

Output:



Non-related component communication

In order to send data across non related components then we need to use both @Input and @Output decorators

Create a new component users-list
Create two more components add-user and user inside users-list
Data will be sent from add-user to users-list and from users-list to user component. Since we want to send from add-user to users-list, we use @Output() decorator in add-user component. From user-list we want to send data to user (in other words, user component wants to receive the data we use @Input() decorator in user component)

Add-user.html

```
<p>add-user component (child of users-list)</p>
<input type="text" [(ngModel)]= "userName">
<button (click)="onUserAdded()">Add User</button>
<hr>
```

Add-user.ts file

```
import { Component, EventEmitter, Output } from '@angular/core';

@Component({
  selector: 'app-add-user',
  templateUrl: './add-user.component.html',
  styleUrl: './add-user.component.css'
})
export class AddUserComponent {
  userName = '';
  @Output() userAdded = new EventEmitter<string>();
  onUserAdded(){
   this.userAdded.emit(this.userName);
  }
}
```

Users-list.ts file

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

@Component({
  selector: 'app-users-list',
  templateUrl: './users-list.component.html',
  styleUrl: './users-list.component.css'
})
export class UsersListComponent {
  userList = [];
  onUserAddedParent(event:string){
    this.userList.push(event);
  }
}
```

Users-list.html file

```
<app-add-user (userAdded)="onUserAddedParent($event)"></app-add-user>
<p>users-list works!(Parent of add-user and user)</p>
<hr>
<app-user *ngFor="let user of userList" [user]="user"></app-user>
```

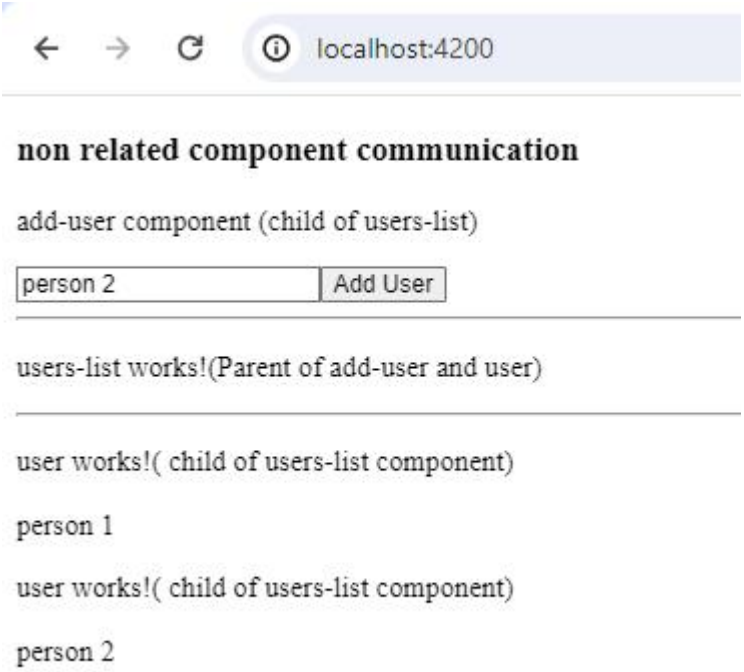
User.ts file

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

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css'
})
export class UserComponent {
@Input() user = '';
}
```

User.html

```
<p>user works!( child of users-list component)</p>
<p>{{user}}</p>
```

Output:



Template reference variable

It is a variable which stores a reference to a DOM element, component or directive on which it is used

Create a component template-reference-variable

Html file

```
<p>template-reference-variable works!</p>
<input placeholder="Type here to Search" #searchInput>
<button (click)="onUpdateSearchText(searchInput)">Search</button>
<p *ngIf="searchText"> You are searching for {{searchText}}</p>
```

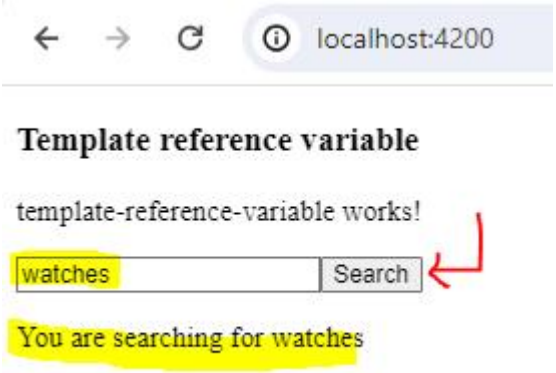
Here #searchInput is the template reference variable which will hold the value of < input > tag

Ts file

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

@Component({
  selector: 'app-template-reference-variable',
  templateUrl: './template-reference-variable.component.html',
  styleUrl: './template-reference-variable.component.css'
})
export class TemplateReferenceVariableComponent {
  searchText = '';
onUpdateSearchText(inputEl:HTMLInputElement){
```

```
  this.searchText = inputEl.value;
}
}
```

Output:



Send Data from Parent to Child Component using Custom Properties with @Input() in Angular.

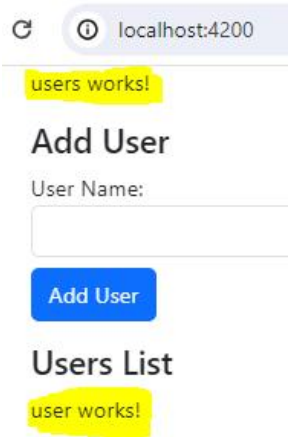
Create new project (ng new notes-project --no-standalone)
Install bootstrap (npm install bootstrap)
Create new components - user and users
For this example users will be parent and user will be child

Setup app.component.html file as below

```
<div class="container">
  <div class="row">
    <div class="col-sm-12">
      <app-users></app-users>
    </div>
  </div>
</div>
```

Setup the users.component.html file as below

```
<p>users works!</p>
<div class="form-group">
    <h4>Add User</h4>
    <label>User Name:</label>
    <input type="text" class="form-control">
    <div class="mt-2">
        <button class="btn btn-primary">Add User</button>
    </div>
</div>
<div class="mt-3">
    <h4>Users List</h4>
    <app-user></app-user>
</div>
```



In order to capture the data user enters we will use ngModel and 2 way databinding with a variable (userName) which is defined in users.component.ts file in the input tag.

Users.component.html file

```
<h4>Add User</h4>
   <label>User Name:</label>
   <input type="text" class="form-control" [(ngModel)]="userName">
   <div class="mt-2">
      <button class="btn btn-primary">Add User</button>
   </div>
```

Users.component.ts file

```
export class UsersComponent implements OnInit {

  userName:string = '';
  constructor(){}
  ngOnInit(){}
}
```

To use ngModel, we need to import FormsModule from ‘angular/forms’ in app.module.ts and it should be added to imports array

App.module.ts file

```
import { FormsModule } from '@angular/forms';
@NgModule({
  declarations: [
    AppComponent,
    UsersComponent,
    UserComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

Next step:
When user enters some name and clicks on Add User button the entered name will get added to a usersList defined in the users.component.ts file

users.component.ts file

Define a variable usersList
Define a method (onUserAdded) that will be called on the click of Add User button

```
export class UsersComponent implements OnInit {

  userName:string = '';
  usersList = [];
  constructor(){}
  ngOnInit(){}
  onUserAdded(){
    this.usersList.push(this.userName);
  }
}
```

users.component.html file

```
<div class="form-group">
    <h4>Add User</h4>
    <label>User Name:</label>
    <input type="text" class="form-control" [(ngModel)]="userName">
    <div class="mt-2">
        <button class="btn btn-primary" (click)="onUserAdded()">Add User</button>
    </div>
</div>
```

Now we can use *ngFor directive in order to display user component for each user added through the button

```
<div class="mt-3">
    <h4>Users List</h4>
    <app-user *ngFor="let user of usersList"></app-user>
</div>
```

Next Step:

Each user name should be passed to user component.
To achieve this we will use property binding. We update users.component.html file as follows

```
<div class="mt-3">
    <h4>Users List</h4>
    <app-user *ngFor="let user of usersList" [userName]="user"></app-user>
</div>
```

Now the userName property should be declared in the user.component.ts file
To achieve this we should use @Input() annotation which is imported from '@angular/core'

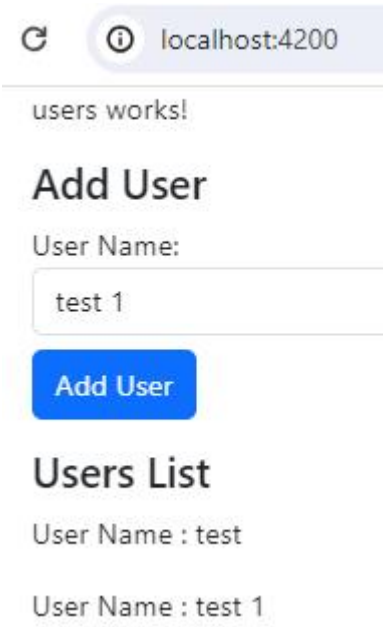
user.component.ts

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

```
@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css'
})
export class UserComponent implements OnInit {
  @Input()userName  = '';
  constructor(){}
  ngOnInit(){}
}
```

Update the user.component.html file as follows
user.component.html

```
<p>User Name : {{userName}}</p>
```



Note: if we want to use different property name in parent html file and child ts file we can do so as follows
users.component.html file as follows

```
<div class="mt-3">
   <h4>Users List</h4>
   <app-user *ngFor="let user of usersList" [user]="user"></app-user>
</div>
```

user.component.ts

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

@Component({
```

```
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css'
})
export class UserComponent implements OnInit {
  @Input('user')userName  = '';
  constructor(){}
  ngOnInit(){}
}
```

Send Data from Child to Parent Component. Binding to Custom Events using @Output in Angular.

Situation: we will create a new component add-user,so the user name should be sent from add-user component to users component from there to user component

Create a new component add-user (ng g c add-user)

Remove the following code from users.component.html file and paste it in add-user.component.html file

```
<div class="form-group">
    <h4>Add User</h4>
    <label>User Name:</label>
    <input type="text" class="form-control" [(ngModel)]="userName">
    <div class="mt-2">
        <button class="btn btn-primary" (click)="onUserAdded()">Add User</button>
    </div>
</div>
```

Also we can remove userName declaration and onUserAdded method from users.component.ts file

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

@Component({
  selector: 'app-users',
  templateUrl: './users.component.html',
  styleUrl: './users.component.css'
})
export class UsersComponent implements OnInit {
 // userName:string = '';
  usersList = [];
  constructor(){}
  ngOnInit(){}
  // onUserAdded(){
```

```
  //   this.usersList.push(this.userName);
  // }
}
```

And add the following code to users.component.html file

```
<p>users works!</p>
<app-add-user></app-add-user>

<div class="mt-3">
    <h4>Users List</h4>
    <app-user *ngFor="let user of usersList" [user]="user"></app-user>
</div>
```

Declare userName in add-user.component.ts file

Define the method onUserAdded method
Inside this method what ever the name we enter it to be added to usersList which is declared in the users.component.ts file
That means we need to send the data from child to parent (I.e add-user to users component)
This is achieved through event, we need to emmit an event

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

@Component({
  selector: 'app-add-user',
  templateUrl: './add-user.component.html',
  styleUrl: './add-user.component.css'
})
export class AddUserComponent implements OnInit {
  userName : string ;
  @Output() userAdded = new EventEmitter<string>(); // Output is used to indicate we are sending the data to the parent, userAdded is a custom event which we can use it in add-user component ( like (click) we can now use (userAdded)
  constructor(){}
  ngOnInit(){}
  onUserAdded(){
          this.userAdded.emit(this.userName);
          } // to be implemented later
}
```

Now update users.component.html file as below

```
<p>users works!</p>
<app-add-user (userAdded)="onUserAddedParent($event)"></app-add-user>

<div class="mt-3">
```

```
    <h4>Users List</h4>
    <app-user *ngFor="let user of usersList" [user]="user"></app-user>
</div>
```

In users.component.ts file, Define the method onUserAddedParent() and write the logic to add the userName to usersList

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

@Component({
  selector: 'app-users',
  templateUrl: './users.component.html',
  styleUrl: './users.component.css'
})
export class UsersComponent implements OnInit {
 // userName:string = '';
  usersList = [];
  constructor(){}
  ngOnInit(){}
  // onUserAdded(){
  //   this.usersList.push(this.userName);
  // }
  onUserAddedParent(event:string){
    this.usersList.push(event);
  }
}
```

24. Understand View Encapsulation in Angular. Difference between Emulated, None, and Shadow Dom.

Using Local References in Angular. Access the HTML Element in the Typescript file.

Change the add-user.component.html file as below

```
<p>add-user works!</p>
<div class="form-group">
    <h4>Add User</h4>
    <label>User Name:</label>
    <!-- <input type="text" class="form-control" [(ngModel)]="userName"> -->
    <input type="text" class="form-control" #userName>
    <div class="mt-2">
        <button class="btn btn-primary" (click)="onUserAdded(userName)">Add
User</button>
    </div>
</div>
```

Change add-user.component.ts file as below

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

@Component({
  selector: 'app-add-user',
  templateUrl: './add-user.component.html',
  styleUrl: './add-user.component.css'
})
export class AddUserComponent implements OnInit {
 // userName : string ;
  @Output() userAdded = new EventEmitter<string>();
  constructor(){}
  ngOnInit(){}
  // onUserAdded(){
  //   this.userAdded.emit(this.userName);
  // }
  onUserAdded(input: HTMLInputElement){
    this.userAdded.emit(input.value);
  }
}
```

Access HTML Elements in The DOM & Template with @ViewChild and the type ElementRef in Angular.

This is another approach to the above
Update add-user.component.html as below

```
<p>add-user works!</p>
<div class="form-group">
    <h4>Add User</h4>
    <label>User Name:</label>
    <!-- <input type="text" class="form-control" [(ngModel)]="userName"> -->
    <input type="text" class="form-control" #userName>
    <div class="mt-2">
        <!-- <button class="btn btn-primary" (click)="onUserAdded(userName)">Add
User</button> -->
        <button class="btn btn-primary" (click)="onUserAdded()">Add User</button>
    </div>
</div>
```

Update add-user.component.ts file as below

@Viewchild and ElementRef should be imported from angular/core

```
import { Component, ElementRef, EventEmitter, OnInit, Output, ViewChild } from
'@angular/core';

@Component({
  selector: 'app-add-user',
```

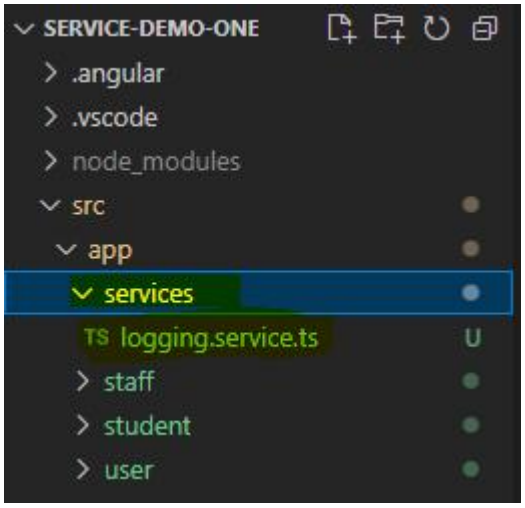
```
  templateUrl: './add-user.component.html',
  styleUrl: './add-user.component.css'
})
export class AddUserComponent implements OnInit {
  @ViewChild('userName') userInput: ElementRef;
 // userName : string ;
  @Output() userAdded = new EventEmitter<string>();
  constructor(){}
  ngOnInit(){}
  // onUserAdded(){
  //   this.userAdded.emit(this.userName);
  // }
  // onUserAdded(input: HTMLInputElement){
  //   this.userAdded.emit(input.value);
  // }
  onUserAdded(){
    this.userAdded.emit(this.userInput.nativeElement.value);
  }
}
```

27. Projecting the HTML Content written between the component using ng-content in Angular.

28. Check ngOnChanges, ngOnInit, and Constructor of the angular component life cycle with example

Services in Angular
Ref: Service-demo-one
Create a new project
Install bootstrap
Create 3 new components (user, staff, student)
If some code should be used in many components then such code can be written as a service and can be reused in many components

Create a folder (services) inside app folder of the project
Create a new file (logging.service.ts) inside services folder



Logging.service.ts is a simple class that contains the code which will be reused

```
export class LoggingService{
    logToconsole(type:string){
        console.log("Button Clicked :" + type);
    }
}
```

In all the three components (user, staff and student) we will have a button and on click of the button we need to output the message to the console using the above service method

Update all three component.html file as below

```
<p>user works!</p>
<button class="btn btn-primary" (click)="onUserClick()">User Clicked</button>
```

```
<p>student works!</p>
<button class="btn btn-primary" (click)="onStudentClick()">Student Clicked</button>
```

```
<p>staff works!</p>
<button class="btn btn-primary" (click)="onStaffClick()">Staff Clicked</button>
```

Now component.ts file of all there components should be updated such that the service method can be invoked
To do so, service should be injected through constructor as follows
And the click method should be implemented through which service method will be invoked

```
import { Component } from '@angular/core';
import { LoggingService } from '../services/logging.service';

@Component({
  selector: 'app-staff',
  templateUrl: './staff.component.html',
  styleUrl: './staff.component.css',
  providers:[LoggingService]
})
export class StaffComponent {
constructor(private loggingService:LoggingService){}
onStaffClick(){
  this.loggingService.logToconsole('Staff');
}
}
```

The above changes should be done in other two component.ts files also

```
import { Component } from '@angular/core';
import { LoggingService } from '../services/logging.service';

@Component({
  selector: 'app-student',
  templateUrl: './student.component.html',
  styleUrl: './student.component.css',
  providers:[LoggingService]
})
export class StudentComponent {
  constructor(private loggingService:LoggingService){}
  onStudentClick(){
    this.loggingService.logToconsole('Student');
  }
```

```
}
```

```
import { Component } from '@angular/core';
import { LoggingService } from '../services/logging.service';

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css',
  providers:[LoggingService]
})
export class UserComponent {
  constructor(private loggingService:LoggingService){
  }
  onUserClick(){
    this.loggingService.logToconsole('User');
  }
}
```

Run the app

42. Create New Service and use the service as Data in Angular. Pass Data from service to Components

Create new project (service-demo-two)
Add bootstrap
Create components ==> add-user and user
Create a folder inside app ==> services
Create user.service.ts file inside services (can also use command ng generate service services/user ==> in this case services folder need not be created manually)

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

@Injectable({
  providedIn: 'root'
}) //this code will not be there if we create manually
export class UserService {
  users = [
    { name:'krishna', status:'active'},
  {name:'krishna 1', status:'active' },
  {name:'krishna 2', status:'active'}
]
  constructor() { }
}
```

This service we want to use it in app.component.ts, so import this and add providers, inject it through constructor
Create an array users to store the list of users that will be received from user.service
In ngOnInit() we will write the code to get the users from user.service and store it in users array

```
import { Component, OnInit } from '@angular/core';
import { UserService } from './services/user.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css',
  providers:[UserService]
})
export class AppComponent implements OnInit {
  title = 'service-demo-two';
  users:{ name:string; status:string}[]=[];
  constructor( private userService:UserService){}
  ngOnInit(){
    this.users = this.userService.users;
  }
}
```

Update app.component.html file as to show all the users as below

```
<div class="container mt-3">
  <div class="row">
    <div class="col-md-12">
      <h4>Services</h4>
      <hr>
      <app-user *ngFor="let user of users" [user]="user">
      </app-user>
    </div>
  </div>
</div>
```

Now define the property user in user.component.ts file using @Input decorator

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

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css'
})
export class UserComponent {
@Input() user:{name:string;status:string};
```

```
}
```

Update user.component.html file as below

```
<p>User Name: {{user.name}} status: {{user.status}}</p>
```

Update the service file by adding a new method to add a new user as follows
User.service.ts file

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

@Injectable({
  providedIn: 'root'
})
export class UserService {
  users = [
    { name:'krishna', status:'active'},
  {name:'krishna 1', status:'active' },
  {name:'krishna 2', status:'active'}
]
  constructor() { }
  addUser(name:string, status:string){
    this.users.push({name, status});
  }
}
```

Update the add-user.component.html file as follows

```
<p>add-user works!</p>
<div class="row">
    <div class="col-md-12">
        <div class="form-group">
            <label>User Name:</label>
            <input class="form-control" type="text" [(ngModel)]="userName">

        </div>
        <div>
            <button class="btn btn-primary" (click)=”onAddUser()”>Add User</button>
        </div>
    </div>
</div>
```

Note: inorder to work with ngModel we need to import FormsModule from @angular/forms and add it to imports array in app.module.ts file

Update add-user.component.ts file

Declare the variable userName
Define the method onAddUser()
With in this method we need to use the service method addUser()
For this import and inject user.Service through constructor

```
import { Component } from '@angular/core';
import { UserService } from '../services/user.service';

@Component({
  selector: 'app-add-user',
  templateUrl: './add-user.component.html',
  styleUrl: './add-user.component.css',
  providers:[UserService]
})
export class AddUserComponent {
  userName:string='';
  constructor(private userService:UserService){}
  onAddUser(){
    this.userService.addUser(this.userName,'Active');
  }
}
```

Update app.component.html file to render add-user component

```
<div class="container mt-3">
  <div class="row">
    <div class="col-md-12">
      <h4>Services</h4>
      <hr>
      <app-add-user></app-add-user>
    </div>
  </div>
  <div class="row">
    <div class="col-md-12">
      <app-user *ngFor="let user of users" [user]="user">
      </app-user>
    </div>
  </div>
</div>
```

Note: we can not see the newly added user yet

In user.service file we add a new method to update the status

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

@Injectable({
```

```
  providedIn: 'root'
})
export class UserService {
  users = [
    { name:'krishna', status:'active'},
  {name:'krishna 1', status:'active' },
  {name:'krishna 2', status:'active'}
]
  constructor() { }
  addUser(name:string, status:string){
    this.users.push({name, status});
  }
  updateStatus(id:number, status:string){
    this.users[id].status = status;
  }
}
```

Update the user.component.html file as follows

```
<p>User Name: {{user.name}} status: {{user.status}}</p>
<span>
    <button class="btn btn-sm btn-primary" (click)="onUpdateStatus('active')">Set
Active</button>
    <button class="btn btn-sm btn-primary" (click)="onUpdateStatus('inactive')">Set In-
active</button>
</span>
```

Inorder to update the status we need the id of the user, so update the app.component.html file as follows

```
<div class="container mt-3">
  <div class="row">
    <div class="col-md-12">
      <h4>Services</h4>
      <hr>
      <app-add-user></app-add-user>
    </div>
  </div>
  <div class="row">
    <div class="col-md-12">
      <app-user *ngFor="let user of users; let i = index" [user]="user" [id]="i">
      </app-user>
    </div>
  </div>
</div>
```

Update user.component.ts file as follows

Define the id through @Input()
Inject user.service through constructor
Define the method onUpdateStatus and invoke updateStatus method defined in user.service file

```
import { Component, Input } from '@angular/core';
import { UserService } from '../services/user.service';

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css',
  providers:[UserService]
})
export class UserComponent {
@Input() user:{name:string;status:string};
@Input() id:number;
constructor(private userService:UserService){}
onUpdateStatus(status:string){
  this.userService.updateStatus(this.id,status);
}
}
```

Note: Still add user, set active, set in-active we cannot see in the output.

43. Understand the Hierarchical Injection of the Services and sharing of instances in the angular.

The problem is we are injecting the service in each components separately (ie in app, user and add-user component). So each component will have their own instances of the service, data is not shared that is why we can not see the changes in the output

So services must be provided in the highest level ==>

App.module - then same instance of the service will be shared by all components and other services
App.component - then same instance of the service will be shared by all child components but different instances by other services
Child components - each component will have different instances

Solution: delete the providers:[UserService] from add-user.component.ts and user.component.ts files

```
import { Component } from '@angular/core';
import { UserService } from '../services/user.service';

@Component({
  selector: 'app-add-user',
  templateUrl: './add-user.component.html',
  styleUrl: './add-user.component.css',
 // providers:[UserService]
})
export class AddUserComponent {
  userName:string='';
  constructor(private userService:UserService){}
  onAddUser(){
    this.userService.addUser(this.userName,'Active');
  }
}
```

```
import { Component, Input } from '@angular/core';
import { UserService } from '../services/user.service';

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css',
//  providers:[UserService]
})
export class UserComponent {
@Input() user:{name:string;status:string};
@Input() id:number;
constructor(private userService:UserService){}
onUpdateStatus(status:string){
  this.userService.updateStatus(this.id,status);
}
}
```

44. Injecting Services into another Services. Usage of @Injectable decorator in the angular.

Create a new service log.service inside services folder

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

@Injectable({
  providedIn: 'root'
})
```

```
export class LogService {
  constructor() { }
  logStatus(status:string){
    console.log("logging the status in console and the status is :" +status);
  }
}
```

Since this service needs to be used inside another service(user service) then both services should be provided in the highest level (app.module.ts)

Remove providers from app.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { UserService } from './services/user.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css',
 // providers:[UserService]
})
export class AppComponent implements OnInit {
  title = 'service-demo-two';
  users:{ name:string; status:string}[]=[];
  constructor( private userService:UserService){}
  ngOnInit(){
    this.users = this.userService.users;
  }
}
```

Add UserService and LogService inside providers of app.module.ts file

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

import { AppModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { AddUserComponent } from './add-user/add-user.component';
import { UserComponent } from './user/user.component';
import { FormsModule } from '@angular/forms';
import { UserService } from './services/user.service';
import { LogService } from './services/log.service';
@NgModule({
  declarations: [
    AppComponent,
    AddUserComponent,
    UserComponent
  ],
```

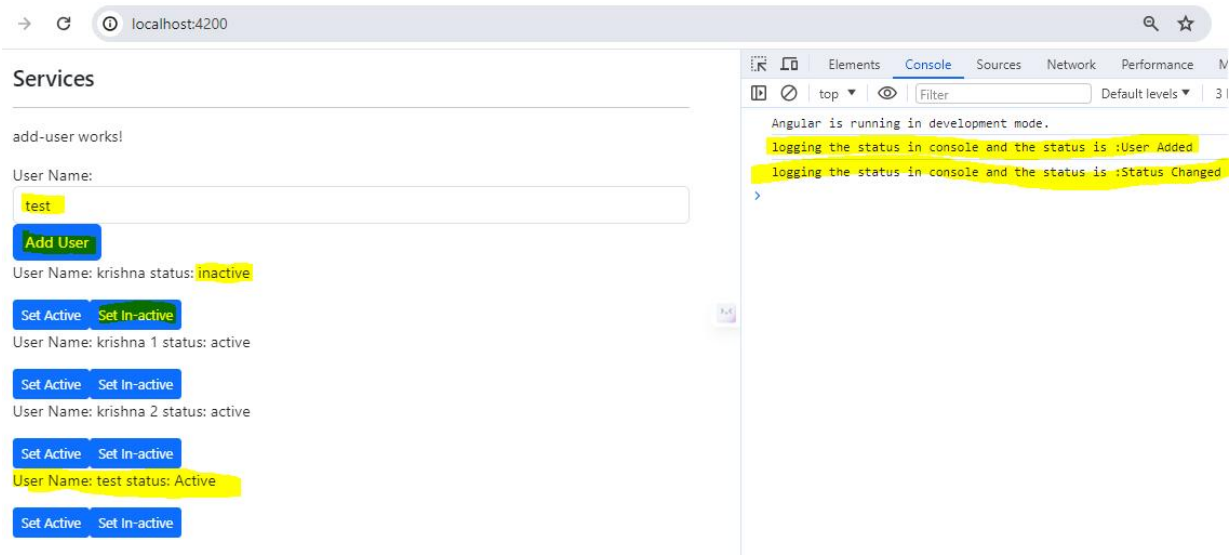
```
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [ UserService, LogService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Now inject LogService inside the UserService file through constructor

User.service.ts file

```
import { Injectable } from '@angular/core';
import { LogService } from './log.service';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  users = [
    { name:'krishna', status:'active'},
  {name:'krishna 1', status:'active' },
  {name:'krishna 2', status:'active'}
]
  constructor(private logService:LogService) { }
  addUser(name:string, status:string){
    this.users.push({name, status});
    this.logService.logStatus('User Added');
  }
  updateStatus(id:number, status:string){
    this.users[id].status = status;
    this.logService.logStatus('Status Changed');
  }
}
```

Note: while injecting a service to another service then we need to add a decorator called @Injectable, this is required only if we are creating service manually without using ng command



45. Making the cross component communication using the services by event emitter in the angular.

Lets say whenever the status gets changed in user component we want to send some data to the add-user component. This can be achieved through service We can create a new eventEmitter in user.service.ts file

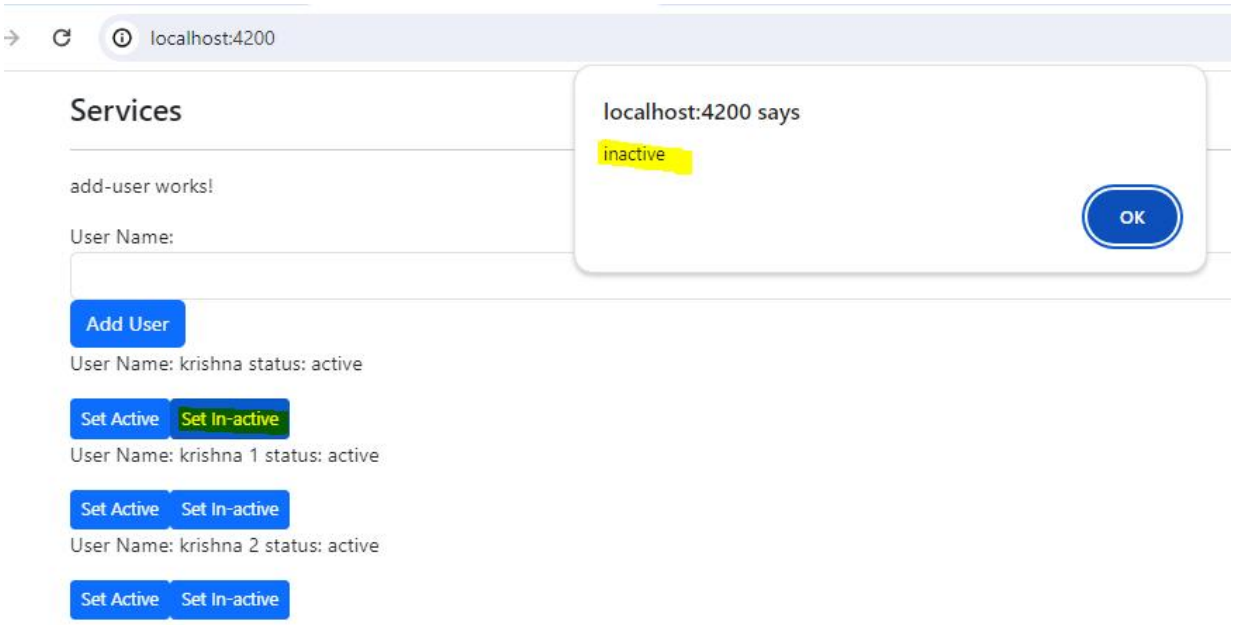
User.service.ts file

```
import { EventEmitter, Injectable } from '@angular/core';
import { LogService } from './log.service';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  users = [
    { name:'krishna', status:'active'},
  {name:'krishna 1', status:'active' },
  {name:'krishna 2', status:'active'}
]

statusUpdated = new EventEmitter<string>();

  constructor(private logService:LogService) { }
  addUser(name:string, status:string){
    this.users.push({name, status});
    this.logService.logStatus('User Added');
  }
  updateStatus(id:number, status:string){
```

```
    this.users[id].status = status;
    this.statusUpdated.emit(status);
    this.logService.logStatus('Status Changed');
  }
}
```

Now we can get that status in user.component.ts file as follows

```
import { Component, OnInit } from '@angular/core';
import { UserService } from '../services/user.service';

@Component({
  selector: 'app-add-user',
  templateUrl: './add-user.component.html',
  styleUrl: './add-user.component.css',
 // providers:[UserService]
})
export class AddUserComponent implements OnInit {
  userName:string='';
  constructor(private userService:UserService){}


 ngOnInit(){
    this.userService.statusUpdated.subscribe((data)=>{
      alert(data);
    })
  }
  onAddUser(){
    this.userService.addUser(this.userName,'Active');
  }
}
```



Angular Routing

https://youtu.be/wl18jE_LHME?si=yl0lV1tC6d9HHM-P

Create a new project (ng new angular-routing --no-standalone)
Install bootstrap (npm install bootstrap from inside the project)
Update angular.json file
Add "node_modules/bootstrap/dist/css/bootstrap.min.css"
to the styles array
Create some components (as you desire -- ng g c home etc)
To create a routing, in app.module.ts import Routes from @angular/router

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

Define a constant variable of type Routes which is an array of objects.
Each object is a key value pair which defines the path

```
const appRoutes: Routes=[
  {path:'',component:HomeComponent},
  {path:'users',component:UsersComponent},
  {path:'categories',component:CategoriesComponent},
];
```

After this RouterModule.forRoot(appRoutes) should be included in imports array of app.module.ts file, where forRoot is a method to which appRoutes should be passed

```
imports: [
    BrowserModule,
```

```
  AppRoutingModule,
  RouterModule.forRoot(appRoutes)
 ],
```

Finally the app.module.ts file will look something like below

```
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { UsersComponent } from './users/users.component';
import { CategoriesComponent } from './categories/categories.component';
import { RouterModule, Routes } from '@angular/router';
const appRoutes: Routes=[
  {path:'',component:HomeComponent},
  {path:'users',component:UsersComponent},
  {path:'categories',component:CategoriesComponent},
];
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    UsersComponent,
    CategoriesComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    RouterModule.forRoot(appRoutes)
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Update the app.component.html file as below
<router-outlet> is the one who is responsible for routing properly in the front end

```
<div class="container">
 <div class="row">
  <div class="col-md-3">
   <ul class = "nav flex-column nav-pills">
    <li class="nav-item ">
                <a class = "nav-link" href="/">Home</a>
        </li>
    <li class="nav-item ">
                <a class = "nav-link" href="/users">Users</a>
         </li>
    <li class="nav-item ">
                <a class = "nav-link" href="/categories">Categories</a>
        </li>
   </ul>
  </div>

  <div class="col-md-9">
   <div class="row">
```

```
      <div class="col-md-12">
       <router-outlet></router-outlet>
      </div>
    </div>
  </div>
 </div>
</div>
```

But in the above configuration every time we select a path page gets reloaded, which is not the required behaviour of SPA (Single Page Application)

Solution:
We should not use href instead we should use the directive routerLink as below

```
<ul class="nav flex-column nav-pills">
      <li class="nav-item">
              <a class="nav-link" routerLink="/">Home</a>
            </li>
      <li class="nav-item">
        <a class="nav-link" routerLink="/users">Users</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" routerLink="/categories">Categories</a>
      </li>
    </ul>
```

routerLink can also be used as property binding as below,

```
<li class="nav-item ">
<a class="nav-link" [routerLink]="['/categories']">Categories</a>
</li>
```

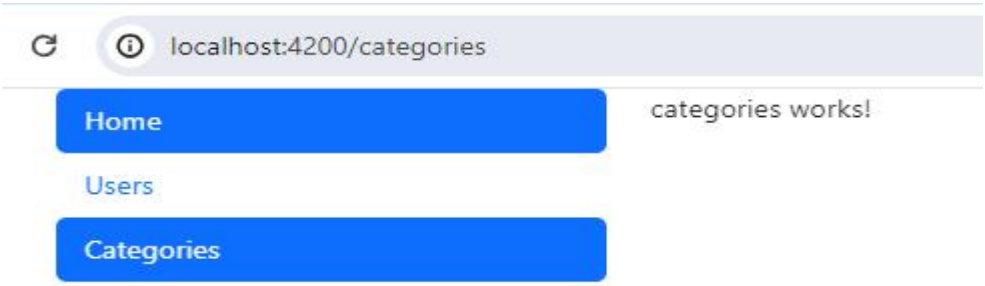
Styling the Active Router Link using routerLinkActive and routerLinkActiveOptions in angular.

We can use routerLinkActive directive to style the active path.

```
<div class="col-md-3">
    <ul class="nav flex-column nav-pills">
      <li class="nav-item">
        <a class="nav-link" routerLink="/" routerLinkActive="active">Home</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" routerLink="/users" routerLinkActive="active"
          >Users</a
        >
      </li>
```

```
      <li class="nav-item">
        <a class="nav-link" routerLink="/categories"
routerLinkActive="active"
          >Categories</a
        >
      </li>
    </ul>
  </div>
```

But the problem is default path (in this case Home is always active irrespective of which link is actually active)



To overcome this problem we can use routerLinkActiveOptions directive. This directive should be used to the default path (/ in this case , I.e Home)

```
<div class="col-md-3">
    <ul class="nav flex-column nav-pills">
      <li class="nav-item">
        <a
          class="nav-link"
          routerLink="/"
          routerLinkActive="active"
          [routerLinkActiveOptions]="{ exact: true }"
          >Home</a
        >
      </li>
      <li class="nav-item">
        <a class="nav-link" routerLink="/users" routerLinkActive="active"
          >Users</a
        >
      </li>
      <li class="nav-item">
        <a class="nav-link" routerLink="/categories" routerLinkActive="active"
          >Categories</a
        >
      </li>
    </ul>
  </div>
```

Navigate between pages using router programmatically in Typescript code in angular

Situation==> Say for example we have a button on a page and when we click on that button we want to navigate to another page.
First create a button in the required component.html file with (click) property and assingn the function to be called on the click of the button.

```
<button class="btn btn-primary " (click)="onCategoriesClick()">Go to Categories</button>
```

In the corresponding component.ts file create the method which is mentioned in the button.
In order to support routing, we need to import Router from '@angular/core and inject Router through the constructor.

Then in the method we write the logic for routing

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

@Component({
  selector: 'app-users',
  templateUrl: './users.component.html',
  styleUrl: './users.component.css'
})
export class UsersComponent implements OnInit {
  constructor(private router:Router){}
  ngOnInit(){
  }
  onCategoriesClick(){
    //this.router.navigateByUrl('/categories'); // any one method can be used
    this.router.navigate(['/categories']);
  }
}
```

50. Passing and Fetching Parameters to Routes using ActivatedRoute snapshot in Angular.

Situation:
Say for example we want to change the url dynamically something lile (/users/1/krishna or /users/2/moorthy) and a new component (lets say usercomponent should be loaded)

Solution:
Create the user component (ng g c user)

Update the app.module.ts file by adding new path as below

```
const appRoutes: Routes=[
  {path:'',component:HomeComponent},
  {path:'users',component:UsersComponent},
  {path:'categories',component:CategoriesComponent},
  {path:'users/:id/:name', component:UserComponent},
];
```

In the new path :id and :name will change dynamically

The id and name should be received in the user component. For this

In userComponent.ts file (that is the page which we want to show) We need to import ActivatedRoute (this holds the details of current route, this is required because we need id and name from current activated route) from '@angular/core' and inject through constructor.

Next define a variable to store the id and name received from the route

Next inside the ngOnInit() method store the id and name received from through the route
Finally the component.ts file will looks something like below

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

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css'
})
export class UserComponent implements OnInit {
  user! : {
    id: string,
```

```
  name:string
 }
 constructor(private route:ActivatedRoute){}
 ngOnInit(){
  this.user = {
    id:this.route.snapshot.params['id'],
    name:this.route.snapshot.params['name']
  }
 }
}
```

Update the user.component.html file as follows

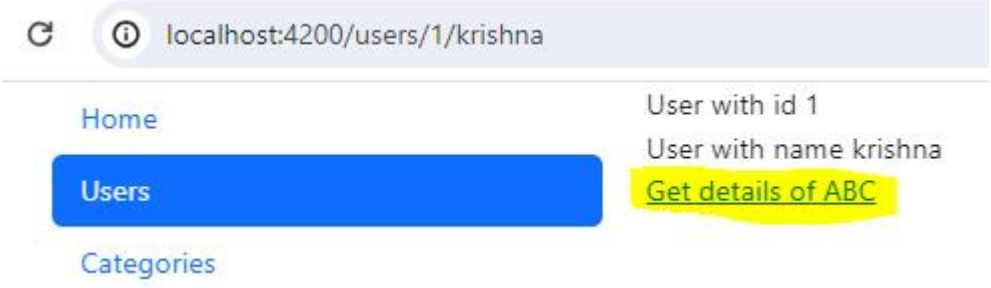
```
<div>User with id {{user.id}}</div>
<div>User with name {{user.name}}</div>
```

Drawback is id and name should be typed in the url then only it works.

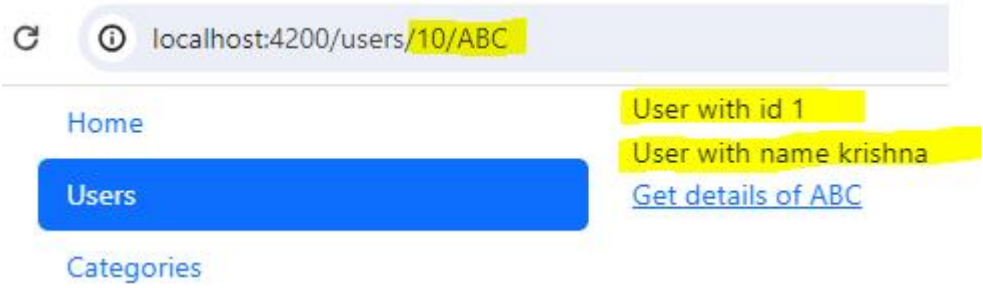
51. Fetch Route Parameters Reactively using Params Subscribe with ActivatedRoute in angular.

Another drawback with the previous approach:
Imagine we have a button on user.component.html page to get the details of another user with id 10 and name abc (example), html file will look like below

```
<div>User with id {{user.id}}</div>
<div>User with name {{user.name}}</div>

<div>
   <a [routerLink]="['/users', '10','ABC']">Get details of ABC</a>
</div>
```



When we click on the link the url gets changed, but the message remains same



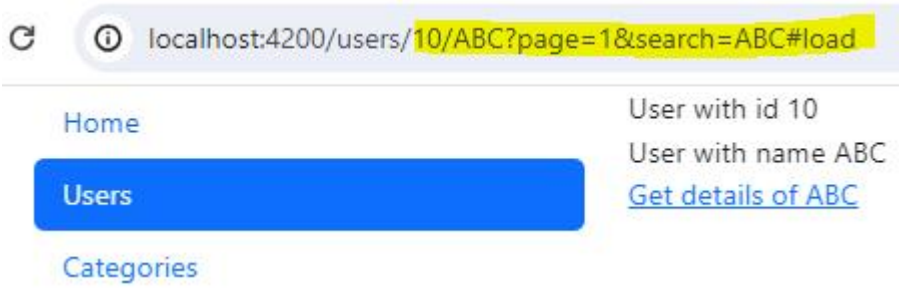
To overcome this problem, we need to use subscribe method inside ngOnInit method as below in component.ts file.

```
import { ActivatedRoute, Params } from '@angular/router';

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css'
})
export class UserComponent implements OnInit {
  user! : {
    id: string,
    name:string
  }
  constructor(private route:ActivatedRoute){}
  ngOnInit(){
    this.user = {
      id:this.route.snapshot.params['id'],
      name:this.route.snapshot.params['name']
    };
    this.route.params.subscribe((data:Params)=>{
      this.user = {
        id:data['id'],
        name:data['name']
      };
    });
  }
```

52. Passing Query Parameters and Fragments to the Url Route with the Template and Program in Angular

Passing query parameters and fragments statically

```
<div>User with id {{ user.id }}</div>
<div>User with name {{ user.name }}</div>

<div>
 <a
  [routerLink]="['/users', '10', 'ABC']"
  [queryParams]="{ page: 1, search: 'ABC' }"
  [fragment]="'load'"
  >Get details of ABC</a
 >
</div>
```



Passing query parameters and fragments dynamically

Modify the html file as below

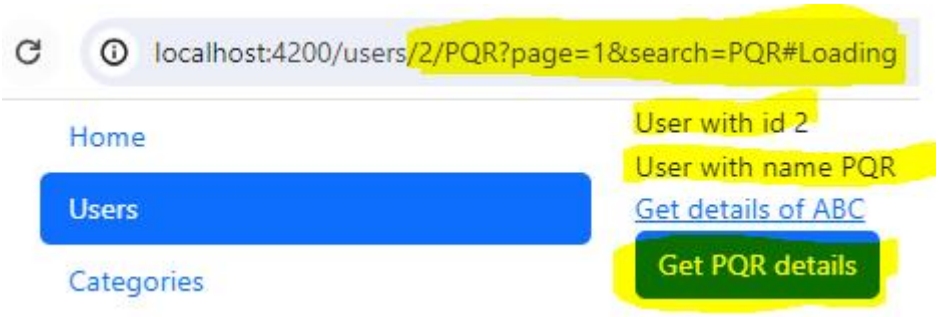
```
<div>User with id {{ user.id }}</div>
<div>User with name {{ user.name }}</div>

<div>
 <a
  [routerLink]="['/users', '10', 'ABC']"
  [queryParams]="{ page: 1, search: 'ABC' }"
  [fragment]="'load'"
  >Get details of ABC</a
```

```
  >
</div>
<div>
  <button class="btn btn-primary" (click)="getPQRDetails()">Get PQR details</button>
</div>
```

In component.ts file inject router through the constructor
Define the method getPQRDetails

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Params, Router } from '@angular/router';

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css'
})
export class UserComponent implements OnInit {
```

```
  user! : {
    id: string,
    name:string
  }
  constructor(private route:ActivatedRoute, private router:Router){}
  ngOnInit(){
    this.user = {
      id:this.route.snapshot.params['id'],
      name:this.route.snapshot.params['name']
    };
    this.route.params.subscribe((data:Params)=>{
      this.user = {
        id:data['id'],
        name:data['name']
      };
    });
  }
  getPQRDetails(){
    this.router.navigate(['/users', 2, 'PQR'],
    {queryParams:{page:1, search:'PQR'},
    fragment:'Loading'
  });
  }
}
```



53. Retrieving Query Parameters and Fragments from the URL through Typescript Code in the angular, update later

Users.component.html file

```
<p>users works!</p>
<button class="btn btn-primary" (click)="onCategoriesClick()">
  Go to Categories
</button>

<div>
  <a
    [routerLink]="['/users', 10, 'ABC']"
    [queryParams]="{ page: 1, search: 'ABC' }"
    [fragment]="'load'"
    >Get Details of ABC</a
  >
</div>
```

User.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Params, Router } from '@angular/router';

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css'
})
export class UserComponent implements OnInit {
  user! : {
    id: string,
    name:string
  }
  constructor(private route:ActivatedRoute, private router:Router){}
  ngOnInit(){
    this.user = {
      id:this.route.snapshot.params['id'],
```

```
    name:this.route.snapshot.params['name']
   };
   this.route.params.subscribe((data:Params)=>{
    this.user = {
     id:data['id'],
     name:data['name']
    };
   });
   console.log(this.route.snapshot.queryParams);
   console.log(this.route.snapshot.fragment);
  }
  getPQRDetails(){
   this.router.navigate(['/users', 2, 'PQR'],
   {queryParams:{page:1, search:'PQR'},
   fragment:'Loading'
  });
  }
}
```

Through subscribe also we can retrive the queryparams and fragments as below

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Params, Router } from '@angular/router';

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css'
})
export class UserComponent implements OnInit {
  user! : {
    id: string,
    name:string
  }
  constructor(private route:ActivatedRoute, private router:Router){}
  ngOnInit(){
    this.user = {
      id:this.route.snapshot.params['id'],
      name:this.route.snapshot.params['name']
    };
    this.route.params.subscribe((data:Params)=>{
      this.user = {
        id:data['id'],
        name:data['name']
      };
    });
    // console.log(this.route.snapshot.queryParams);
    // console.log(this.route.snapshot.fragment);
    this.route.queryParams.subscribe((data)=>{console.log(data)});
    this.route.fragment.subscribe(data=>console.log(data));
```

```
  }
  getPQRDetails(){
    this.router.navigate(['/users', 2, 'PQR'],
    {queryParams:{page:1, search:'PQR'},
    fragment:'Loading'
  });
  }
}
```

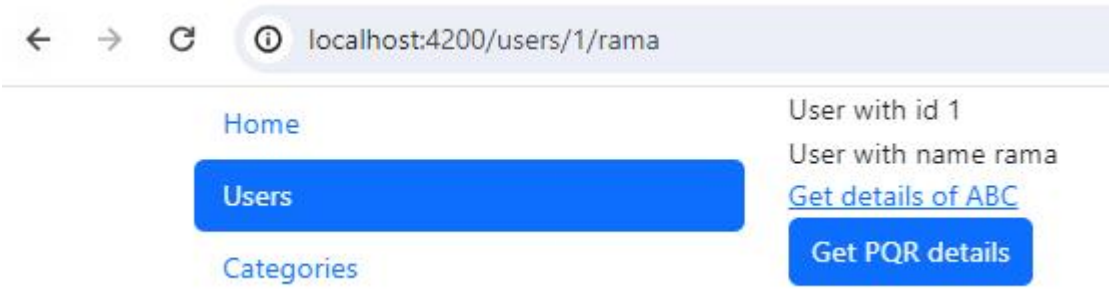
54. Setting up the child or Nested Routes using the children key in routing module in the Angular.

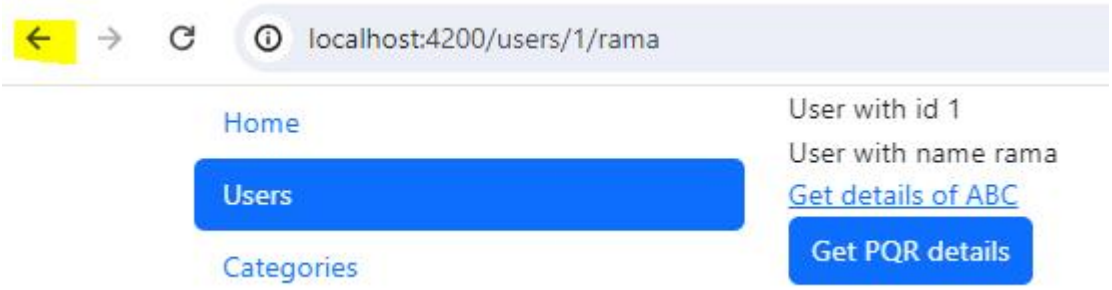
Lets say we are in users component where we have list of users. If we click on each of the user his id and name should be displayed through user component.

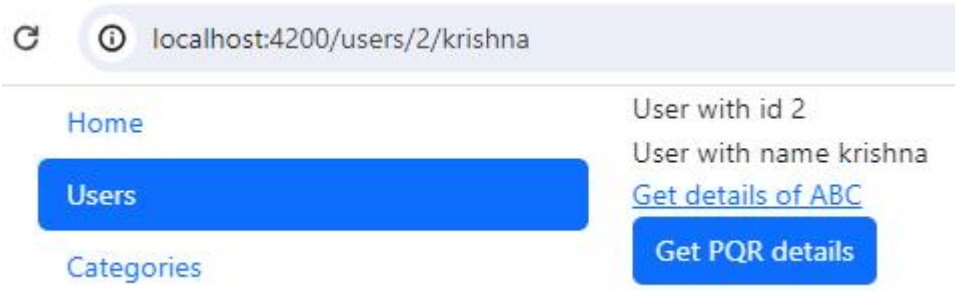
Users.component.html file

```
<ul class="nav flex-column">
  <li class="nav-item ">
    <a [routerLink]="['/users',1,'rama']" class="nav-link ">Get rama details</a>
  </li>
  <li class="nav-item ">
    <a [routerLink]="['/users',2,'krishna']" class="nav-link ">Get krishna details</a>
  </li>
  <li class="nav-item ">
    <a [routerLink]="['/users',1,'john']" class="nav-link ">Get john details</a>
  </li>
</ul>
```

If we just do the above changes then if we click any user user.component will be loaded, but the problem is to check the details of another user we need to go back and click on the next user







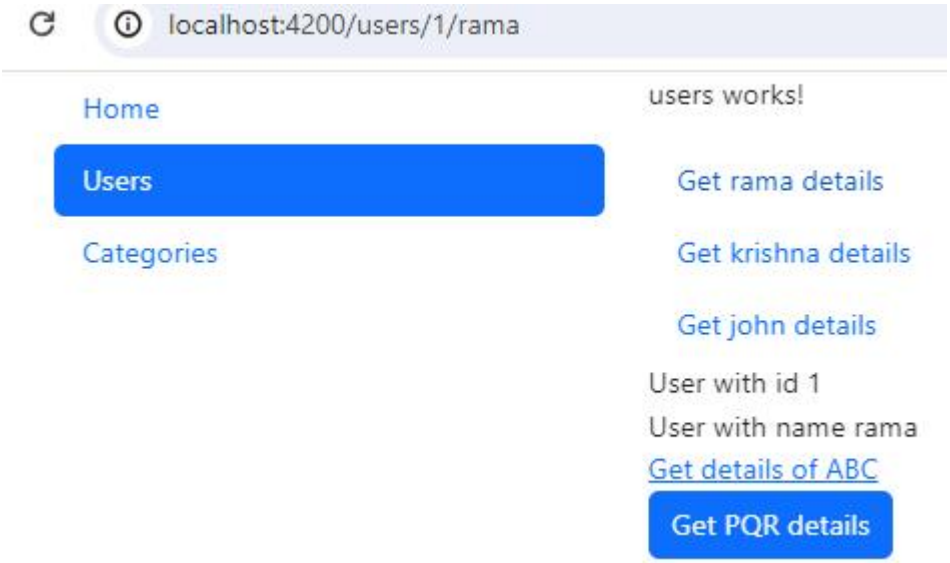
What is really required is when we click on any user below itslef user component should get loaded (I.e user component should act as child of users component)

To achieve this change the routes in app.module.ts file as below

```
import { UsersComponent } from './users/users.component';
import { CategoriesComponent } from './categories/categories.component';
import { RouterModule, Routes } from '@angular/router';
import { UserComponent } from './user/user.component';
const appRoutes: Routes = [
  { path: '', component: HomeComponent },
  {
    path: 'users',
    component: UsersComponent,
    children: [{ path: ':id/:name', component: UserComponent }]
  },
  { path: 'categories', component: CategoriesComponent },
  // {path:'users/:id/:name', component:UserComponent},
];
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    UsersComponent,
    CategoriesComponent,
    UserComponent,
  ],
  imports: [BrowserModule, AppRoutingModule, RouterModule.forRoot(appRoutes)],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Update the users.component.html file by adding router-outlet as follows

```
<ul class="nav flex-column">
  <li class="nav-item ">
    <a [routerLink]="['/users',1,'rama']" class="nav-link ">Get rama details</a>
  </li>
  <li class="nav-item ">
    <a [routerLink]="['/users',2,'krishna']" class="nav-link ">Get krishna details</a>
  </li>
  <li class="nav-item ">
    <a [routerLink]="['/users',1,'john']" class="nav-link ">Get john details</a>
  </li>
</ul>
<div>
  <router-outlet></router-outlet>
</div>
```



55. Preserve or merge the query parameters by forwarding with queryparamsHandling in Angular.

If we want to send query params form one page to another page

Create a new component edit-user

Lets say we have some query params in the users component links as below

```
<ul class="nav flex-column">
 <li class="nav-item">
  <a
   [routerLink]="['/users', 1, 'rama']"
   [queryParams]="{ page: 1, search: 'rama' }"
   class="nav-link"
   >Get rama details</a
  >
 </li>
 <li class="nav-item">
  <a
   [routerLink]="['/users', 2, 'krishna']"
   [queryParams]="{ page: 1, search: 'krishna' }"
   class="nav-link"
   >Get krishna details</a
  >
 </li>
 <li class="nav-item">
  <a
   [routerLink]="['/users', 1, 'john']"
   [queryParams]="{ page: 1, search: 'john' }"
```

```
    class="nav-link"
    >Get john details</a
   >
  </li>
</ul>
<div>
  <router-outlet></router-outlet>
</div>
```

Add the new path in app.module.ts file

```
const appRoutes: Routes = [
  { path: '', component: HomeComponent },
  {
    path: 'users',
    component: UsersComponent,
    children: [{ path: ':id/:name', component: UserComponent },
               { path: ':id/:name/edit', component: EditUserComponent }
              ]
  },
  { path: 'categories', component: CategoriesComponent },
  // {path:'users/:id/:name', component:UserComponent},
];
```

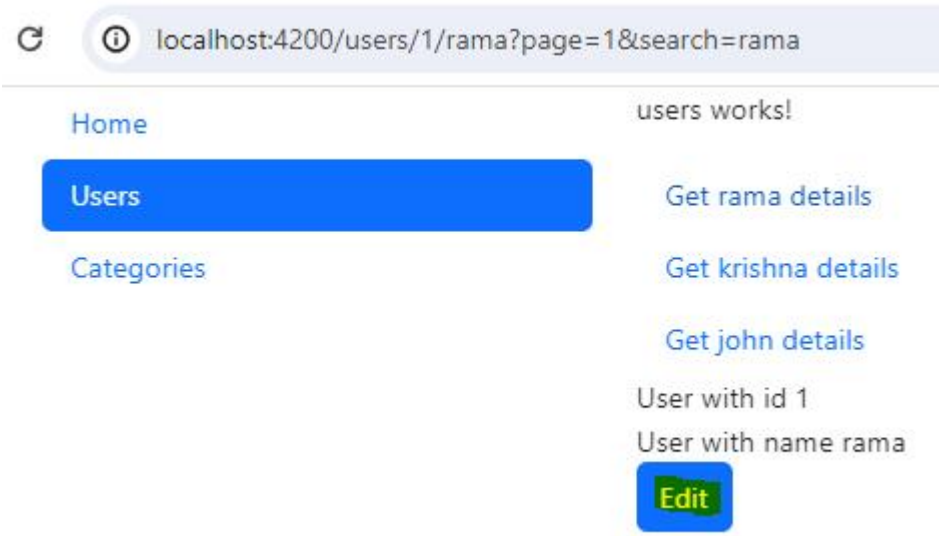
In user component add a button to edit the user as below
User.component.hmtl

```
<div>User with id {{ user.id }}</div>
<div>User with name {{ user.name }}</div>

<!-- <div>
  <a
    [routerLink]="['/users', '10', 'ABC']"
    [queryParams]="{ page: 1, search: 'ABC' }"
    [fragment]="'load'"
    >Get details of ABC</a
  >
</div>
<div>
  <button class="btn btn-primary" (click)="getPQRDetails()">Get PQR details</button>
</div> -->
<div>
  <button class="btn btn-primary" (click)="onEditUser()">Edit</button>
</div>
```

Now in user.component.ts file we need to define the method onEditUser and write logic to navigate to edituser component and to send the query parameters as below

User.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Params, Router } from '@angular/router';

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css'
})
export class UserComponent implements OnInit {
  user! : {
    id: string,
    name:string
  }
  constructor(private route:ActivatedRoute, private router:Router){}
  ngOnInit(){
    this.user = {
      id:this.route.snapshot.params['id'],
      name:this.route.snapshot.params['name']
    };
    this.route.params.subscribe((data:Params)=>{
      this.user = {
        id:data['id'],
        name:data['name']
      };
    });
    // console.log(this.route.snapshot.queryParams);
    // console.log(this.route.snapshot.fragment);
    this.route.queryParams.subscribe((data)=>{console.log(data)});
    this.route.fragment.subscribe(data=>console.log(data));
  }
  getPQRDetails(){
    this.router.navigate(['/users', 2, 'PQR'],
    {queryParams:{page:1, search:'PQR'},
    fragment:'Loading'
  });
  }
  onEditUser(){
    this.router.navigate(['/users',this.user.id,this.user.name, 'edit'],
{queryParamsHandling:'preserve'});
  }
}
```



56. Implement Custom 404 Page adding wildcard Route, redirectTo option in the angular routing module

To show a custom 404 page when the url does not match any of the paths

Create a new component say page-not-fonud

Page-not-found.component.html

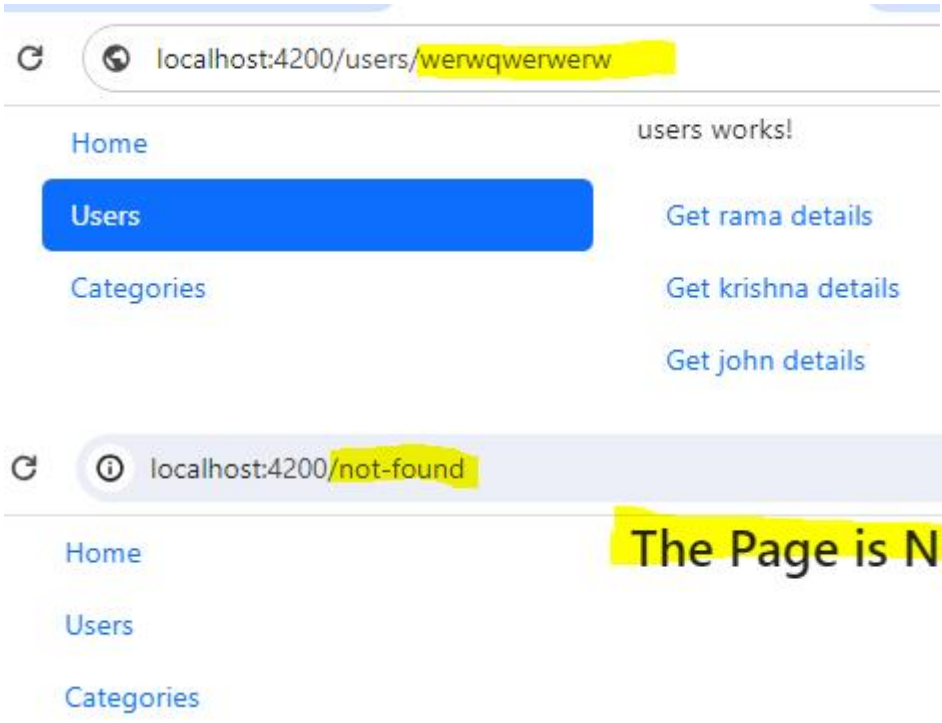
```
<h3>The Page is NOT FOUND</h3>
```

App.module.ts file

Add the routes as below : remember it should be added after all the valid routes

```
const appRoutes: Routes = [
  { path: '', component: HomeComponent },
  {
    path: 'users',
```

```
    component: UsersComponent,
    children: [{ path: ':id/:name', component: UserComponent },
               { path: ':id/:name/edit', component: EditUserComponent }
              ]
  },
  { path: 'categories', component: CategoriesComponent },
  // {path:'users/:id/:name', component:UserComponent},
  {path: 'not-found', component:NotFoundPageComponent},
  {path:'**', redirectTo:'not-found'}
];
```



57. Separate all the Routing configuration code into another file app-routing.module in the angular.

Because of all routing paths app.module.ts file became very clumsy. So we can write all routing related code in a separate file and import it in app.module.ts file

Create a new file app-routing.module.ts inside app folder

Remove all the codes related to routing from app.module.ts file and paste it in app-routing.module.ts file

App-routing.module.ts file

```
import { HomeComponent } from './home/home.component';
import { UsersComponent } from './users/users.component';
```

```
import { CategoriesComponent } from './categories/categories.component';
import { UserComponent } from './user/user.component';
import { EditUserComponent } from './edit-user/edit-user.component';
import { NotFoundPageComponent } from './not-found-page/not-found-
page.component';

const appRoutes: Routes = [
  { path: '', component: HomeComponent },
  {
    path: 'users',
    component: UsersComponent,
    children: [{ path: ':id/:name', component: UserComponent },
               { path: ':id/:name/edit', component: EditUserComponent }
              ]
  },
  { path: 'categories', component: CategoriesComponent },
  // {path:'users/:id/:name', component:UserComponent},
  {path: 'not-found', component:NotFoundPageComponent},
  {path:'**', redirectTo:'not-found'}
];
@NgModule({
  imports: [RouterModule.forRoot(appRoutes)],
  exports: [RouterModule] //very important
})
export class AppRoutingModule { }
```

App.module.ts file

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { UsersComponent } from './users/users.component';
import { CategoriesComponent } from './categories/categories.component';
import { UserComponent } from './user/user.component';
import { EditUserComponent } from './edit-user/edit-user.component';
import { NotFoundPageComponent } from './not-found-page/not-found-
page.component';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    UsersComponent,
    CategoriesComponent,
    UserComponent,
    EditUserComponent,
```

```
    NotFoundPageComponent,
  ],
  imports: [BrowserModule, AppRoutingModule],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

58. Introduction to Routing Guards. Implementation of canActivate Route Guard in the angular.
Ref : routing-guard
Create a new project routing-guard
Install bootstrap
Create all the components- home, users, user, categories, not-found

Setup routing in app-routing.module.ts file

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { UsersComponent } from './users/users.component';
import { CategoriesComponent } from './categories/categories.component';
import { NotFoundComponent } from './not-found/not-found.component';

const routes: Routes = [
  {path:'',component:HomeComponent},
  {path:'users', component:UsersComponent},
  {path:'categories',component:CategoriesComponent},
  {path:'not-found',component:NotFoundComponent},
  {path:'**', redirectTo:'not-found'}
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

App.module.ts file

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { UsersComponent } from './users/users.component';
import { CategoriesComponent } from './categories/categories.component';
import { UserComponent } from './user/user.component';
import { NotFoundComponent } from './not-found/not-found.component';
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    UsersComponent,
    CategoriesComponent,
    UserComponent,
    NotFoundComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

App.component.html file

```
<div class="container">
  <h3>Routing Guard</h3>
  <div class="row">
    <div class="col-md-3">
      <ul class="nav flex-column nav-pills">
        <li class="nav-item">
          <a
            class="nav-link"
            routerLink="/"
            routerLinkActive="active"
            [routerLinkActiveOptions]="{ exact: true }"
            >Home</a
          >
        </li>
        <li class="nav-item">
          <a class="nav-link" routerLink="/users" routerLinkActive="active"
            >Users</a
          >
```

```
      </li>
      <li class="nav-item">
        <a class="nav-link" routerLink="/categories"
        routerLinkActive="active"
          >Categories</a
        >
      </li>
    </ul>
  </div>

  <div class="col-md-9">
    <div class="row">
      <div class="col-md-12">
        <router-outlet></router-outlet>
      </div>
    </div>
  </div>
 </div>
</div>
```

Create a folder services inside app
Create another folder guard inside services

Create auth.service.ts inside services folder

```
export class AuthService{
  isLoggedIn = false;

  login(){
     this.isLoggedIn = true;
  }
  logout(){
     this.isLoggedIn = true;
    }
      isAuthenticated(){
     return this.isLoggedIn;
  }

}
```

Create another service auth-guard.service.ts inside guard folder

```
import {
 ActivatedRouteSnapshot,
 CanActivate,
 Router,
 RouterStateSnapshot,
} from '@angular/router';
import { Observable } from 'rxjs';
```

```
import { AuthService } from '../auth.service';
import { Injectable } from '@angular/core';


@Injectable()
export class AuthGuardService implements CanActivate {
    constructor(private authService:AuthService, private router:Router){}
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ):
    | boolean
    | Observable<boolean>
    | Promise<boolean> {
   let isLoggedIn =  this.authService.isAuthenticated();
   if(isLoggedIn){
    return true;
   }
   else{
   return this.router.navigate(['/']);
   }
  }
}
```

Add this service in providers[] array of app.module.ts

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { UsersComponent } from './users/users.component';
import { CategoriesComponent } from './categories/categories.component';
import { UserComponent } from './user/user.component';
import { NotFoundComponent } from './not-found/not-found.component';
import { AuthService } from './services/auth.service';
import { AuthGuardService } from './services/guard/auth-guard.service';
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    UsersComponent,
    CategoriesComponent,
    UserComponent,
    NotFoundComponent
  ],
```

```
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [AuthService,AuthGuardService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Now we need to modify the routing path, we need to add canActivate to the path to which we need guard.
Modify app-routing.service.ts file as below (we are applying guard to the users component)

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { UsersComponent } from './users/users.component';
import { CategoriesComponent } from './categories/categories.component';
import { NotFoundComponent } from './not-found/not-found.component';
import { AuthGuardService } from './services/guard/auth-guard.service';

const routes: Routes = [
  {path:'',component:HomeComponent},
  {path:'users', component:UsersComponent, canActivate:[AuthGuardService]},
  {path:'categories',component:CategoriesComponent},
  {path:'not-found',component:NotFoundComponent},
  {path:'**', redirectTo:'not-found'}
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

We will modify the app.component.html file by adding 2 buttons to login and logout as below

```
<div class="container">
  <h3>Routing Guard</h3>
  <div class="row">
    <div class="col-md-6">
    <button class="btn btn-primary" (click)="onLogin()">Login</button>
    <button class="btn btn-primary" (click)="onLogout()">Loout</button>
  </div>
  </div>
  <div class="row">
    <div class="col-md-3">
```

```
    <ul class="nav flex-column nav-pills">
      <li class="nav-item">
        <a
          class="nav-link"
          routerLink="/"
          routerLinkActive="active"
          [routerLinkActiveOptions]="{ exact: true }"
          >Home</a
        >
      </li>
      <li class="nav-item">
        <a class="nav-link" routerLink="/users" routerLinkActive="active"
          >Users</a
        >
      </li>
      <li class="nav-item">
        <a class="nav-link" routerLink="/categories" routerLinkActive="active"
          >Categories</a
        >
      </li>
    </ul>
  </div>

  <div class="col-md-9">
    <div class="row">
      <div class="col-md-12">
        <router-outlet></router-outlet>
      </div>
    </div>
  </div>
 </div>
</div>
```

Modify the app.component.ts file as below

```
import { Component } from '@angular/core';
import { AuthService } from './services/auth.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'routing-guard';
  constructor(private authService:AuthService){}
  onLogin(){
    this.authService.login();
  }
```

```
  onLogout(){
    this.authService.logout();
  }
}
```

The code above are added so that when we click login button the variable isLoggedIn defined in auth.service will we set to true, if we press logout button then variable isLoggedIn defined in auth.service will we set to false.

So the output will be, if we login then users component will be accessible or else we cannot access users component.

Login case output

Logout case output

59. Implement canActivateChild Route Guard for the Nested Child Routes in the Angular.

Requirement: If we have some child paths and we need to guard only child path not the parent then we can use canActivateChild as below

Add child path to users in app-routing.component.ts file
Instead of canActivate, we should use canActivateChild

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { UsersComponent } from './users/users.component';
import { CategoriesComponent } from './categories/categories.component';
import { NotFoundComponent } from './not-found/not-found.component';
import { AuthGuardService } from './services/guard/auth-guard.service';
import { UserComponent } from './user/user.component';

const routes: Routes = [
  { path: '', component: HomeComponent },
  {
    path: 'users',
    component: UsersComponent,
    canActivateChild: [AuthGuardService],
    children: [{ path: ':id/:name', component: UserComponent }],
  },
  { path: 'categories', component: CategoriesComponent },
  { path: 'not-found', component: NotFoundComponent },
  { path: '**', redirectTo: 'not-found' },
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

Modify the users.component.html file as below

```
<p>users works!</p>
<ul class="nav flex-column">
    <li class="nav-item">
      <a
        [routerLink]="['/users', 1, 'rama']"
        [queryParams]="{ page: 1, search: 'rama' }"
        class="nav-link"
        >Get rama details</a
      >
    </li>
    <li class="nav-item">
      <a
        [routerLink]="['/users', 2, 'krishna']"
        [queryParams]="{ page: 1, search: 'krishna' }"
        class="nav-link"
        >Get krishna details</a
      >
```

```
    </li>
    <li class="nav-item">
      <a
        [routerLink]="['/users', 1, 'john']"
        [queryParams]="{ page: 1, search: 'john' }"
        class="nav-link"
        >Get john details</a
      >
    </li>
  </ul>
  <div>
    <router-outlet></router-outlet>
  </div>
```

Modify the auth-guard.service.ts file as below

```
import {
  ActivatedRouteSnapshot,
  CanActivate,
  CanActivateChild,
  Router,
  RouterStateSnapshot,
} from '@angular/router';
import { Observable } from 'rxjs';
import { AuthService } from '../auth.service';
import { Injectable } from '@angular/core';


@Injectable()
export class AuthGuardService implements CanActivate, CanActivateChild {
    constructor(private authService:AuthService, private router:Router){}
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ):
    | boolean
    | Observable<boolean>
    | Promise<boolean> {
   let isLoggedIn =  this.authService.isAuthenticated();
   if(isLoggedIn){
    return true;
   }
   else{
   return this.router.navigate(['/']);
   }
  }
```

```
  canActivateChild(route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): | boolean
    | Observable<boolean>
    | Promise<boolean>{
     return this.canActivate(route,state);
    }
}
```

Aonter way:

Auth-guard.service.ts file

```
import {
  ActivatedRouteSnapshot,
  CanActivateChild,
  Router,
  RouterStateSnapshot,
} from '@angular/router';
import { Observable } from 'rxjs';
import { AuthService } from '../auth.service';
import { Injectable } from '@angular/core';


@Injectable()
export class AuthGuardService implements  CanActivateChild {
    constructor(private authService:AuthService, private router:Router){}
  canActivateChild(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ):
    | boolean
    | Observable<boolean>
    | Promise<boolean> {
   let isLoggedIn =  this.authService.isAuthenticated();
   if(isLoggedIn){
    return true;
   }
   else{
   return this.router.navigate(['/']);
   }
  }

}
```

Now the out put is even if we are logged out we can access users component, but user component is accessible only we if login

If we want we can also written promise

Modify auth.service.ts as below

```
export class AuthService{
    isLoggedIn = false;

    login(){
        this.isLoggedIn = true;
    }
    logout(){
        this.isLoggedIn = false;
    }
    isAuthenticated(){
     //   return this.isLoggedIn;
   return  new Promise((resolve, reject)=>{
        setTimeout(()=>{
            resolve(this.isLoggedIn);
        },1000);
     })
    }
}
```

Modify auth-guard.service.ts file as below

```
import {
  ActivatedRouteSnapshot,
  CanActivate,
  CanActivateChild,
  Router,
  RouterStateSnapshot,
} from '@angular/router';
import { Observable } from 'rxjs';
import { AuthService } from '../auth.service';
import { Injectable } from '@angular/core';


@Injectable()
export class AuthGuardService implements CanActivate, CanActivateChild {
```

```
    constructor(private authService:AuthService, private router:Router){}
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ):
    | boolean
    | Observable<boolean>
```

```
  | Promise<boolean> {
//  let isLoggedIn =  this.authService.isAuthenticated();
//  if(isLoggedIn){
//   return true;
//  }
//  else{
//  return this.router.navigate(['/']);
//  }
    return this.authService.isAuthenticated().then((data)=>{
      if(data){
        return true;
      }
      else{
        return this.router.navigate(['/']);
      }
    });
 }
 canActivateChild(route: ActivatedRouteSnapshot,
   state: RouterStateSnapshot): | boolean
   | Observable<boolean>
   | Promise<boolean>{
    return this.canActivate(route,state);
   }
}
```

Now in the output when we click rama details and so on it will take 1s time to redirect

60. Controlling Navigation with CanDeactivate Route Guard in the angular

71. Template Driven Forms in Angular. Get NgForm Object from the template to code in Angular.

Forms in Angular

71. Template Driven Forms in Angular. Get NgForm Object from the template to code in Angular.

Ref: forms-demo-one (it is brand new project, not the one in tutorial)
Template approach

Create a new project
Install bootstrap

Create a form in app.component.html as below

```
<div class="container">
 <div class="row">
  <div class="col-md-12">
   <form>
    <div class="form-group">
     <label>User Name: </label>
     <input type="text" class="form-control" id="username">
    </div>
    <div class="form-group">
     <label>Email : </label>
     <input type="text" class="form-control" id="email">
    </div>
    <div class="form-group">
     <label>Gender</label>
     <select class="form-control" id="gender">
      <option value="male">Male</option>
      <option value="Female">Female</option>
     </select>
    </div>
    <div class="mt-2">
     <button type="submit" class="btn btn-primary">Submit</button>
    </div>
   </form>
  </div>
 </div>
</div>
```

Inoder to convert the above form into angular template form follow the below steps

Add name and ngModel property to all the input fields of the form (ngModel is not two way data binding here, name is required to capture all these data in ts file)
Add a local reference in the form tag
Add (ngSubmit) property to the form tag and call some method and pass the local reference as parameter

```
<div class="container">
 <div class="row">
  <div class="col-md-12">
   <form (ngSubmit)="onSubmit(f)" #f=”ngForm”>
    <div class="form-group">
     <label>User Name: </label>
     <input
      type="text"
      class="form-control"
      id="username"
```

```
          name="username"
          ngModel
        />
      </div>
      <div class="form-group">
        <label>Email : </label>
        <input
          type="text"
          class="form-control"
          id="email"
          name="email"
          ngModel
        />
      </div>
      <div class="form-group">
        <label>Gender</label>
        <select class="form-control" id="gender" name="gender" ngModel>
          <option value="male">Male</option>
          <option value="Female">Female</option>
        </select>
      </div>
      <div class="mt-2">
        <button type="submit" class="btn btn-primary">Submit</button>
      </div>
    </form>
  </div>
 </div>
</div>
```

FormsModule should be imported from @angular/forms and added to imports array of app.module.ts file

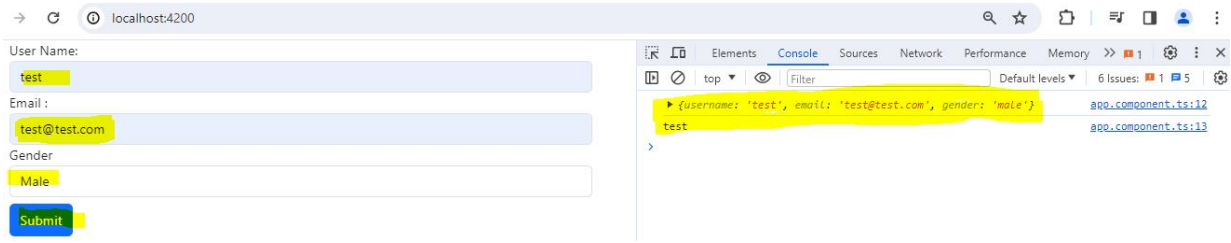
```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';
@NgModule({
 declarations: [
   AppComponent
 ],
 imports: [
   BrowserModule,
   AppRoutingModule,
   FormsModule
 ],
 providers: [],
 bootstrap: [AppComponent]
```

```
})
export class AppModule { }
```

Define the the method in respective ts file as below

```
import { Component } from '@angular/core';
import { NgForm } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'forms-demo-one';
  onSubmit(form:NgForm){
    console.log(form.value);
    console.log(form.value['username']);
  }
}
```

Output



72. Advantages of using ViewChild in the Template Driven Form to get Form Object in angular.

We can perform the above activities without passing the reference f to the function using @ViewChild in ts file

Change the form tag in the above code as below

```
<form (ngSubmit)="onSubmit()" #f="ngForm">
```

Change the respective ts file as below

```
import { Component, ViewChild } from '@angular/core';
import { NgForm } from '@angular/forms';

@Component({
```

```
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'forms-demo-one';
  @ViewChild('f') submittedForm:NgForm;
  // onSubmit(form:NgForm){
  //   console.log(form.value);
  //   console.log(form.value['username']);
  // }
  onSubmit(){
    console.log(this.submittedForm.value);
    console.log(this.submittedForm.value['username']);
  }
}
```

Viewchild option can be used if we want to capture the data while entering in the input fields itself

74. Validations for the Template Driven Forms. Show Validation Messages for the Form in Angular.

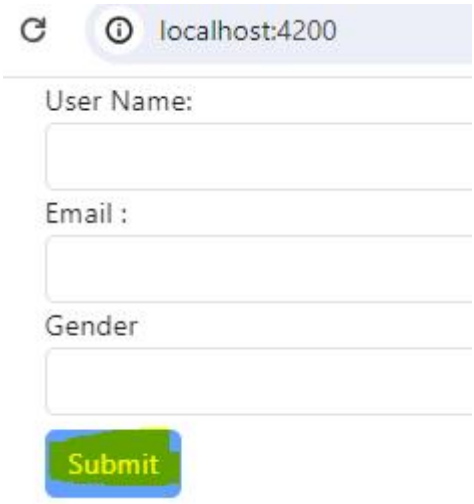
Add validations to username and email field by adding required and email

Now we want to disable the button on load and only if the form becomes valid then button should be enabled

```
<div class="container">
  <div class="row">
    <div class="col-md-12">
      <!-- <form (ngSubmit)="onSubmit(f)" #f="ngForm"> -->
        <form (ngSubmit)="onSubmit()" #f="ngForm">
        <div class="form-group">
          <label>User Name: </label>
          <input
            type="text"
            class="form-control"
            id="username"
            name="username"
            ngModel
            required
          />
        </div>
        <div class="form-group">
          <label>Email : </label>
          <input
            type="text"
```

```
          class="form-control"
          id="email"
          name="email"
          ngModel
          required
          email
        />
      </div>
      <div class="form-group">
        <label>Gender</label>
        <select class="form-control" id="gender" name="gender" ngModel>
          <option value="male">Male</option>
          <option value="Female">Female</option>
        </select>
      </div>
      <div class="mt-2">
        <button type="submit" class="btn btn-primary"
[disabled]="!f.valid">Submit</button>
      </div>
    </form>
  </div>
 </div>
</div>
```

Output
Onload

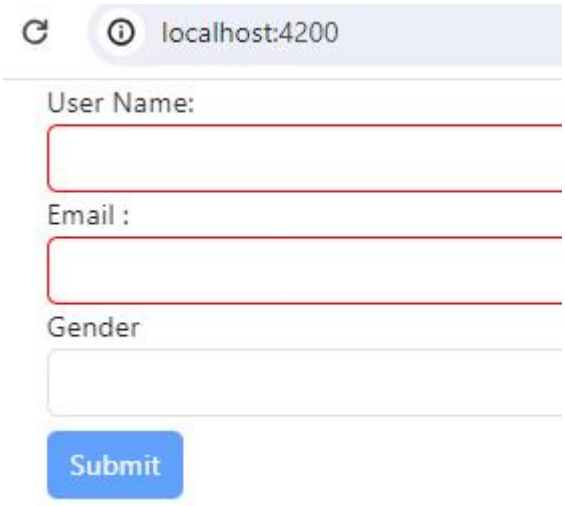


After entering user name and email button gets enabled

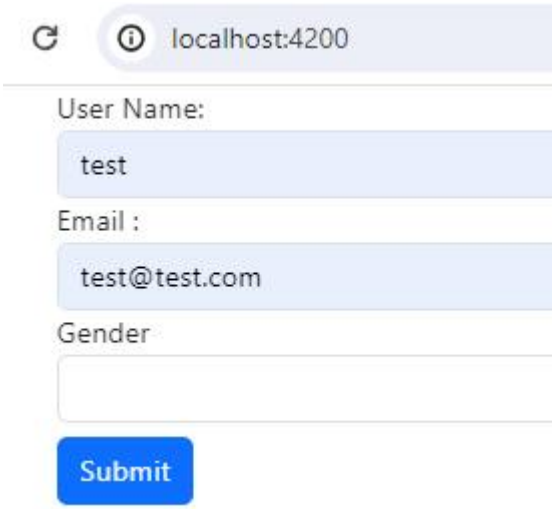
Now when ever the input fields are invalid we want to show a red color border.
In the corresponding css file do the following
App.component.css file

```
input.ng-invalid{
   border: 1px solid red;
}
```

Here input is the html tag and .ng-invalid is the angular provided class (not user defined)
Output
Onload



After entering username and email



If we want to show the red colour only if user clicks on the input files, not on load then we an add ng-touched class to the css as below

```
input.ng-invalid.ng-touched{
   border: 1px solid red;
}
```

If we want to show some warning text whenever input fields are invalid then do the following

Note: if there is error something like 'email …..'
Change

```
"noPropertyAccessFromIndexSignature": true,
```

To

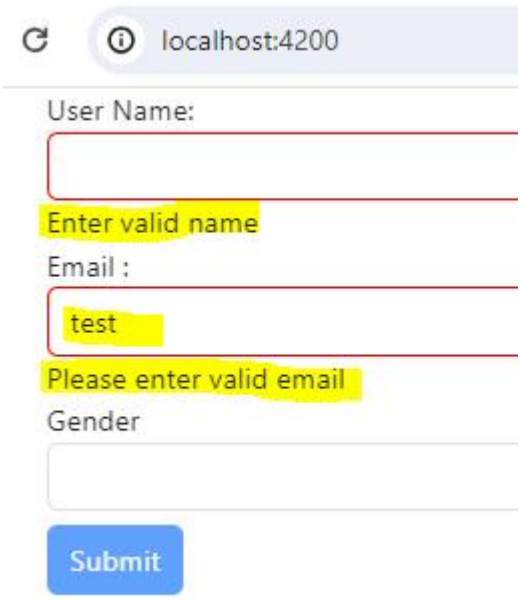
```
"noPropertyAccessFromIndexSignature": false,
```

App.component.html file

```
<div class="container">
 <div class="row">
  <div class="col-md-12">
   <!-- <form (ngSubmit)="onSubmit(f)" #f="ngForm"> -->
   <form (ngSubmit)="onSubmit()" #f="ngForm">
    <div class="form-group">
     <label>User Name: </label>
     <input
      type="text"
      class="form-control"
      id="username"
      name="username"
      ngModel
      required
```

```
        />
        <span class="help-text"
          *ngIf="f.controls.username
          && f.controls.username.touched">Enter valid name</span>
      </div>
      <div class="form-group">
        <label for="email">Email : </label>
        <input
          type="text"
          class="form-control"
          id="email"
          name="email"
          ngModel
          required
          email
        />
        <span
          class="help-text"
          *ngIf="f.controls.email &&
            !f.controls.email.valid &&
            f.controls.email.touched"
          >Please enter valid email</span>
      </div>
      <div class="form-group">
        <label>Gender</label>
        <select class="form-control" id="gender" name="gender" ngModel>
          <option value="male">Male</option>
          <option value="Female">Female</option>
        </select>
      </div>
      <div class="mt-2">
        <button type="submit" class="btn btn-primary" [disabled]="!f.valid">
          Submit
        </button>
      </div>
    </form>
  </div>
 </div>
</div>
```

Output: invalid case



Output: valid case

We can write it in another way
Rather than writing .controls.email and so on we can create a local reference to the input fields inorder to user shorthand way as below

```
<div class="container">
 <div class="row">
  <div class="col-md-12">
   <!-- <form (ngSubmit)="onSubmit(f)" #f="ngForm"> -->
   <form (ngSubmit)="onSubmit()" #f="ngForm">
    <div ngModelGroup="userData">
    <div class="form-group">
     <label>User Name: </label>
     <input
      type="text"
      class="form-control"
      id="username"
      name="username"
      ngModel
      required
```

```
        #username = "ngModel"
      />
      <span
        class="help-text"
        *ngIf="
         username &&
          !username.valid &&
          username.touched
        "
        >Enter valid name</span
      >
    </div>
    <div class="form-group">
      <label for="email">Email : </label>
      <input
        type="text"
        class="form-control"
        id="email"
        name="email"
        ngModel
        required
        email
        #email = "ngModel"
      />
      <span
        class="help-text"
        *ngIf="
         email &&
          !email.valid &&
          email.touched
        "
        >Please enter valid email</span>
    </div>
  </div>
    <div class="form-group">
      <label>Gender</label>
      <select
        class="form-control"
        id="gender"
        name="gender"
        [ngModel]="gender"
      >
        <option value="male">Male</option>
        <option value="Female">Female</option>
      </select>
    </div>
    <div class="form-group">
      <label>About Yourself</label>
```

```
        <textarea
          class="form-control"
          rows="3"
          name="about"
          [(ngModel)]="about"
        ></textarea>
        <p>{{ about }}</p>
      </div>
      <div class="mt-2">
        <button type="submit" class="btn btn-primary" [disabled]="!f.valid">
          Submit
        </button>
      </div>
    </form>
  </div>
 </div>
</div>
```

75. Using ngModel for Two Way & One Way Binding to populate Data in Template Driven Forms - Angular

In the previous output geneder is blank by default. If we want either male or female to be shown while loading, then in the App.component.ts file
Define a variable something like gender and assign male.

```
import { Component, ViewChild } from '@angular/core';
import { NgForm } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'forms-demo-one';
  gender = 'male';
  @ViewChild('f') submittedForm:NgForm;
  // onSubmit(form:NgForm){
  //   console.log(form.value);
  //   console.log(form.value['username']);
  // }
  onSubmit(){
    // console.log(this.submittedForm.value);
    // console.log(this.submittedForm.value['username']);
    console.log(this.submittedForm)
  }
```

```
}
```

In the app.component.html file
Use mgModel as a property binding (one-way, even if we change gender to female in the front end it wont change in the backend) and assign the variable defined in the ts file

```
<div class="container">
  <div class="row">
    <div class="col-md-12">
      <!-- <form (ngSubmit)="onSubmit(f)" #f="ngForm"> -->
      <form (ngSubmit)="onSubmit()" #f="ngForm">
        <div ngModelGroup="userData">
        <div class="form-group">
          <label>User Name: </label>
          <input
            type="text"
            class="form-control"
            id="username"
            name="username"
            ngModel
            required
            #username = "ngModel"
          />
          <span
            class="help-text"
            *ngIf="
             username &&
              !username.valid &&
              username.touched
            "
            >Enter valid name</span
          >
        </div>
        <div class="form-group">
          <label for="email">Email : </label>
          <input
            type="text"
            class="form-control"
            id="email"
            name="email"
            ngModel
            required
            email
            #email = "ngModel"
          />
          <span
            class="help-text"
            *ngIf="
             email &&
```

```
            !email.valid &&
            email.touched
          "
          >Please enter valid email</span>
        </div>
      </div>
      <div class="form-group">
        <label>Gender</label>
        <select
          class="form-control"
          id="gender"
          name="gender"
          [ngModel]="gender"
        >
          <option value="male">Male</option>
          <option value="Female">Female</option>
        </select>
      </div>
      <div class="form-group">
        <label>About Yourself</label>
        <textarea
          class="form-control"
          rows="3"
          name="about"
          [(ngModel)]="about"
        ></textarea>
        <p>{{ about }}</p>
      </div>
      <div class="mt-2">
        <button type="submit" class="btn btn-primary" [disabled]="!f.valid">
          Submit
        </button>
      </div>
    </form>
  </div>
 </div>
</div>
```

Two way example:

In ts file define a variable about

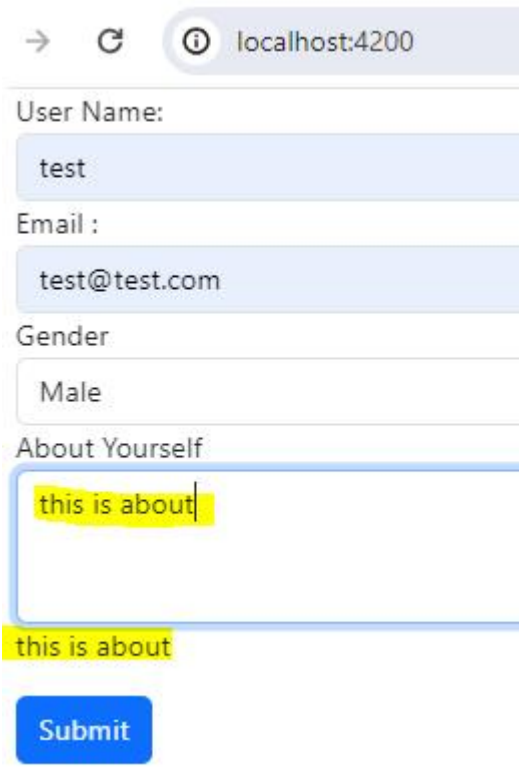
```
export class AppComponent {
 title = 'forms-demo-one';
 gender = 'male';
 about = '';
 @ViewChild('f') submittedForm:NgForm;
```

In html file update as below

```
<div class="container">
  <div class="row">
    <div class="col-md-12">
      <!-- <form (ngSubmit)="onSubmit(f)" #f="ngForm"> -->
      <form (ngSubmit)="onSubmit()" #f="ngForm">
        <div ngModelGroup="userData">
        <div class="form-group">
          <label>User Name: </label>
          <input
            type="text"
            class="form-control"
            id="username"
            name="username"
            ngModel
            required
            #username = "ngModel"
          />
          <span
            class="help-text"
            *ngIf="
             username &&
              !username.valid &&
              username.touched
            "
            >Enter valid name</span
          >
        </div>
        <div class="form-group">
          <label for="email">Email : </label>
          <input
            type="text"
            class="form-control"
            id="email"
            name="email"
            ngModel
            required
            email
            #email = "ngModel"
          />
          <span
            class="help-text"
            *ngIf="
             email &&
              !email.valid &&
              email.touched
            "
            >Please enter valid email</span>
        </div>
```

```
    </div>
      <div class="form-group">
        <label>Gender</label>
        <select
          class="form-control"
          id="gender"
          name="gender"
          [ngModel]="gender"
        >
          <option value="male">Male</option>
          <option value="Female">Female</option>
        </select>
      </div>
      <div class="form-group">
        <label>About Yourself</label>
        <textarea
          class="form-control"
          rows="3"
          name="about"
          [(ngModel)]="about"
        ></textarea>
        <p>{{ about }}</p>
      </div>
      <div class="mt-2">
        <button type="submit" class="btn btn-primary" [disabled]="!f.valid">
          Submit
        </button>
      </div>
    </form>
  </div>
 </div>
</div>
```

Output



76. NgModelGroup - Grouping The Form Controls in Template Driven Forms using ngModelGroup in angular
If we want to group some inputs into a group, then such inputs we can wrap inside a div tag and use ngModelGroup attribute and assign some name to the group as shown below. Also we cab apply some sort of validation

App.component.html file

```
<div class="container">
  <div class="row">
    <div class="col-md-12">
      <!-- <form (ngSubmit)="onSubmit(f)" #f="ngForm"> -->
      <form (ngSubmit)="onSubmit()" #f="ngForm">
        <div>
          <span *ngIf="userData && !userData.valid && userData.touched">
            Enter valid user Data
          </span>
        </div>
        <div ngModelGroup="userData" #userData = "ngModelGroup">
          <div class="form-group">
            <label>User Name: </label>
            <input
              type="text"
              class="form-control"
              id="username"
              name="username"
              ngModel
```

```
        required
        #username = "ngModel"
      />
      <span
        class="help-text"
        *ngIf="
        username &&
          !username.valid &&
          username.touched
        "
        >Enter valid name</span
      >
    </div>
    <div class="form-group">
      <label for="email">Email : </label>
      <input
        type="text"
        class="form-control"
        id="email"
        name="email"
        ngModel
        required
        email
        #email = "ngModel"
      />
      <span
        class="help-text"
        *ngIf="
        email &&
          !email.valid &&
          email.touched
        "
        >Please enter valid email</span>
    </div>
  </div>

  <div class="form-group">
    <label>Gender</label>
    <select
      class="form-control"
      id="gender"
      name="gender"
      [ngModel]="gender"
    >
      <option value="male">Male</option>
      <option value="Female">Female</option>
    </select>
  </div>
```

```
      <div class="form-group">
        <label>About Yourself</label>
        <textarea
          class="form-control"
          rows="3"
          name="about"
          [(ngModel)]="about"
        ></textarea>
        <p>{{ about }}</p>
      </div>
      <div class="mt-2">
        <button type="submit" class="btn btn-primary" [disabled]="!f.valid">
          Submit
        </button>
      </div>
    </form>
  </div>
 </div>
</div>
```

77. Set Value and Patch Value for populating Form Elements in the Template Driven Forms in Angular.

Suppose if we want to fill the form with some data on a button click, App.component.html file

```
<div class="container">
 <div class="row">
  <div class="col-md-12">
   <!-- <form (ngSubmit)="onSubmit(f)" #f="ngForm"> -->
   <form (ngSubmit)="onSubmit()" #f="ngForm">
    <div>
     <button class="btn btn-sm btn-info" (click)="fillData()">Fill Data</button>
    </div>
    <div>
     <span *ngIf="userData && !userData.valid && userData.touched">
      Enter valid user Data
     </span>
    </div>
    <div ngModelGroup="userData" #userData = "ngModelGroup">
     <div class="form-group">
      <label>User Name: </label>
      <input
       type="text"
       class="form-control"
       id="username"
```

```
        name="username"
        ngModel
        required
        #username = "ngModel"
      />
      <span
        class="help-text"
        *ngIf="
        username &&
          !username.valid &&
          username.touched
        "
        >Enter valid name</span
      >
    </div>
    <div class="form-group">
      <label for="email">Email : </label>
      <input
        type="text"
        class="form-control"
        id="email"
        name="email"
        ngModel
        required
        email
        #email = "ngModel"
      />
      <span
        class="help-text"
        *ngIf="
        email &&
          !email.valid &&
          email.touched
        "
        >Please enter valid email</span>
    </div>
  </div>

  <div class="form-group">
    <label>Gender</label>
    <select
      class="form-control"
      id="gender"
      name="gender"
      [ngModel]="gender"
    >
      <option value="male">Male</option>
      <option value="Female">Female</option>
```

```
        </select>
      </div>
      <div class="form-group">
        <label>About Yourself</label>
        <textarea
          class="form-control"
          rows="3"
          name="about"
          [(ngModel)]="about"
        ></textarea>
        <p>{{ about }}</p>
      </div>
      <div class="mt-2">
        <button type="submit" class="btn btn-primary" [disabled]="!f.valid">
          Submit
        </button>
      </div>
    </form>
  </div>
 </div>
</div>
```

Now define the method is ts file
App.component.ts file

```
import { Component, ViewChild } from '@angular/core';
import { NgForm } from '@angular/forms';

@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrl: './app.component.css'
})
export class AppComponent {
 title = 'forms-demo-one';
 gender = 'male';
 about = '';
 @ViewChild('f') submittedForm:NgForm;
 // onSubmit(form:NgForm){
 //   console.log(form.value);
 //   console.log(form.value['username']);
 // }
```

```
 onSubmit(){
   // console.log(this.submittedForm.value);
   // console.log(this.submittedForm.value['username']);
   console.log(this.submittedForm)

 }
```

```
  fillData(){
    this.submittedForm.form.setValue({
      userData:{
        username:'test',
        email:'test@test.com'
      },
      gender:'male',
      about:'test about myself'
    });
  }
}
```

The setValue will take all the input fields as input, we cannot leave some fields empty

If we want to fill only some input fields then we can use patchValue

```
  fillData(){
    // this.submittedForm.form.setValue({
    //   userData:{
    //     username:'test',
    //     email:'test@test.com'
    //   },
    //   gender:'male',
    //   about:'test about myself'
    // });
    this.submittedForm.form.patchValue({
      userData:{
        email:'test@test.com'
      },
    });
  }
}
```

78. Get and Reset the Form Data controls in the Template Driven Forms in the Angular

App.component.ts file

```
import { Component, ViewChild } from '@angular/core';
import { NgForm } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'forms-demo-one';
```

```
  gender = 'male';
  about = '';
  submitted = false;
  user = {
    username:'',
    email:'',
    gender:'',
    about:''
  }
  @ViewChild('f') submittedForm:NgForm;
  // onSubmit(form:NgForm){
  //   console.log(form.value);
  //   console.log(form.value['username']);
  // }
  onSubmit(){
    // console.log(this.submittedForm.value);
    // console.log(this.submittedForm.value['username']);
    console.log(this.submittedForm)
    this.user.username = this.submittedForm.value.userData.username;
    this.user.email = this.submittedForm.value.userData.email;
    this.user.gender = this.submittedForm.value.gender;
    this.user.about = this.submittedForm.value.about;
    this.submitted = true;
    this.submittedForm.reset();
  }
  fillData(){
    // this.submittedForm.form.setValue({
    //   userData:{
    //     username:'test',
    //     email:'test@test.com'
    //   },
    //   gender:'male',
    //   about:'test about myself'
    // });
    this.submittedForm.form.patchValue({
      userData:{
        email:'test@test.com'
      },
    });
  }
}
```

App.component.html file

```
<div class="container">
  <div class="row">
    <div class="col-md-12">
```

```
<!-- <form (ngSubmit)="onSubmit(f)" #f="ngForm"> -->
<form (ngSubmit)="onSubmit()" #f="ngForm">
  <div>
    <button class="btn btn-sm btn-info" (click)="fillData()">Fill Data</button>
  </div>
  <div>
    <span *ngIf="userData && !userData.valid && userData.touched">
      Enter valid user Data
    </span>
  </div>
  <div ngModelGroup="userData" #userData = "ngModelGroup">
    <div class="form-group">
      <label>User Name: </label>
      <input
        type="text"
        class="form-control"
        id="username"
        name="username"
        ngModel
        required
        #username = "ngModel"
      />
      <span
        class="help-text"
        *ngIf="
        username &&
          !username.valid &&
          username.touched
        "
        >Enter valid name</span
      >
    </div>
    <div class="form-group">
      <label for="email">Email : </label>
      <input
        type="text"
        class="form-control"
        id="email"
        name="email"
        ngModel
        required
        email
        #email = "ngModel"
      />
      <span
        class="help-text"
        *ngIf="
        email &&
```

```
            !email.valid &&
            email.touched
          "
          >Please enter valid email</span>
        </div>
      </div>

      <div class="form-group">
        <label>Gender</label>
        <select
          class="form-control"
          id="gender"
          name="gender"
          [ngModel]="gender"
        >
          <option value="male">Male</option>
          <option value="Female">Female</option>
        </select>
      </div>
      <div class="form-group">
        <label>About Yourself</label>
        <textarea
          class="form-control"
          rows="3"
          name="about"
          [(ngModel)]="about"
        ></textarea>
        <p>{{ about }}</p>
      </div>
      <div class="mt-2">
        <button type="submit" class="btn btn-primary" [disabled]="!f.valid">
          Submit
        </button>
      </div>
    </form>
  </div>
 </div>
 <div class="row" *ngIf="submitted">
  <h3>Your Data</h3>
  <div>User Name :{{user.username}}</div>
  <div>User email :{{user.email}}</div>
  <div>User gender :{{user.gender}}</div>
  <div>About User :{{user.about}}</div>
 </div>
</div>
```

79. Introduction to Reactive Forms Approach. Create FormGroup and FormControl with code in Angular.
Ref : forms-demo-two

Create new project
Install bootstrap
Inorder to work with reactive forms - reactiveFormsModule should be added to the imports array in app.module.ts file

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ReactiveFormsModule } from '@angular/forms';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

App.component.html

```
<div class="container">
  <div class="row">
    <div class="col-md-12">
    <h3>Reactive Forms</h3>
    <hr>
    <form>
      <div class="form-group">
        <label>User Name: </label>
        <input
          type="text"
          class="form-control"
        />
      </div>
      <div class="form-group">
        <label>User Email: </label>
        <input
          type="text"
          class="form-control"
        />
      </div>
      <div *ngFor="let gender of genders">
        <label>
```

```
        <input type="radio" name="gender" [value]="gender">{{gender}}
      </label>
    </div>
    <div>
      <button class="btn btn-primary" type="submit">Add</button>
    </div>
    </form>
  </div>
 </div>
</div>
```

App.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup } from '@angular/forms';

@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrl: './app.component.css'
})
export class AppComponent implements OnInit {
 title = 'forms-demo-two';
 genders = ['male','female'];
 signUpForm: FormGroup;
 constructor(){}
 ngOnInit(){
  this.signUpForm = new FormGroup({
   'username': new FormControl(''),
   'email': new FormControl(''),
   'gender':new FormControl('male')
  });
 }
}
```

Above is the initial set up, still there is no connection between ts and html file

80. Attach the HTML File using the FormGroup with FormControlName using Reactive Forms in Angular.

The FormGroup variable created in ts file should be attached to the formGroup property in form tag in html file
All FormControl created in ts file should be mapped to all input fields in the html file

```
<div class="container">
 <div class="row">
  <div class="col-md-12">
```

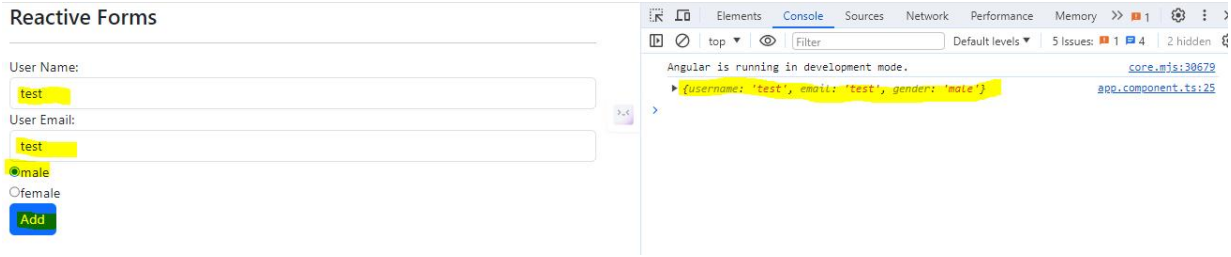
```
    <h3>Reactive Forms</h3>
    <hr>
    <form [formGroup]="signUpForm"  (ngSubmit)="onSubmit()">
      <div class="form-group">
        <label>User Name: </label>
        <input
          type="text"
          class="form-control"
          formControlName="username"
        />
      </div>
      <div class="form-group">
        <label>User Email: </label>
        <input
          type="text"
          class="form-control"
          formControlName="email"
        />
      </div>
      <div *ngFor="let gender of genders">
        <label>
          <input type="radio" name="gender" [value]="gender"
formControlName="gender">{{gender}}
        </label>
      </div>
      <div>
        <button class="btn btn-primary" type="submit">Add</button>
      </div>
      </form>
    </div>
  </div>
</div>
```

App.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent implements OnInit {
  title = 'forms-demo-two';
  genders = ['male','female'];
  signUpForm: FormGroup;
  constructor(){}
  ngOnInit(){
```

```
    this.signUpForm = new FormGroup({
      'username': new FormControl(''),
      'email': new FormControl(''),
      'gender':new FormControl('male')
    });
  }
  onSubmit(){
    console.log(this.signUpForm.value);
  }
}
```

Output:



81. Apply Validations for Reactive Forms and also show messages in the HTML Template - Angular.

If we want to add validations to input fields then it should be done in ts file App.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent implements OnInit {
  title = 'forms-demo-two';
  genders = ['male','female'];
  signUpForm: FormGroup;
  constructor(){}
  ngOnInit(){
    this.signUpForm = new FormGroup({
      'username': new FormControl('', Validators.required),
      'email': new FormControl('', [Validators.required, Validators.email]),
      'gender':new FormControl('male')
    });
  }
  onSubmit(){
    console.log(this.signUpForm.value);
```

```
  }
}
```

App.component.css file

```
input.ng-invalid.ng-touched{
border: 1px solid red;
}
```

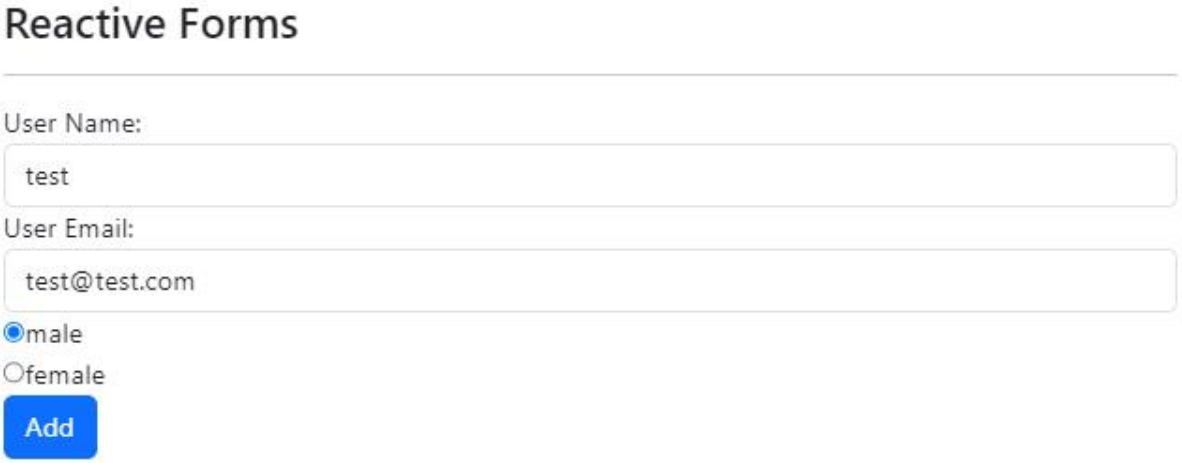
App.component.html file

```
<div class="container">
  <div class="row">
    <div class="col-md-12">
      <h3>Reactive Forms</h3>
      <hr />
      <form [formGroup]="signUpForm" (ngSubmit)="onSubmit()">
        <div class="form-group">
          <label>User Name: </label>
          <input type="text" class="form-control" formControlName="username" />
          <span
            class="help-block"
            *ngIf="
              !signUpForm.get('username').valid &&
              signUpForm.get('username').touched
            "
            >Please enter a valid user name</span
          >
        </div>
        <div class="form-group">
          <label>User Email: </label>
          <input type="text" class="form-control" formControlName="email" />
          <span
            class="help-block"
            *ngIf="
              !signUpForm.get('email').valid && signUpForm.get('email').touched
            "
            >Please enter a valid email</span
          >
        </div>
        <div *ngFor="let gender of genders">
          <label>
            <input
              type="radio"
              name="gender"
              [value]="gender"
              formControlName="gender"
            />{{ gender }}
          </label>
```

```
      </div>
      <div>
        <button class="btn btn-primary" type="submit"
[disabled]="!signUpForm.valid">Add</button>
      </div>
    </form>
  </div>
 </div>
</div>
```

Output:
Invalid case:

Valid case:



82. Grouping the Controls in the Reactive Forms using FormGroupName in FormGroup - Angular.

If we want to group username and email together, then do the following

App.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent implements OnInit {
  title = 'forms-demo-two';
  genders = ['male','female'];
  signUpForm: FormGroup;
  constructor(){}
  ngOnInit(){
    this.signUpForm = new FormGroup({
      'userData': new FormGroup(
        {
          'username': new FormControl('', Validators.required),
          'email': new FormControl('', [Validators.required, Validators.email])
        }
      ),
      'gender':new FormControl('male')
    });
  }
```

App.component.html file

```
<div class="container">
  <div class="row">
    <div class="col-md-12">
      <h3>Reactive Forms</h3>
      <hr />
      <form [formGroup]="signUpForm" (ngSubmit)="onSubmit()">
        <div formGroupName="userData">
          <div class="form-group">
            <label>User Name: </label>
            <input
              type="text"
              class="form-control"
              formControlName="username"
            />
            <span
              class="help-block"
              *ngIf="
                !signUpForm.get('userData.username').valid &&
                signUpForm.get('userData.username').touched
              "
              >Please enter a valid user name</span
            >
```

```
        </div>
        <div class="form-group">
          <label>User Email: </label>
          <input type="text" class="form-control" formControlName="email" />
          <span
            class="help-block"
            *ngIf="
              !signUpForm.get('userData.email').valid &&
              signUpForm.get('userData.email').touched
            "
            >Please enter a valid email</span
          >
        </div>
      </div>
      <div *ngFor="let gender of genders">
        <label>
          <input
            type="radio"
            name="gender"
            [value]="gender"
            formControlName="gender"
          />{{ gender }}
        </label>
      </div>
      <div>
        <button
          class="btn btn-primary"
          type="submit"
          [disabled]="!signUpForm.valid"
        >
          Add
        </button>
      </div>
    </form>
  </div>
 </div>
</div>
```

83. Dynamically Add Form Controls with FormArray FormArrayName in the Reactive Forms - Angular.

If we want to create some input fields dynamically (lets say we have a button called add hobbies on click of this a text input have to be displayed)

App.component.html

```
<div class="container">
 <div class="row">
  <div class="col-md-12">
```

```
<h3>Reactive Forms</h3>
<hr />
<form [formGroup]="signUpForm" (ngSubmit)="onSubmit()">
  <div formGroupName="userData">
    <div class="form-group">
      <label>User Name: </label>
      <input
        type="text"
        class="form-control"
        formControlName="username"
      />
      <span
        class="help-block"
        *ngIf="
          !signUpForm.get('userData.username').valid &&
          signUpForm.get('userData.username').touched
        "
        >Please enter a valid user name</span
      >
    </div>
    <div class="form-group">
      <label>User Email: </label>
      <input type="text" class="form-control" formControlName="email" />
      <span
        class="help-block"
        *ngIf="
          !signUpForm.get('userData.email').valid &&
          signUpForm.get('userData.email').touched
        "
        >Please enter a valid email</span
      >
    </div>
  </div>
  <div *ngFor="let gender of genders">
    <label>
      <input
        type="radio"
        name="gender"
        [value]="gender"
        formControlName="gender"
      />{{ gender }}
    </label>
  </div>
  <div class="my-2">
    <button class="btn btn-warning" (click)="onAddHobby()">Add Hobby</button>
  </div>
  <div>
    <button
```

```
          class="btn btn-primary"
          type="submit"
          [disabled]="!signUpForm.valid"
        >
          Add
        </button>
      </div>
    </form>
  </div>
 </div>
</div>
```

App.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { FormArray, FormControl, FormGroup, Validators } from '@angular/forms';

@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrl: './app.component.css'
})
export class AppComponent implements OnInit {
 title = 'forms-demo-two';
 genders = ['male','female'];
```

```
 signUpForm: FormGroup;
 constructor(){}
```

```
 ngOnInit(){
  this.signUpForm = new FormGroup({
   'userData': new FormGroup(
    {
     'username': new FormControl('', Validators.required),
     'email': new FormControl('', [Validators.required, Validators.email])
    }
   ),
   'gender':new FormControl('male'),
   'hobbies': new FormArray([])
  });
 }
 onSubmit(){
  console.log(this.signUpForm.value);
 }
```

```
 onAddHobby(){
  const control = new FormControl(null,[Validators.required]);
  (<FormArray>this.signUpForm.get('hobbies')).push(control);
```

```
  }
}
```

Above is the basic setup.
Now we need to add a getter method in the ts file inorder to return the hobbies control to the html file, so that we can dynamically create input fields

App.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { FormArray, FormControl, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent implements OnInit {
  title = 'forms-demo-two';
  genders = ['male','female'];
  signUpForm: FormGroup;
  constructor(){}

     get hobbyControls(){
    return (<FormArray>this.signUpForm.get('hobbies')).controls;
  }
  ngOnInit(){
    this.signUpForm = new FormGroup({
      'userData': new FormGroup(
        {
          'username': new FormControl('', Validators.required),
          'email': new FormControl('', [Validators.required, Validators.email])
        }
      ),
      'gender':new FormControl('male'),
      'hobbies': new FormArray([])
    });
  }
  onSubmit(){
    console.log(this.signUpForm.value);
  }
  onAddHobby(){
    const control = new FormControl(null,[Validators.required]);
    (<FormArray>this.signUpForm.get('hobbies')).push(control);
  }
}
```

Update app.component.html file as follows

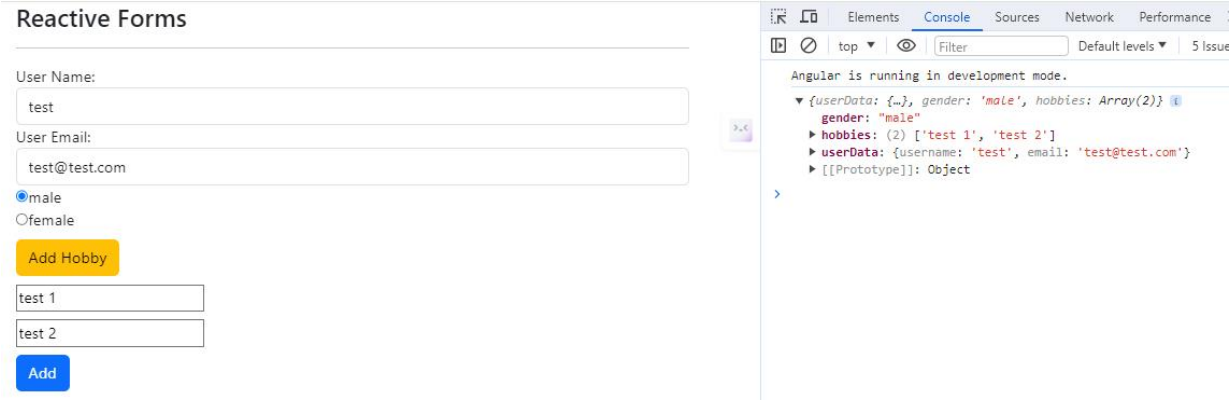
```
<div class="container">
```

```
<div class="row">
  <div class="col-md-12">
    <h3>Reactive Forms</h3>
    <hr />
    <form [formGroup]="signUpForm" (ngSubmit)="onSubmit()">
      <div formGroupName="userData">
        <div class="form-group">
          <label>User Name: </label>
          <input
            type="text"
            class="form-control"
            formControlName="username"
          />
          <span
            class="help-block"
            *ngIf="
              !signUpForm.get('userData.username').valid &&
              signUpForm.get('userData.username').touched
            "
            >Please enter a valid user name</span
          >
        </div>
        <div class="form-group">
          <label>User Email: </label>
          <input type="text" class="form-control" formControlName="email" />
          <span
            class="help-block"
            *ngIf="
              !signUpForm.get('userData.email').valid &&
              signUpForm.get('userData.email').touched
            "
            >Please enter a valid email</span
          >
        </div>
      </div>
      <div *ngFor="let gender of genders">
        <label>
          <input
            type="radio"
            name="gender"
            [value]="gender"
            formControlName="gender"
          />{{ gender }}
        </label>
      </div>
      <div formArrayName="hobbies">
        <div class="my-2">
          <button type = "button"class="btn btn-warning" (click)="onAddHobby()">
```

```
          Add Hobby
        </button>
      </div>
      <div
        class="form-group my-2"
        *ngFor="let hobby of hobbyControls; let i = index"
      >
        <input type="text" [formControlName]="i" />
      </div>
    </div>
    <div>
      <button
        class="btn btn-primary"
        type="submit"
        [disabled]="!signUpForm.valid"
      >
        Add
      </button>
    </div>
  </form>
 </div>
</div>
</div>
```

Note: for add hobby button give type=”button” not type=”submit” because if we use submit then on each click data will be submitted

Output:



.

84. Create Custom Validations for the reactive Forms in the Angular.

If we want to have our own validations (lets say we don’t want certain names to be entered as username)

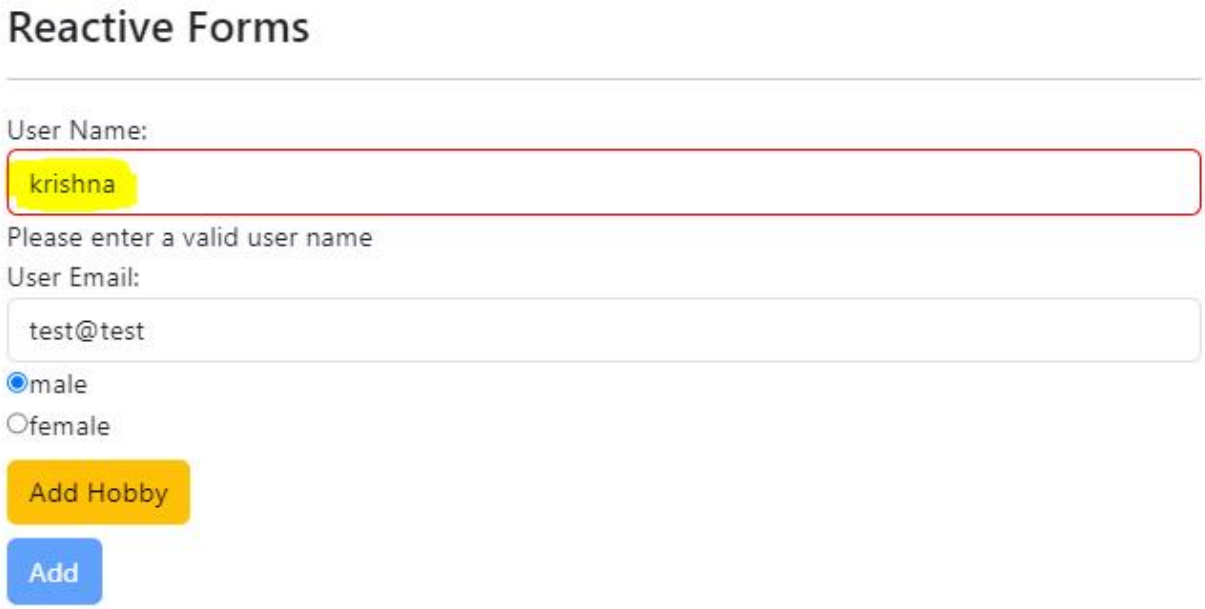
App.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { FormArray, FormControl, FormGroup, Validators } from '@angular/forms';
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent implements OnInit {
  title = 'forms-demo-two';
  genders = ['male','female'];
  signUpForm: FormGroup;

  restrictedNames = ['krishna'];

  constructor(){}

  isRestrictedNames(control:FormControl):{[s:string]:boolean}{
    if(this.restrictedNames.includes(control.value)){
      return {isNameRestricted:true};
    }
    return null;
  }

  get hobbyControls(){
    return (<FormArray>this.signUpForm.get('hobbies')).controls;
  }

  ngOnInit(){
    this.signUpForm = new FormGroup({
      'userData': new FormGroup(
        {
          'username': new FormControl('', [Validators.required,
this.isRestrictedNames.bind(this)]),
          'email': new FormControl('', [Validators.required, Validators.email])
        }
      ),
      'gender':new FormControl('male'),
      'hobbies': new FormArray([])
    });
  }

  onSubmit(){
    console.log(this.signUpForm.value);
  }

  onAddHobby(){
    const control = new FormControl(null,[Validators.required]);
    (<FormArray>this.signUpForm.get('hobbies')).push(control);
  }
```

```
}
```

Output:



If we want to show custom error message (say : if user does not enter any name then different error message and if user enters restricted name then different error message)

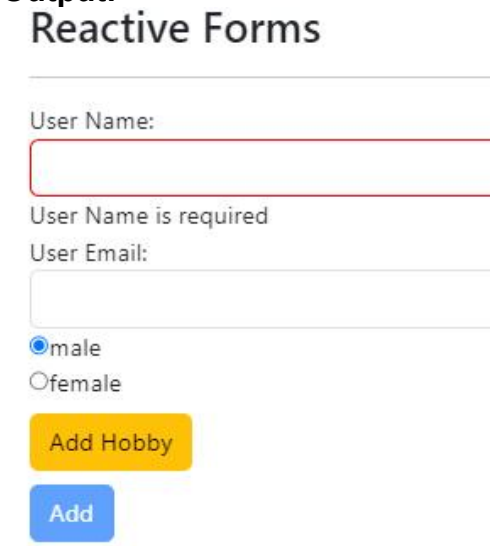
App.component.html file

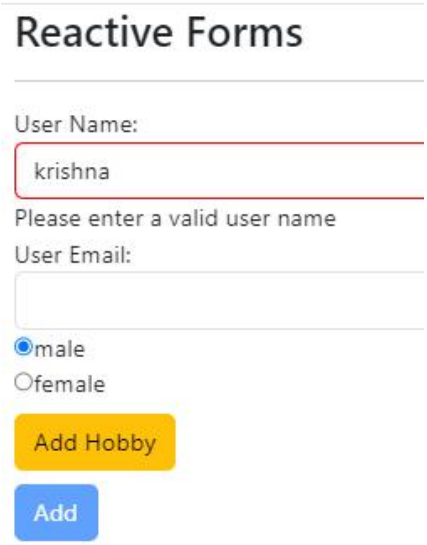
```
<div class="container">
  <div class="row">
    <div class="col-md-12">
      <h3>Reactive Forms</h3>
      <hr />
      <form [formGroup]="signUpForm" (ngSubmit)="onSubmit()">
        <div formGroupName="userData">
          <div class="form-group">
            <label>User Name: </label>
            <input
              type="text"
              class="form-control"
              formControlName="username"
            />
            <span
              class="help-block"
              *ngIf="
                !signUpForm.get('userData.username').valid &&
                signUpForm.get('userData.username').touched
              "
              ><span *ngIf="signUpForm.get('userData.username').errors['required']">User
Name is required</span>
```

```
          <span
*ngIf="signUpForm.get('userData.username').errors['isNameRestricted']">Please enter
a valid user name</span>
          </span>
        </div>
        <div class="form-group">
          <label>User Email: </label>
          <input type="text" class="form-control" formControlName="email" />
          <span
            class="help-block"
            *ngIf="
              !signUpForm.get('userData.email').valid &&
              signUpForm.get('userData.email').touched
            "
            >Please enter a valid email</span
          >
        </div>
      </div>
      <div *ngFor="let gender of genders">
        <label>
          <input
            type="radio"
            name="gender"
            [value]="gender"
            formControlName="gender"
          />{{ gender }}
        </label>
      </div>
      <div formArrayName="hobbies">
        <div class="my-2">
          <button
            type="button"
            class="btn btn-warning"
            (click)="onAddHobby()"
          >
            Add Hobby
          </button>
        </div>
        <div
          class="form-group my-2"
          *ngFor="let hobby of hobbyControls; let i = index"
        >
          <input type="text" [formControlName]="i" />
        </div>
      </div>
      <div>
        <button
          class="btn btn-primary"
```

```
          type="submit"
          [disabled]="!signUpForm.valid"
        >
          Add
        </button>
      </div>
    </form>
  </div>
 </div>
</div>
```

Output:





85. Create a Custom Asynchronous Validator in the Reactive Forms - Angular

Full Stack application

Ref : e-commerce-project
Backend
Front end
Create a new folder frontend
Inside create a new angular project angular-ecommerce
Ng new angular-ecommerce --no-standalone
Install bootstrap
Npm install bootstrap
Create a component product-list under components folder (ng g c components/product-list)
Initial app.component.html file

```
<div class="container">
  <div class="row">
    <div class="col-md-12 mt-3">
      <h3>Products</h3>
      <app-product-list></app-product-list>
    </div>
  </div>
</div>
```

Create a class Product under common folder (ng generate class common/product)

Product.ts file

```
export class Product {

    constructor(
        public sku:string,
        public name:string,
        public description:string,
        public unitPrice:number,
        public imageUrl:string,
        public active:boolean,
        public unitsInStock:number,
        public dateCreated:Date,
        public lastUpdated:Date
    ){
    }
}
```

Create a new service Product under services (ng generate service services/product)
Inorder to use http we need to import HttpClientModule in app.module.ts file in imports array
Also we need to import ProductService to providers array

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

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

```
import { ProductListComponent } from './components/product-list/product-
list.component';
import { HttpClientModule } from '@angular/common/http';
import { ProductService } from './services/product.service';

@NgModule({
  declarations: [
    AppComponent,
    ProductListComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [ProductService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Product.service.ts file

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable, map } from 'rxjs';
import { Product } from '../common/product';

@Injectable({
  providedIn: 'root'
})
export class ProductService {
  private baseUrl = 'http://localhost:8080/api/products';
  constructor(private httpClient:HttpClient) { }
  getProductList():Observable<Product[]>{
     return this.httpClient.get<GetResponse>(this.baseUrl).pipe(map(response =>
response._embedded.products));
  }
}
interface GetResponse{
  _embedded:{
    products:Product[];
  }
}
```

Now in product-list.component.ts do the following

```
import { Component, OnInit } from '@angular/core';
import { ProductService } from '../../services/product.service';
import { Product } from '../../common/product';
```

```
@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html',
  styleUrl: './product-list.component.css'
})
export class ProductListComponent implements OnInit{
  products:Product[]=[];
  constructor(private productService:ProductService){}
  ngOnInit(): void {
    this.listProducts();
  }
  listProducts(){
    this.productService.getProductList().subscribe(
      data =>{
        this.products = data;
      }
    )
  }
}
```

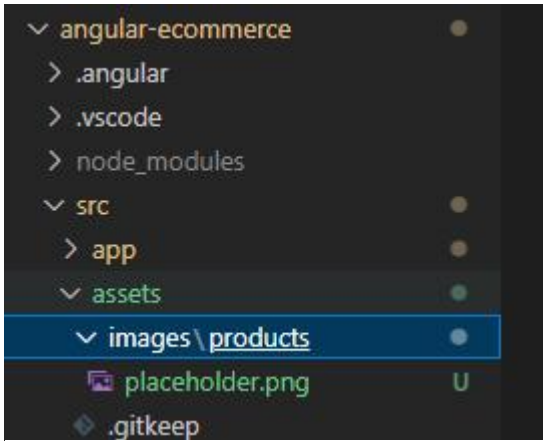
Display the products in html page - product-list.component.html file

```
<p *ngFor="let product of products">

   {{product.name}}:{{product.unitPrice | currency:'INR'}}
</p>
```

Display the products as a table --> product-list.component.html file

```
<table class="table table-striped">
    <thead class="thead-dark">
    <tr>
        <th>Name</th>
        <th>Unit Price</th>
        <th>Quantity</th>
    </tr>
</thead>
<tbody>
    <tr *ngFor="let product of products">
    <td> {{product.name}}</td>
    <td> {{product.unitPrice | currency:'INR'}}</td>
    <td> {{product.unitsInStock}}</td>
    </tr>
</tbody>
</table>
```

Adding Images
Inside src we have a folder called assets inside which create a folder images inside another folder products and have an image with the name placeholder.png



Update the product-list.component.html file as below

```
<table class="table table-striped">
   <thead class="thead-dark">
   <tr>
      <th></th>
      <th>Name</th>
      <th>Unit Price</th>
      <th>Quantity</th>
   </tr>
</thead>
<tbody>
   <tr *ngFor="let product of products">
      <td>
         <img src="{{product.imageUrl}}" height="50">
      </td>
   <td> {{product.name}}</td>
   <td> {{product.unitPrice | currency:'INR'}}</td>
   <td> {{product.unitsInStock}}</td>
   </tr>
</tbody>
</table>
```

Output

Products

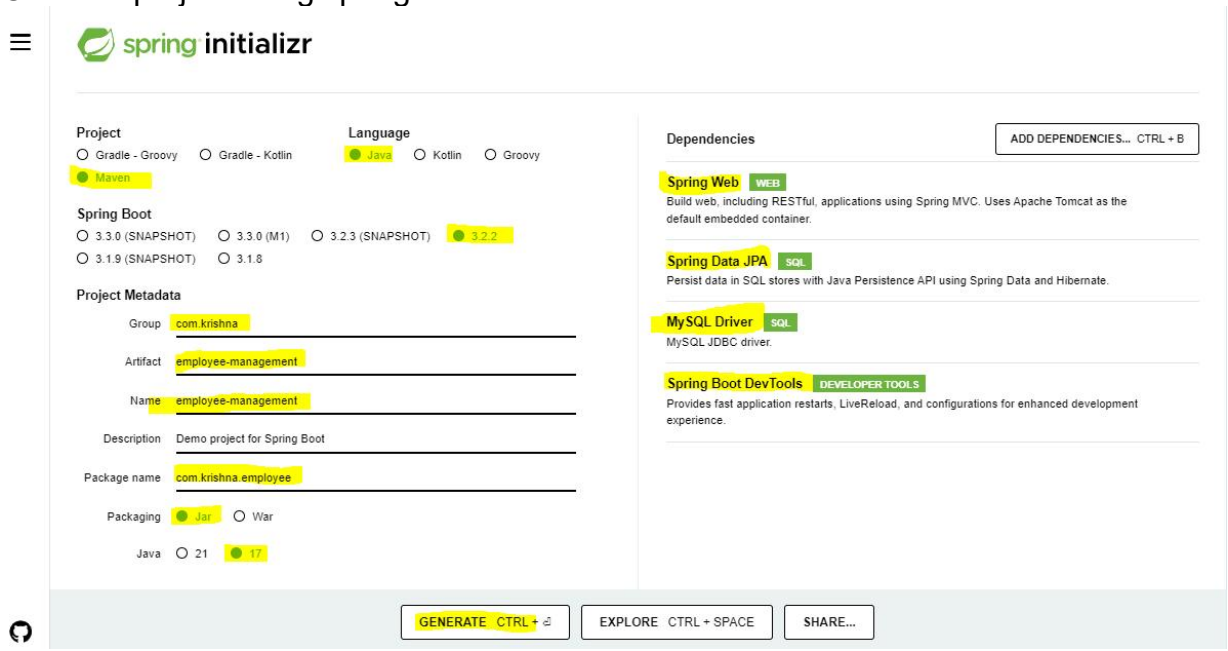
	Name	Unit Price	Quantity
	JavaScript - The Fun Parts	₹19.99	100
	Kubernetes - Deploying Containers	₹24.99	100
	Internet of Things (IoT) - Getting Started	₹29.99	100
	The Go Programming Language: A to Z	₹24.99	100
	Kubernetes - Deploying Containers	₹24.99	100

Full Stack application

Create a database called employee_db in mysql
No need to create the table, with below setting when we run the app table will be automatically created

Ref: D:/learning/angularyoutube/employee-management
Techs: Spring, Hibernate
Backend

Create a project using spring initializr



Extract the downloaded file and copy it to backend folder

Open eclipse and import maven project -> existing maven project -> select the folder

Edit the application.properties file as below

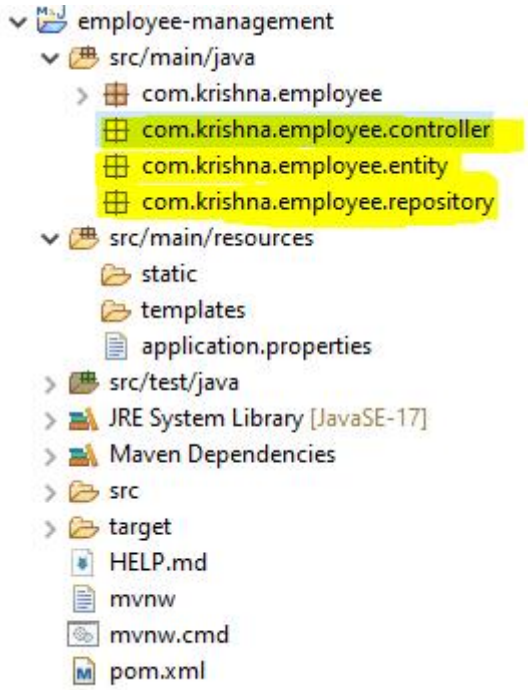
To get dialet value, open EmployeeManagementApllication.java file press ctrl + shift + t, search mysqlDialet, open the file, copy the package name and class name

application.properties

spring.datasource.url = jdbc:mysql://localhost:3306/employee_db
spring.datasource.username = root
spring.datasource.password = Novigo@123
spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto = update
spring.jpa.show_sql = true

Create the following packages -> controller, repository, entity



Create a class Employee under entity package

```
package com.krishna.employee.entity;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name="employee")
public class Employee {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;
	@Column(name="first_name")
	private String firstName;
	@Column(name="last_name")
	private String lastName;
	@Column(name="salary")
	private double salary;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
```

```
	}
	public String getFirstName() {
		return firstName;
	}
	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}
	public String getLastName() {
		return lastName;
	}
	public void setLastName(String lastName) {
		this.lastName = lastName;
	}
	public double getSalary() {
		return salary;
	}
	public void setSalary(double salary) {
		this.salary = salary;
	}

	public Employee() {
		super();
	}
	public Employee(int id, String firstName, String lastName, double salary) {
		super();
		this.id = id;
		this.firstName = firstName;
		this.lastName = lastName;
		this.salary = salary;
	}
}
```

Create an EmployeeRepository interface inside repository folder

```
package com.krishna.employee.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.krishna.employee.entity.Employee;

@Repository
@CrossOrigin(origins = "http://localhost:4200")
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

Run the app
EmployeeManagementApp.java --> right click and run as java app.
Verify employee table is created in the database

Create a EmployeeController inside controller folder

```
package com.krishna.employee.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import com.krishna.employee.entity.Employee;
import com.krishna.employee.repository.EmployeeRepository;

@RestController
public class EmployeeController {

	@Autowired
	private EmployeeRepository empRepo;

	@CrossOrigin(origins = "http://localhost:4200")
	@GetMapping("/employees")
	public List<Employee>getAllEmployess(){

		return empRepo.findAll();
	}
}
```

Note: Since we are using JPA we need not write any sql queries inoder to fetch the data, JpaRepository will provide this by default

Insert some values to the employee table manually

Run the app and check either in postman (Get - http://localhost:8080/employees)
or in browser -> http://localhost:8080/employees

Frontend

Create a folder frontend inside empolyee-management
Create a new angular project (ng new employee-management --no-standalone)
Install bootstrap (npm install bootstrap)

Create a class employee inside common folder (ng g class common/employee)

```
export class Employee {
```

```
  id:number = 0;
  firstName:string='';
  lastName:string ='';
  salary:number = 0;
}
```

Variables defined in this class and in the entity class of the back end should be exactly same

Create a service employee inside services folder (ng g s services/employee)

Inorder to use http we need to import HttpClientModule in app.module.ts file in imports array
Also we need to import EmployeeService to providers array
App.module.ts file

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { EmployeeListComponent } from './components/employee-list/employee-list.component';

import { HttpClientModule } from '@angular/common/http';
import { EmployeeService } from './services/employee.service';

@NgModule({
  declarations: [
    AppComponent,
    EmployeeListComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [EmployeeService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Employee.service.ts file

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

```
import { Observable } from 'rxjs';
import { Employee } from '../common/employee';

@Injectable({
  providedIn: 'root'
})
export class EmployeeService {
  private baseUrl = 'http://localhost:8080/employees';
  constructor(private httpClient:HttpClient) { }
  getEmployeeList():Observable<Employee[]>{
      return this.httpClient.get<Employee[]>(`${this.baseUrl}`);
      //return this.httpClient.get<Employee[]>(this.baseUrl); // this is also ok
  }
}
```

Create a component employee-list inside components folder (ng g c components/employee-list)

Employee-list.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { Employee } from '../../common/employee';
import { EmployeeService } from '../../services/employee.service';

@Component({
  selector: 'app-employee-list',
  templateUrl: './employee-list.component.html',
  styleUrl: './employee-list.component.css'
})
export class EmployeeListComponent implements OnInit {

  employees:Employee[]=[];

  constructor(private employeeService:EmployeeService){}

 ngOnInit(): void {
    this.getEmployeeList();
  }

  private getEmployeeList(){
    this.employeeService.getEmployeeList().subscribe(data=>{
      this.employees=data;
    });
  }
}
```

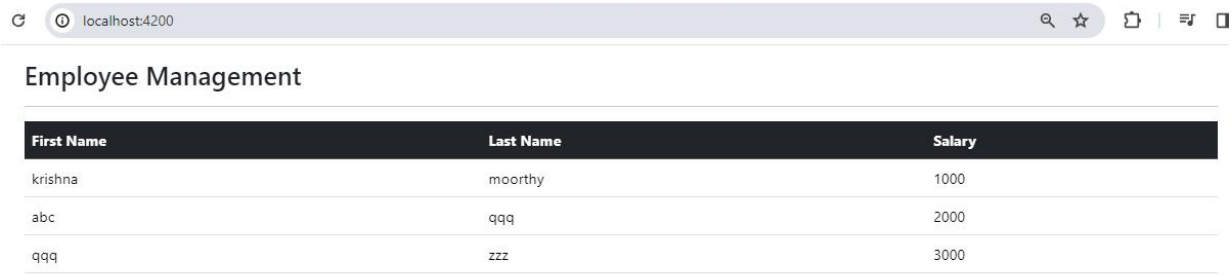
Employee-list.component.html file

```
<table class="table table-stripped">
  <thead class="table-dark ">
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Salary</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let employee of employees">
    <td>{{employee.firstName}}</td>
    <td>{{employee.lastName}}</td>
    <td>{{employee.salary}}</td>
    </tr>
  </tbody>
</table>
```

App.component.html file

```
<div class="container">
 <div class="row">
  <div class="col-md-12 mt-3">
    <h3>Employee Management</h3>
    <hr>
    <app-employee-list></app-employee-list>
  </div>
 </div>
</div>
```

Output:
Backend project should be up and running



Adding routing
App-routing.module.ts

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

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

const routes: Routes = [
  {path:'employees', component:EmployeeListComponent},
  {path:'', redirectTo:'employees',pathMatch:'full'}
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

App.component.html file

```
<div class="container">
  <div class="row">
    <div class="col-md-12 mt-3">
      <h3>Employee Management</h3>
      <hr>
      <router-outlet></router-outlet>
    </div>
  </div>
</div>
```

Adding nav bar to app.component.html

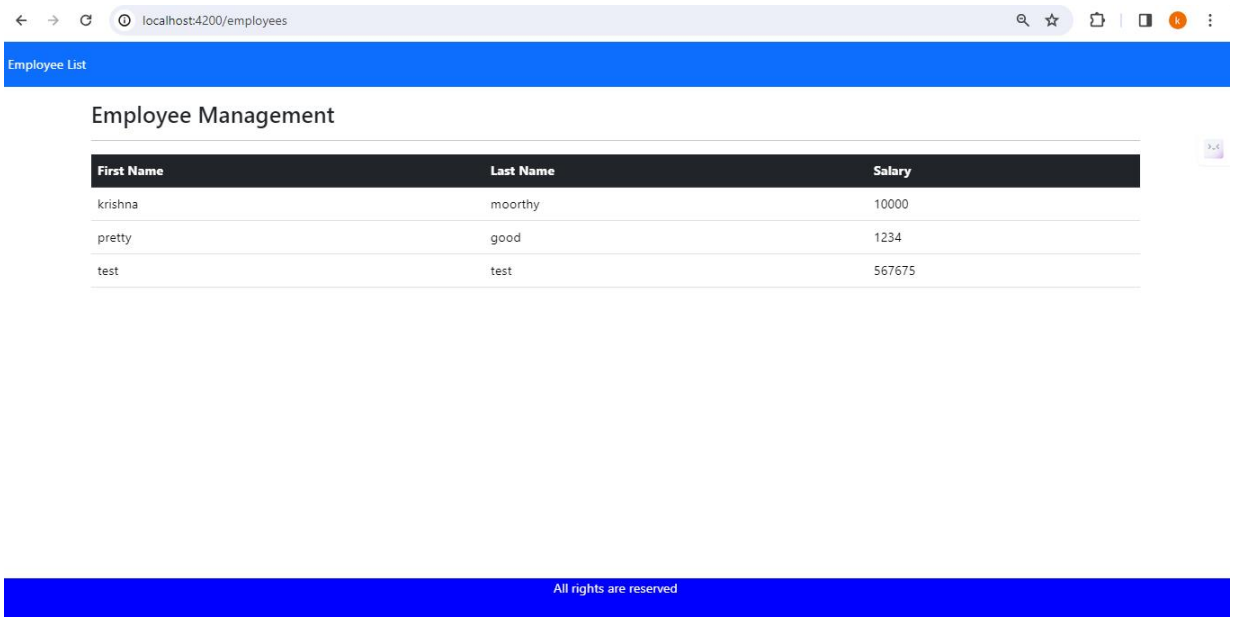
```
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
  <ul class="navbar-nav ">
    <li class="nav-item ">
      <a routerLink="employees" routerLinkActive="active" class="nav-link ">Employee
List</a>
    </li>
  </ul>
</nav>
<div class="container">
  <div class="row">
    <div class="col-md-12 mt-3">
      <h3>Employee Management</h3>
      <hr>
      <router-outlet></router-outlet>
    </div>
  </div>
</div>
<footer class="footer">
<div class="container">
  <span>All rights are reserved</span>
</div>
```

```
</footer>
```

Style.css

```
.footer{
    position: absolute;
    bottom: 0;
    width: 100%;
    height: 50px;
    background-color: blue;
    color: white;
    text-align: center;
}
```

Output:



Adding new employee
Back end
Add a new method in employee controller file

```
package com.krishna.employee.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
```

```
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.krishna.employee.entity.Employee;
import com.krishna.employee.repository.EmployeeRepository;

@RestController
@CrossOrigin(origins = "http://localhost:4200")
public class EmployeeController {

	@Autowired
	private EmployeeRepository empRepo;
	@GetMapping("/employees")
	public List<Employee>getAllEmployess(){

		return empRepo.findAll();
	}

	@PostMapping("/employees")
	public Employee createEmployee(@RequestBody Employee employee) {

		return empRepo.save(employee);

	}
}
```

Front end

Add a method to save the employee in the employee.service.ts file

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { Employee } from '../common/employee';

@Injectable({
  providedIn: 'root'
})
export class EmployeeService {

  private baseUrl = 'http://localhost:8080/employees';

  constructor(private httpClient:HttpClient) { }

  getEmployeeList():Observable<Employee[]>{
    return this.httpClient.get<Employee[]>(`${this.baseUrl}`);
  }

  addEmployee(employee:Employee):Observable<Object>{
```

```
    return this.httpClient.post(`${this.baseUrl}`,employee);
    // return this.httpClient.post(this.baseUrl,employee); // this is also ok
  }
}
```

Create a new component add-employee inside components folder

Add path in the app-routig.module.ts file

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { EmployeeListComponent } from './components/employee-list/employee-
list.component';
import { AddEmployeeComponent } from './components/add-employee/add-
employee.component';

const routes: Routes = [
  {path:'employees', component:EmployeeListComponent},
{path:'add-employee', component:AddEmployeeComponent},
  {path:'', redirectTo:'employees',pathMatch:'full'}, // better write this at the end of all
paths

];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Update the nav bar in app.component.html file

```
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
  <ul class="navbar-nav ">
    <li class="nav-item ">
      <a routerLink="employees" routerLinkActive="active" class="nav-link ">Employee
List</a>
    </li>
    <li class="nav-item ">
      <a routerLink="add-employee" routerLinkActive="active" class="nav-link ">Add
Employee</a>
    </li>
  </ul>
</nav>
<div class="container">
  <div class="row">
    <div class="col-md-12 mt-3">
      <h3>Employee Management</h3>
      <hr>
      <router-outlet></router-outlet>
```

```
    </div>
  </div>
</div>
<footer class="footer">
<div class="container">
  <span>All rights are reserved</span>
</div>
</footer>
```

Since we need to work with forms, import FormsModule in app.module.ts file

Add-employee.component.ts file

```
import { Component } from '@angular/core';
import { Employee } from '../../common/employee';
import { EmployeeService } from '../../services/employee.service';

@Component({
  selector: 'app-add-employee',
  templateUrl: './add-employee.component.html',
  styleUrl: './add-employee.component.css'
})
export class AddEmployeeComponent {

  employee:Employee = new Employee();

  constructor(private employeeService:EmployeeService){}

  onSubmit(){
    this.addEmployee();
  }

  addEmployee(){
    //this.employeeService.addEmployee(this.employee).subscribe() // this is enough
    this.employeeService.addEmployee(this.employee).subscribe(data=>{
      console.log(data);
    });
  }
}
```

Design the form in add-employee.component.html file

```
<div class="container">
    <div class="row">
      <div class="col-md-8 offset-md-3">
        <h3>Add Employee</h3>
        <form (ngSubmit)="onSubmit()" #f="ngForm">
            <div class="form-group">
```

```
          <label>First Name: </label>
          <input
            type="text"
            class="form-control"
            id="firstName"
            name="firstName"
            [(ngModel)]="employee.firstName"
            #firstName = "ngModel"
            required
          />
          <span *ngIf="firstName && !firstName.valid && firstName.touched">Enter first
Name</span>
        </div>
        <div class="form-group">
          <label >Last Name : </label>
          <input
            type="text"
            class="form-control"
            id="lastName"
            name="lastName"
            [(ngModel)]="employee.lastName"
            #lastName = "ngModel"
            required
          />
          <span *ngIf="lastName && !lastName.valid && lastName.touched">Enter last
Name</span>
        </div>

       <div class="form-group">
        <label>Salary :</label>
        <input
          type="text"
          class="form-control"
          id="salary"
          name="salary"
          [(ngModel)]="employee.salary"
        />
       </div>
       <div class="mt-2">
        <button type="submit" class="btn btn-primary" [disabled]="!f.valid">
          Submit
        </button>
       </div>
      </form>
    </div>
  </div>
 </div>
```

Add-employee.component.css file

```
input.ng-invalid.ng-touched{
    border: 1px solid red;
}
```

If we want to redirect to employee list page soon after adding a new employee then,

```
import { Component } from '@angular/core';
import { Employee } from '../../common/employee';
import { EmployeeService } from '../../services/employee.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-add-employee',
  templateUrl: './add-employee.component.html',
  styleUrl: './add-employee.component.css'
})
export class AddEmployeeComponent {

  employee:Employee = new Employee();

  constructor(private employeeService:EmployeeService, private router:Router){}

  onSubmit(){
    this.addEmployee();
  }

  addEmployee(){
    //this.employeeService.addEmployee(this.employee).subscribe() // this is enough
    this.employeeService.addEmployee(this.employee).subscribe(data=>{
      console.log(data);
      this.router.navigateByUrl('employees');
    });
  }
}
```

Updating employee

Back end

Create a package exception
Create a class ResourceNotFoundException

```
package com.krishna.employee.exception;
```

```java
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value=HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {

	public ResourceNotFoundException(String message) {
		super(message);
	}
}
```

Update the employeeController as follows
First we need to get the employee by id inorder to fill the details of the employee in the front end
Then we need to update the employee details

```java
package com.krishna.employee.controller;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.krishna.employee.entity.Employee;
import com.krishna.employee.exception.ResourceNotFoundException;
import com.krishna.employee.repository.EmployeeRepository;

@RestController
@CrossOrigin(origins = "http://localhost:4200")
public class EmployeeController {

	@Autowired
	private EmployeeRepository empRepo;
	@CrossOrigin(origins = "http://localhost:4200")
	@GetMapping("/employees")
	public List<Employee>getAllEmployess(){

		return empRepo.findAll();
	}
```

```
	@PostMapping("/employees")
	public Employee createEmployee(@RequestBody Employee employee) {

		return empRepo.save(employee);

	}

	@GetMapping("/employees/{id}")
	public ResponseEntity<Employee> getEmployeeById(@PathVariable int id) {

	Employee emp = empRepo.findById(id).orElseThrow(
				()-> new ResourceNotFoundException("Employee not
found with id"+id));

		return ResponseEntity.ok(emp);
	}

	@PutMapping("/employees/{id}")
	public ResponseEntity<Employee> updateEmployee( @PathVariable int id,
@RequestBody Employee employee) {
		Employee emp = empRepo.findById(id).orElseThrow(
				()-> new ResourceNotFoundException("Employee not
found with id"+id));

		emp.setFirstName(employee.getFirstName());
		emp.setLastName(employee.getLastName());
		emp.setSalary(employee.getSalary());
		empRepo.save(emp);
		return ResponseEntity.ok(emp);
	}
}
```

Front end

Create a component update-employee

Update app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { EmployeeListComponent } from './components/employee-list/employee-
list.component';
import { TestComponent } from './test/test.component';
import { AddEmployeeComponent } from './components/add-employee/add-
employee.component';
import { UpdateEmployeeComponent } from './components/update-employee/update-
employee.component';

const routes: Routes = [
```

```
  {path:'employees', component:EmployeeListComponent},
  {path:'add-employee', component:AddEmployeeComponent},
  {path:'update-employee/:id', component:UpdateEmployeeComponent},
  {path:'', redirectTo:'employees',pathMatch:'full'},

];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Update employee-list.component.html file

```
<table class="table table-stripped">
  <thead class="table-dark">
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Salary</th>
      <th>Update/Delete</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let employee of employees">
      <td>{{ employee.firstName }}</td>
      <td>{{ employee.lastName }}</td>
      <td>{{ employee.salary }}</td>
      <td>
        <button
          type="button"
          class="btn btn-primary"
          (click)="updateEmployee(employee.id)"
        >
          Update
        </button>
      </td>
    </tr>
  </tbody>
</table>
```

Update employee-list.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { Employee } from '../../common/employee';
import { EmployeeService } from '../../services/employee.service';
import { Router } from '@angular/router';

@Component({
```

```
  selector: 'app-employee-list',
  templateUrl: './employee-list.component.html',
  styleUrl: './employee-list.component.css'
})
export class EmployeeListComponent implements OnInit {
  employees:Employee[]=[];
  constructor(private employeeService:EmployeeService, private router:Router){}
  ngOnInit(): void {
    this.getEmployeeList();
  }
  private getEmployeeList(){
    this.employeeService.getEmployeeList().subscribe(data=>{
      this.employees=data;
    });
  }

  updateEmployee(id:number){
    this.router.navigate(['update-employee',id]);
  }
}
```

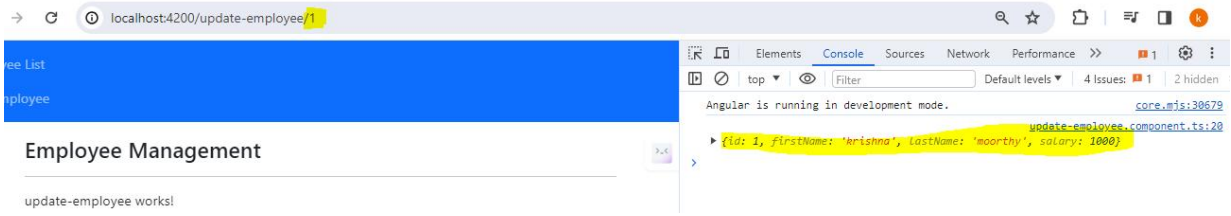
Update employee.service.ts file

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { Employee } from '../common/employee';

@Injectable({
  providedIn: 'root'
})
export class EmployeeService {
  private baseUrl = 'http://localhost:8080/employees';
  constructor(private httpClient:HttpClient) { }
  getEmployeeList():Observable<Employee[]>{
   // return this.httpClient.get<Employee[]>(`${this.baseUrl}`);
    return this.httpClient.get<Employee[]>(this.baseUrl);
  }
  addEmployee(employee:Employee):Observable<Object>{
  //  return this.httpClient.post(`${this.baseUrl}`,employee);
    return this.httpClient.post(this.baseUrl,employee);
  }

  getEmployeeById(id:number):Observable<Employee>{
    return this.httpClient.get<Employee>(`${this.baseUrl}/${id}`);
  }
}
```

Update update-employee.componet.ts file

```
import { Component, OnInit } from '@angular/core';
import { EmployeeService } from '../../services/employee.service';
import { ActivatedRoute } from '@angular/router';
import { Employee } from '../../common/employee';

@Component({
  selector: 'app-update-employee',
  templateUrl: './update-employee.component.html',
  styleUrl: './update-employee.component.css'
})
export class UpdateEmployeeComponent implements OnInit{
  id:number=0;
  employee:Employee;
  constructor(private employeeService:EmployeeService, private
route:ActivatedRoute){}
  ngOnInit(): void {
   this.id =  this.route.snapshot.params['id'];
    this.employeeService.getEmployeeById(this.id).subscribe(data=>{
     this.employee = data;
     console.log(this.employee);
    });
  }
}
```

If we check the output we will get the following



Update update-employee.component.html as below (same as that of add-employee.component.html file)

```
<div class="container">
   <div class="row">
     <div class="col-md-8 offset-md-3">
       <h3>Update Employee</h3>
       <form (ngSubmit)="onSubmit()" #f="ngForm">
          <div class="form-group">
            <label>First Name: </label>
            <input
              type="text"
              class="form-control"
              id="firstName"
```

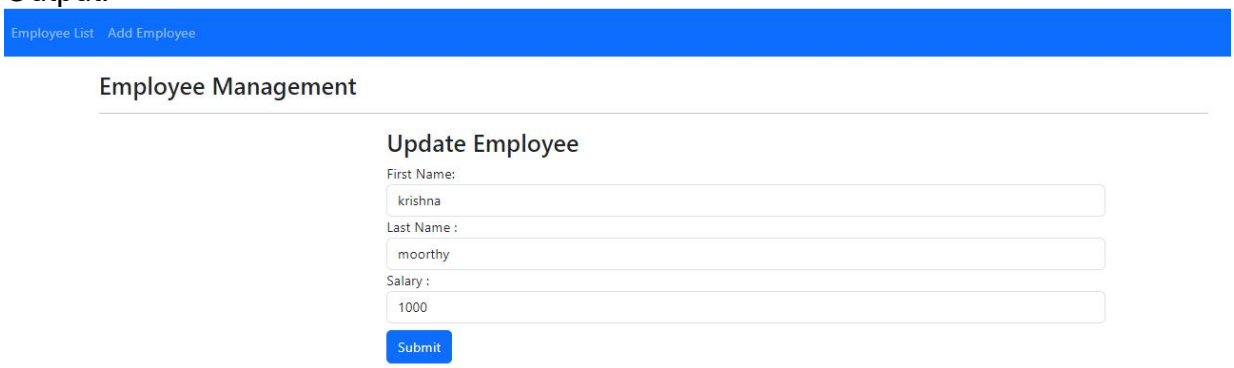
```
            name="firstName"
            [(ngModel)]="employee.firstName"
            #firstName = "ngModel"
            required
           />
           <span *ngIf="firstName && !firstName.valid && firstName.touched">Enter first
Name</span>
          </div>
          <div class="form-group">
           <label >Last Name : </label>
           <input
            type="text"
            class="form-control"
            id="lastName"
            name="lastName"
            [(ngModel)]="employee.lastName"
            #lastName = "ngModel"
            required
           />
           <span *ngIf="lastName && !lastName.valid && lastName.touched">Enter last
Name</span>
          </div>

         <div class="form-group">
          <label>Salary :</label>
          <input
            type="text"
            class="form-control"
            id="salary"
            name="salary"
            [(ngModel)]="employee.salary"
           />
         </div>
         <div class="mt-2">
          <button type="submit" class="btn btn-primary" [disabled]="!f.valid">
           Submit
          </button>
         </div>
        </form>
      </div>
    </div>
  </div>
```

Define onSubmit() method in update-employee.component.ts file and leave it blank for the time being

Update update-employee.component.css as below

```
input.ng-invalid.ng-touched{
    border: 1px solid red;
}
```

Output:



Now we need to write the code for updating the employee details

Update employee.service.ts as below.

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { Employee } from '../common/employee';

@Injectable({
  providedIn: 'root'
})
export class EmployeeService {
  private baseUrl = 'http://localhost:8080/employees';
  constructor(private httpClient:HttpClient) { }
  getEmployeeList():Observable<Employee[]>{
   // return this.httpClient.get<Employee[]>(`${this.baseUrl}`);
    return this.httpClient.get<Employee[]>(this.baseUrl);
  }
  addEmployee(employee:Employee):Observable<Object>{
  //  return this.httpClient.post(`${this.baseUrl}`,employee);
    return this.httpClient.post(this.baseUrl,employee);
  }
  getEmployeeById(id:number):Observable<Employee>{
    return this.httpClient.get<Employee>(`${this.baseUrl}/${id}`);
  }

  updateEmployee(id:number, employee:Employee):Observable<Employee>{
    return this.httpClient.put<Employee>(`${this.baseUrl}/${id}`,employee);
  }
```

```
}
```

Update update-employee.component.ts file as below

```
import { Component, OnInit } from '@angular/core';
import { EmployeeService } from '../../services/employee.service';
import { ActivatedRoute, Router } from '@angular/router';
import { Employee } from '../../common/employee';

@Component({
  selector: 'app-update-employee',
  templateUrl: './update-employee.component.html',
  styleUrl: './update-employee.component.css',
})
export class UpdateEmployeeComponent implements OnInit {
  id: number = 0;
  employee: Employee;
  constructor(
    private employeeService: EmployeeService,
    private route: ActivatedRoute,
    private router: Router
  ) {}
  ngOnInit(): void {
    this.id = this.route.snapshot.params['id'];
    this.employeeService.getEmployeeById(this.id).subscribe((data) => {
      this.employee = data;
      console.log(this.employee);
    });
  }
  onSubmit() {
    console.log(this.employee);
    this.employeeService
      .updateEmployee(this.id, this.employee)
      .subscribe((data) => {
        this.employee = data;
        this.router.navigate(['employees']);
      });
  }
}
```

To delete an Employee

Backend

employeeController file

```
package com.krishna.employee.controller;
```

```
import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.krishna.employee.entity.Employee;
import com.krishna.employee.exception.ResourceNotFoundException;
import com.krishna.employee.repository.EmployeeRepository;

@RestController
@CrossOrigin(origins = "http://localhost:4200")
public class EmployeeController {

	@Autowired
	private EmployeeRepository empRepo;
	@CrossOrigin(origins = "http://localhost:4200")
	@GetMapping("/employees")
	public List<Employee>getAllEmployess(){

		return empRepo.findAll();
	}

	@PostMapping("/employees")
	public Employee createEmployee(@RequestBody Employee employee) {

		return empRepo.save(employee);

	}

	@GetMapping("/employees/{id}")
	public ResponseEntity<Employee> getEmployeeById(@PathVariable int id) {

	Employee emp = empRepo.findById(id).orElseThrow(
				()-> new ResourceNotFoundException("Employee not
found with id"+id));

		return ResponseEntity.ok(emp);
	}

	@PutMapping("/employees/{id}")
```

```
	public ResponseEntity<Employee> updateEmployee( @PathVariable int id,
@RequestBody Employee employee) {
		Employee emp = empRepo.findById(id).orElseThrow(
				()-> new ResourceNotFoundException("Employee not
found with id"+id));

		emp.setFirstName(employee.getFirstName());
		emp.setLastName(employee.getLastName());
		emp.setSalary(employee.getSalary());
		empRepo.save(emp);
		return ResponseEntity.ok(emp);
	}

	@DeleteMapping("/employees/{id}")
	public void deleteById(@PathVariable int id) {
		Employee emp = empRepo.findById(id).orElseThrow(
				()-> new ResourceNotFoundException("Employee not
found with id"+id));
		empRepo.delete(emp);

	}

//	@DeleteMapping("/employees/{id}")
//	public ResponseEntity<Map<String,Boolean>> deleteById(@PathVariable int id)
{
//		Employee emp = empRepo.findById(id).orElseThrow(
//				()-> new ResourceNotFoundException("Employee not
found with id"+id));
//		empRepo.delete(emp);
//		Map<String,Boolean>response = new HashMap<String,Boolean>();
//		response.put("Deleted", Boolean.TRUE);
//		return ResponseEntity.ok(response);
//
//	}
}
```

Front end

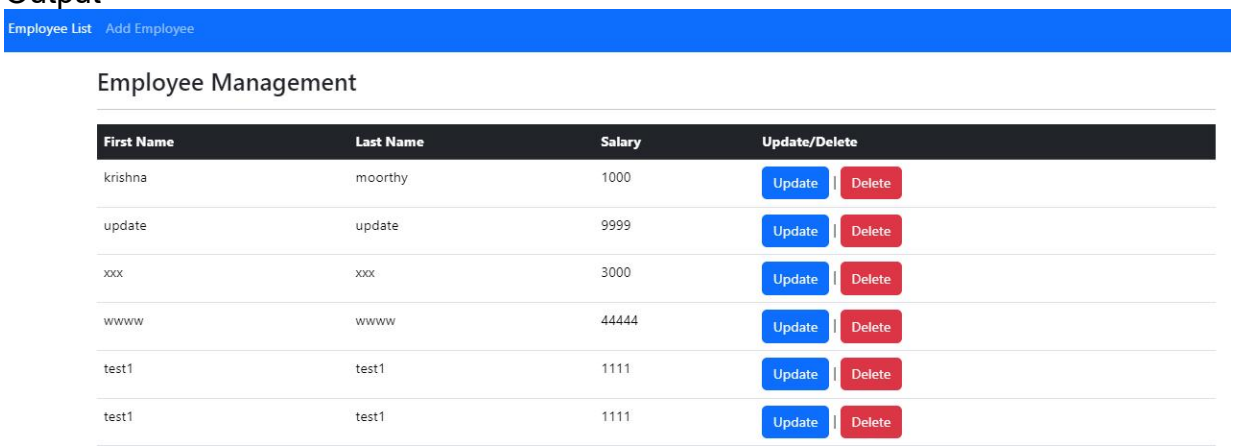
Update employee-listcomponent.html as below

```
<table class="table table-stripped">
  <thead class="table-dark">
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Salary</th>
      <th>Update/Delete</th>
    </tr>
  </thead>
  <tbody>
```

```
    <tr *ngFor="let employee of employees">
      <td>{{ employee.firstName }}</td>
      <td>{{ employee.lastName }}</td>
      <td>{{ employee.salary }}</td>
      <td>
        <button
          type="button"
          class="btn btn-primary"
          (click)="updateEmployee(employee.id)"
        >
          Update
        </button> |
        <button
          type="button"
          class="btn btn-danger"
          (click)="deleteEmployee(employee.id)"
        >
          Delete
        </button>
      </td>
    </tr>
  </tbody>
</table>
```

Define the method deleteEmployee() in employee-list.componet.ts and leave it blank for the time being

Output



Update employee.service.ts file.

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { Employee } from '../common/employee';

@Injectable({
  providedIn: 'root'
```

```
})
export class EmployeeService {
  private baseUrl = 'http://localhost:8080/employees';
  constructor(private httpClient:HttpClient) { }
  getEmployeeList():Observable<Employee[]>{
   // return this.httpClient.get<Employee[]>(`${this.baseUrl}`);
    return this.httpClient.get<Employee[]>(this.baseUrl);
  }
  addEmployee(employee:Employee):Observable<Object>{
  //  return this.httpClient.post(`${this.baseUrl}`,employee);
    return this.httpClient.post(this.baseUrl,employee);
  }
  getEmployeeById(id:number):Observable<Employee>{
    return this.httpClient.get<Employee>(`${this.baseUrl}/${id}`);
  }
  updateEmployee(id:number, employee:Employee):Observable<Employee>{
    return this.httpClient.put<Employee>(`${this.baseUrl}/${id}`,employee);
  }
  deleteEmployee(id:number):Observable<object>{
    return this.httpClient.delete(`${this.baseUrl}/${id}`);
  }
}
```

Update list-employee.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { Employee } from '../../common/employee';
import { EmployeeService } from '../../services/employee.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-employee-list',
  templateUrl: './employee-list.component.html',
  styleUrl: './employee-list.component.css'
})
export class EmployeeListComponent implements OnInit {
  employees:Employee[]=[];
  constructor(private employeeService:EmployeeService, private router:Router){}
  ngOnInit(): void {
    this.getEmployeeList();
  }
  private getEmployeeList(){
    this.employeeService.getEmployeeList().subscribe(data=>{
      this.employees=data;
    });
  }
  updateEmployee(id:number){
    this.router.navigate(['update-employee',id]);
  }
```

```
  deleteEmployee(id:number){
    this.employeeService.deleteEmployee(id).subscribe(data=>{
      console.log(data);
      this.getEmployeeList();
    });
  }
}
```

View employee

Generate a new component view-employee inside components

Update the app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { EmployeeListComponent } from './components/employee-list/employee-
list.component';
import { AddEmployeeComponent } from './components/add-employee/add-
employee.component';
import { UpdateEmployeeComponent } from './components/update-employee/update-
employee.component';
import { ViewEmployeeComponent } from './components/view-employee/view-
employee.component';

const routes: Routes = [
  {path:'employees', component:EmployeeListComponent},
  {path:'add-employee', component:AddEmployeeComponent},
  {path:'update-employee/:id', component:UpdateEmployeeComponent},
  {path:'view-employee/:id', component:ViewEmployeeComponent},
  {path:'', redirectTo:'employees',pathMatch:'full'},

];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Update list-employee.component.html file

```
<table class="table table-stripped">
  <thead class="table-dark">
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Salary</th>
      <th>Update/Delete</th>
```

```
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let employee of employees">
      <td>{{ employee.firstName }}</td>
      <td>{{ employee.lastName }}</td>
      <td>{{ employee.salary }}</td>
      <td>
        <button
          type="button"
          class="btn btn-primary"
          (click)="updateEmployee(employee.id)"
        >
          Update
        </button> |
        <button
          type="button"
          class="btn btn-danger"
          (click)="deleteEmployee(employee.id)"
        >
          Delete
        </button> |
        <button
        type="button"
        class="btn btn-success"
        (click)="viewEmployee(employee.id)"
      >
       View
      </button>
      </td>
    </tr>
  </tbody>
</table>
```

Update list-employee.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { Employee } from '../../common/employee';
import { EmployeeService } from '../../services/employee.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-employee-list',
  templateUrl: './employee-list.component.html',
  styleUrl: './employee-list.component.css'
})
export class EmployeeListComponent implements OnInit {
  employees:Employee[]=[];
  constructor(private employeeService:EmployeeService, private router:Router){}
```

```
  ngOnInit(): void {
    this.getEmployeeList();
  }
  private getEmployeeList(){
    this.employeeService.getEmployeeList().subscribe(data=>{
      this.employees=data;
    });
  }
  updateEmployee(id:number){
    this.router.navigate(['update-employee',id]);
  }
  deleteEmployee(id:number){
    this.employeeService.deleteEmployee(id).subscribe(data=>{
      console.log(data);
      this.getEmployeeList();
    });

  }
  viewEmployee(id:number){
    this.router.navigate(['view-employee',id]);
  }
}
```

View-employee.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { EmployeeService } from '../../services/employee.service';
import { ActivatedRoute, Router } from '@angular/router';
import { Employee } from '../../common/employee';

@Component({
  selector: 'app-view-employee',
  templateUrl: './view-employee.component.html',
  styleUrl: './view-employee.component.css',
})
export class ViewEmployeeComponent implements OnInit {
  id: number = 0;
  employee: Employee;
  constructor(
    private employeeService: EmployeeService,
    private route: ActivatedRoute,
    private router: Router
  ) {}
  ngOnInit(): void {
    this.id = this.route.snapshot.params['id'];
    this.employeeService.getEmployeeById(this.id).subscribe((data) => {
      this.employee = data;
    });
  }
```

```
  goToList() {
    this.router.navigate(['employees']);
  }
}
```

View-employee.component.html file

```
<div class="container">
    <div class="row">
      <div class="col-md-8 offset-md-3">
        <h3>Employee Details</h3>
        <label>First Name  </label>
        <p>{{employee.firstName}}</p>
        <label>Last Name</label>
        <p>{{employee.lastName}}</p>
        <label>Salary</label>
        <p>{{employee.salary}}</p>
      </div>
    </div>
        <button type="button" class="btn btn-primary" (click)="goToList()">Go to
List</button>
  </div>
```

Angular material project

Backend
Create a new database in mysql (no need to create the table)
Copy the previous example project and change application.properties inorder to change it to the new database
Change the entity employee.java by adding new fields and reconstruct getters, setters and constructor
Run the app and make sure table is created
Add some values to the table manually
Run the app and verify we can get the employee details through browser or postman

Front end

Create angular project
Install angular material (ng add @angular/material)

Required output

After installing the angular material to the project,
App.component.html file

```
<mat-toolbar color="primary">
  <span>Employee Management</span>
  <span class="example-spacer"></span>
  <button mat-raised-button>Add Employee</button>
</mat-toolbar>
```

App.component.css file

```
.example-spacer {
    flex: 1 1 auto;
  }
```

Required modules should be imported in app.module.ts file
App.module.ts file

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { provideAnimationsAsync } from '@angular/platform-browser/animations/async';
import {MatIconModule} from '@angular/material/icon';
import {MatButtonModule} from '@angular/material/button';
import {MatToolbarModule} from '@angular/material/toolbar';
@NgModule({
  declarations: [
    AppComponent,
      ],
  imports: [
    BrowserModule,
    AppRoutingModule,
       MatToolbarModule, MatButtonModule, MatIconModule
         ],
  providers: [
    provideAnimationsAsync()
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

All these codes are available in
https://material.angular.io/components/toolbar/examples
Ref - basic toolbar

Next on click of add employee, a dialog should be opened showing add-employee component
Ref - https://material.angular.io/components/dialog/overview

Create a new component add-employee inside components folder (ng g c components/add-employee)

To work with dialog we need to import MatDialogModule to app.module.ts file

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { provideAnimationsAsync } from '@angular/platform-browser/animations/async';
import {MatIconModule} from '@angular/material/icon';
import {MatButtonModule} from '@angular/material/button';
import {MatToolbarModule} from '@angular/material/toolbar';
import { MatDialogModule } from '@angular/material/dialog';
import { AddEmployeeComponent } from './components/add-employee/add-employee.component';
```

```
@NgModule({
  declarations: [
    AppComponent,
    AddEmployeeComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    MatToolbarModule, MatButtonModule, MatIconModule, MatDialogModule
  ],
  providers: [
    provideAnimationsAsync()
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

We need to write code in app.component.ts file (since add employee button is in app component)

```
import { Component } from '@angular/core';
import { MatDialog } from '@angular/material/dialog';
```

```
import { AddEmployeeComponent } from './components/add-employee/add-
employee.component';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'employee-management';
  constructor( private dialog:MatDialog){}
          // below code will open the dialog and show add-employee comp
  openAddEmpFormDialog(){
    this.dialog.open(AddEmployeeComponent);
  }
}
```

The above function should be executed on click of add employee button, so update the app.component.html file as below

app.component.html

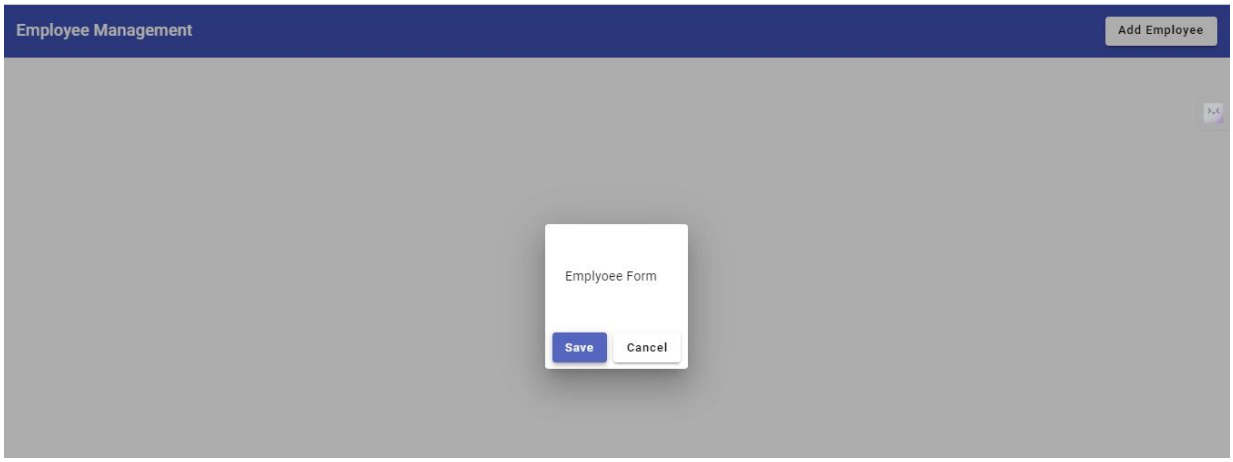
```
<mat-toolbar color="primary">
  <span>Employee Management</span>
  <span class="example-spacer"></span>
  <button mat-raised-button (click)="openAddEmpFormDialog()">Add
Employee</button>
</mat-toolbar>
```

Update add-emplyee.component.html file

Material dialog has 3 divs - 1 --> title
2 --> content
3 --> actions

```
<div mat-dialog-title>
<h3>Emplyoee Form </h3>
</div>
<div mat-dialog-content>

</div>
<div mat-dialog-actions>
<button mat-raised-button color="primary">Save</button>
<button mat-raised-button>Cancel</button>
</div>
```

Output



Now we need to add the input fields (I.e controls)
We use from fields from angular material
https://material.angular.io/components/form-field/overview

Import MatFormFieldModule to app.module.ts file (imports section and imports array)

Import MatInputModule to app.module.ts file (imports section and imports array)
this is required for the input field in the form

Add-employee.component.html file

```
<div mat-dialog-title>
  <h3>Emplyoee Form</h3>
</div>
<div mat-dialog-content class="content">
  <div class="row">
    <mat-form-field appearance="outline">
      <mat-label>First Name</mat-label>
      <input matInput type="text" />
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>Last Name</mat-label>
      <input matInput type="text" />
    </mat-form-field>
  </div>
  <div class="row">
    <mat-form-field appearance="outline">
      <mat-label>Email</mat-label>
      <input matInput type="email" />
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>Salary</mat-label>
      <input matInput />
    </mat-form-field>
  </div>
```

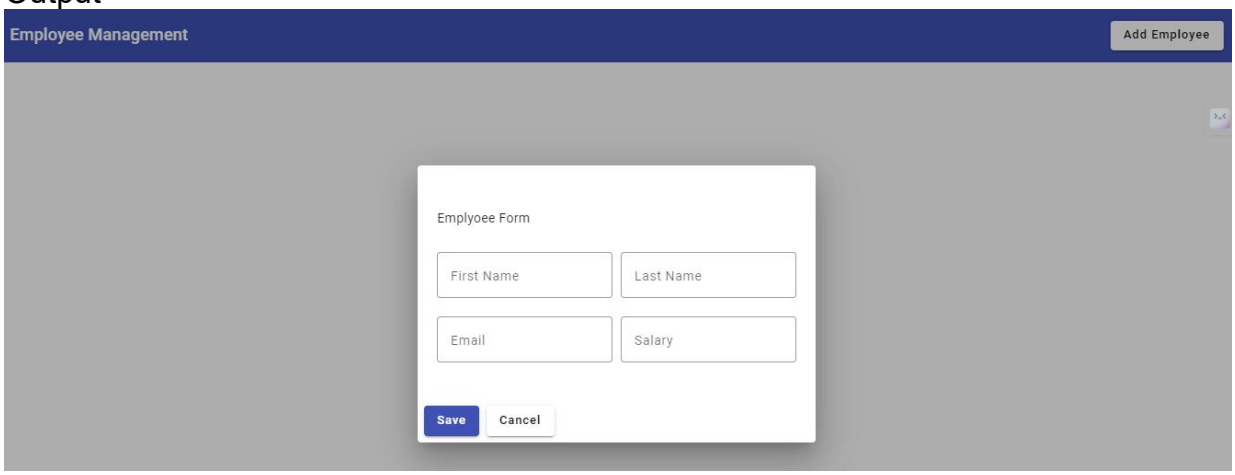
```
</div>
<div mat-dialog-actions>
  <button mat-raised-button color="primary">Save</button>
  <button mat-raised-button>Cancel</button>
</div>
```

Add-eployee.component.css file

```
 .content{
   padding-top: 20px;
 }
 .row{
   display: flex;
   gap: 10px;
 }
```

The above css is not form material, user defined inorder to correct minor spacing issue

Output



Next we will add date picker of material for date of birth field

Import MatDatepickerModule and MatNativeDateModule to app.module.ts file
(imports section and imports array)

Add-employee.component.html file

```
<div mat-dialog-title>
  <h3>Emplyoee Form</h3>
</div>
<div mat-dialog-content class="content">
  <div class="row">
    <mat-form-field appearance="outline">
      <mat-label>First Name</mat-label>
      <input matInput type="text" />
    </mat-form-field>
    <mat-form-field appearance="outline">
```

```
    <mat-label>Last Name</mat-label>
    <input matInput type="text" />
   </mat-form-field>
  </div>
  <div class="row">
   <mat-form-field appearance="outline">
    <mat-label>Email</mat-label>
    <input matInput type="email" />
   </mat-form-field>
   <mat-form-field appearance="outline">
    <mat-label>Salary</mat-label>
    <input matInput />
   </mat-form-field>
  </div>
  <div class="row">
   <mat-form-field appearance="outline">
      <mat-label>Choose DOB</mat-label>
      <input matInput [matDatepicker]="picker">
      <mat-hint>MM/DD/YYYY</mat-hint>
      <mat-datepicker-toggle matIconSuffix [for]="picker"></mat-datepicker-toggle>
      <mat-datepicker #picker></mat-datepicker>
    </mat-form-field>
  </div>
</div>
<div mat-dialog-actions>
  <button mat-raised-button color="primary">Save</button>
  <button mat-raised-button>Cancel</button>
</div>
```

Add-employee.component.css file

```
 .content{
   padding-top: 20px;
 }
 .row{
   display: flex;
   gap: 10px;
   mat-form-field{
      width: 100%;
   }
 }
```

Next we add gender, with radio button
Import MatRadioModule to app.module.ts file (imports section and imports array)

Add-employee.component.html file

```
<div mat-dialog-title>
  <h3>Emplyoee Form</h3>
```

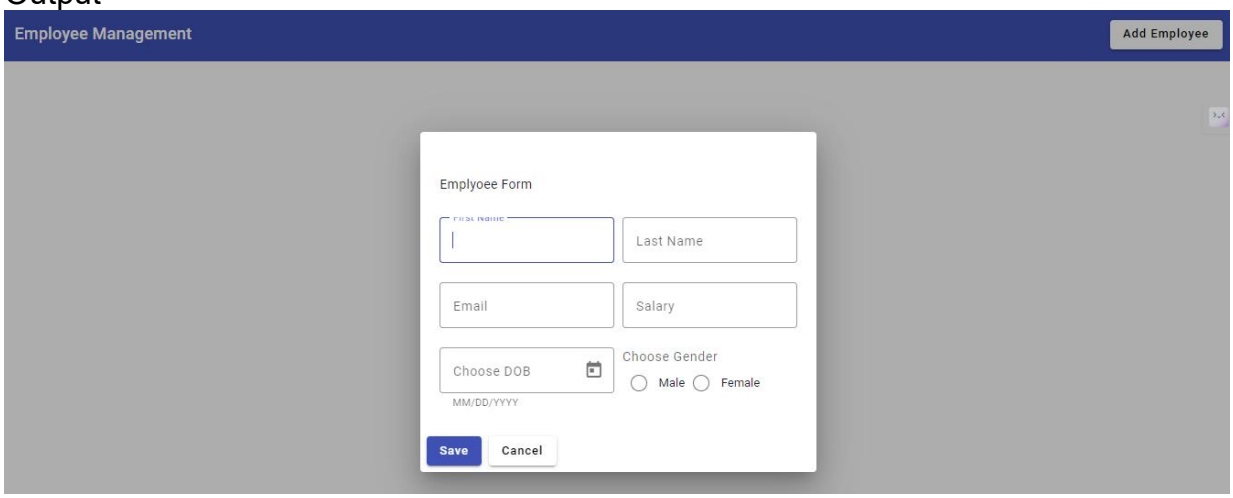
```
</div>
<div mat-dialog-content class="content">
  <div class="row">
    <mat-form-field appearance="outline">
      <mat-label>First Name</mat-label>
      <input matInput type="text" />
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>Last Name</mat-label>
      <input matInput type="text" />
    </mat-form-field>
  </div>
  <div class="row">
    <mat-form-field appearance="outline">
      <mat-label>Email</mat-label>
      <input matInput type="email" />
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>Salary</mat-label>
      <input matInput />
    </mat-form-field>
  </div>
  <div class="row">
    <mat-form-field appearance="outline">
        <mat-label>Choose DOB</mat-label>
        <input matInput [matDatepicker]="picker">
        <mat-hint>MM/DD/YYYY</mat-hint>
        <mat-datepicker-toggle matIconSuffix [for]="picker"></mat-datepicker-
toggle>
        <mat-datepicker #picker></mat-datepicker>
      </mat-form-field>
      <mat-radio-group aria-label="Select an option">
        <mat-label>Choose Gender</mat-label><br>
        <mat-radio-button value="male">Male</mat-radio-button>
        <mat-radio-button value="female">Female</mat-radio-button>
      </mat-radio-group>
  </div>
</div>
<div mat-dialog-actions>
  <button mat-raised-button color="primary">Save</button>
  <button mat-raised-button>Cancel</button>
</div>
```

Add-employee.component.css file

```
  .content{
    padding-top: 20px;
  }
  .row{
```

```
    display: flex;
    gap: 10px;
    mat-form-field{
        width: 100%;
    }
    mat-radio-group{
        width: 100%;
    }
  }
```

Output



we will add education using select
Import MatSelectModule
We will add company name (simillar to first name)
Add-employee.component.html

```
<div mat-dialog-title>
  <h3>Emplyoee Form</h3>
</div>
<div mat-dialog-content class="content">
  <div class="row">
    <mat-form-field appearance="outline">
      <mat-label>First Name</mat-label>
      <input matInput type="text" />
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>Last Name</mat-label>
      <input matInput type="text" />
    </mat-form-field>
  </div>
  <div class="row">
    <mat-form-field appearance="outline">
      <mat-label>Email</mat-label>
      <input matInput type="email" />
    </mat-form-field>
    <mat-form-field appearance="outline">
```

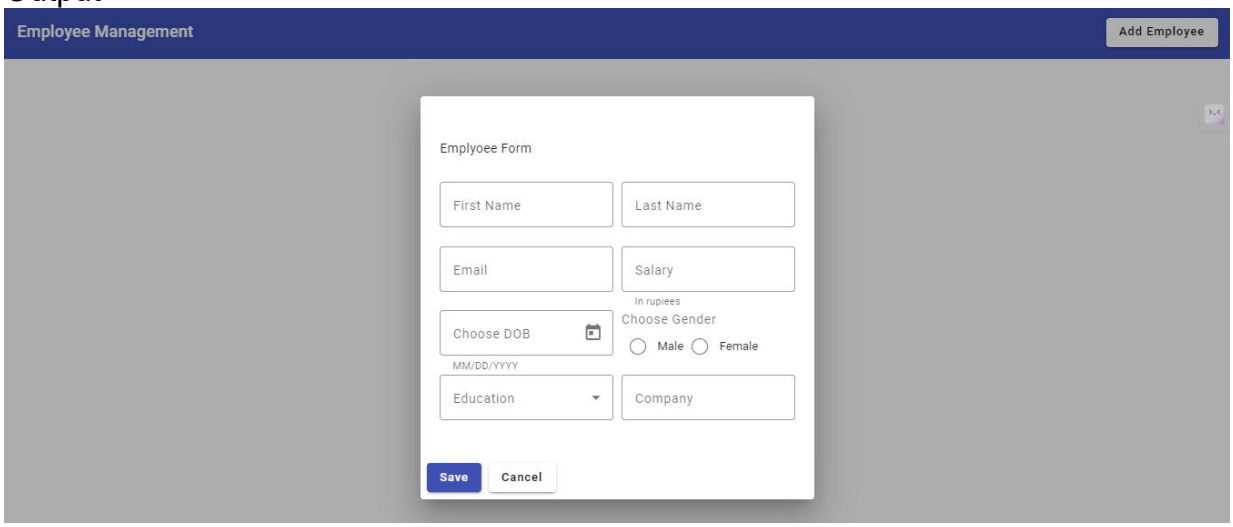
```
      <mat-label>Salary</mat-label>
      <input matInput />
      <mat-hint>In rupiees</mat-hint>
    </mat-form-field>
  </div>
  <div class="row">
    <mat-form-field appearance="outline">
        <mat-label>Choose DOB</mat-label>
        <input matInput [matDatepicker]="picker">
        <mat-hint>MM/DD/YYYY</mat-hint>
        <mat-datepicker-toggle matIconSuffix [for]="picker"></mat-datepicker-
toggle>
        <mat-datepicker #picker></mat-datepicker>
      </mat-form-field>
      <mat-radio-group aria-label="Select an option">
        <mat-label>Choose Gender</mat-label><br>
        <mat-radio-button value="male">Male</mat-radio-button>
        <mat-radio-button value="female">Female</mat-radio-button>
      </mat-radio-group>
  </div>
  <div class="row">
    <mat-form-field appearance="outline">
        <mat-label>Education</mat-label>
        <mat-select>
            <mat-option *ngFor="let val of education"
[value]="val">{{val}}</mat-option>
        </mat-select>
      </mat-form-field>
      <mat-form-field appearance="outline">
        <mat-label>Company</mat-label>
        <input matInput type="text" />
      </mat-form-field>
  </div>
</div>
<div mat-dialog-actions>
  <button mat-raised-button color="primary">Save</button>
  <button mat-raised-button>Cancel</button>
</div>
```

Add-employee.component.ts

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

@Component({
  selector: 'app-add-employee',
  templateUrl: './add-employee.component.html',
  styleUrl: './add-employee.component.css'
})
export class AddEmployeeComponent {
```

```
education:string[]=[
  'Metric',
  'Degree',
  'Graduate',
  'Post Graduate'
];
}
```

Output



To make button appearance better add a class row to actions div and define the style in add-employee.component.css file

add-employee.component.html

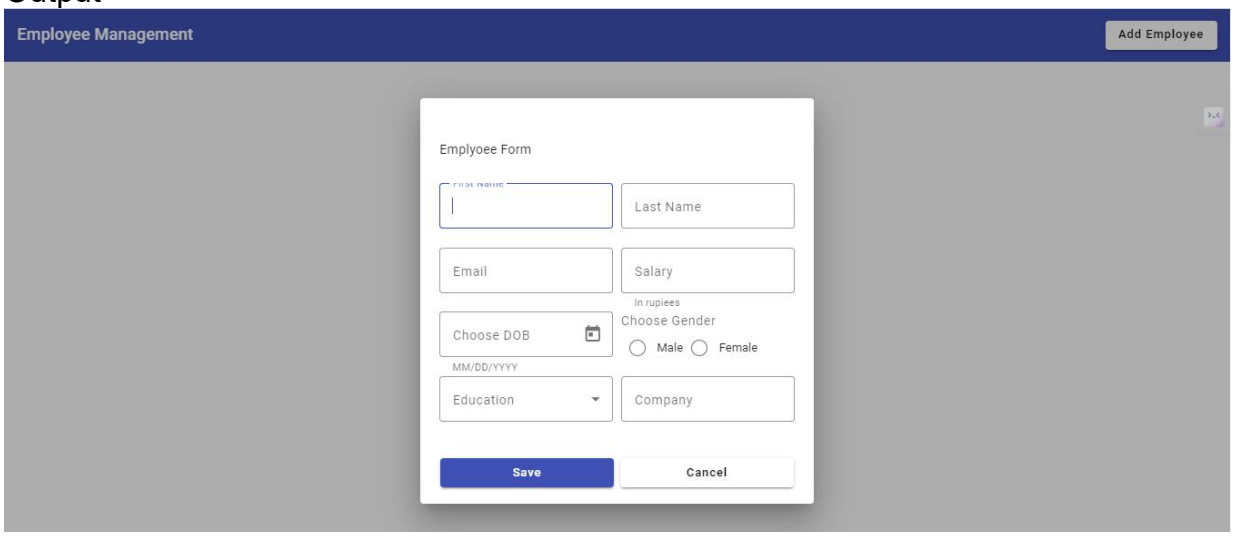
```
<div mat-dialog-actions class="action">
  <button mat-raised-button color="primary">Save</button>
  <button mat-raised-button>Cancel</button>
</div>
```

add-employee.component.css

```
  .content{
    padding-top: 20px;
  }
  .row{
    display: flex;
    gap: 10px;
    mat-form-field{
        width: 100%;
    }
    mat-radio-group{
        width: 100%;
    }
  }
```

```
.action{
  padding: 0px 25px 20px;
  button{
    flex:1;
  }
}
```

Output



We need to convert the add-eployee to a form. We use reactive form approach.
We need to import ReactiveFormsModule to app.module.ts file
Next we can use earlier approach (refer **79. Introduction to Reactive Forms Approach. Create FormGroup and FormControl with code in Angular.)**

Or else we can use a bit different method of reactive forms in ts file of add-employee.component (no changes in html file during new approach or old approach)

Add-employee.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormControl, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-add-employee',
  templateUrl: './add-employee.component.html',
  styleUrl: './add-employee.component.css'
})
export class AddEmployeeComponent implements OnInit {
education:string[]=[
  'Metric',
  'Degree',
  'Graduate',
  'Post Graduate'
];
signupForm: FormGroup;
// new approach
constructor( private fb:FormBuilder){
  this.signupForm = this.fb.group({
    'firstName':'',
    'lastName':'',
```

```
    'email':'',
    'salary':'',
    'dob':'',
    'gender':'',
    'education':'',
    'company':''
  })
}
//old approach
 ngOnInit(): void {}
//   this.signupForm = new FormGroup({
//     'firstName': new FormControl(''),
//     'lastName': new FormControl(''),
//     'email': new FormControl(''),
//     'salary': new FormControl(''),
//     'dob': new FormControl(''),
//     'gender': new FormControl(''),
//     'education': new FormControl(''),
//     'company': new FormControl(''),
//   });
// }
onSubmit(){
  console.log(this.signupForm)
}
}
```

Add-employee.component.html file

```
<div mat-dialog-title>
  <h3>Emplyoee Form</h3>
</div>
<form [formGroup]="signupForm" (ngSubmit)="onSubmit()">
<div mat-dialog-content class="content">
  <div class="row">
    <mat-form-field appearance="outline">
      <mat-label>First Name</mat-label>
      <input matInput type="text"  formControlName="firstName"/>
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>Last Name</mat-label>
      <input matInput type="text" formControlName="lastName"/>
    </mat-form-field>
  </div>
  <div class="row">
    <mat-form-field appearance="outline">
      <mat-label>Email</mat-label>
      <input matInput type="email" formControlName="email"/>
```

```
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>Salary</mat-label>
      <input matInput formControlName="salary"/>
      <mat-hint>In rupiees</mat-hint>
    </mat-form-field>
  </div>
  <div class="row">
    <mat-form-field appearance="outline">
        <mat-label>Choose DOB</mat-label>
        <input matInput [matDatepicker]="picker" formControlName="dob">
        <mat-hint>MM/DD/YYYY</mat-hint>
        <mat-datepicker-toggle matIconSuffix [for]="picker"></mat-datepicker-
toggle>
        <mat-datepicker #picker></mat-datepicker>
      </mat-form-field>
      <mat-radio-group aria-label="Select an option" formControlName="gender">
        <mat-label>Choose Gender</mat-label><br>
        <mat-radio-button value="male">Male</mat-radio-button>
        <mat-radio-button value="female">Female</mat-radio-button>
      </mat-radio-group>
  </div>
  <div class="row">
    <mat-form-field appearance="outline">
        <mat-label>Education</mat-label>
        <mat-select formControlName="education">
            <mat-option *ngFor="let val of education"
[value]="val">{{val}}</mat-option>
        </mat-select>
      </mat-form-field>
      <mat-form-field appearance="outline">
        <mat-label>Company</mat-label>
        <input matInput type="text" formControlName="company"/>
      </mat-form-field>
  </div>
</div>
<div mat-dialog-actions class="action">
  <button mat-raised-button color="primary">Save</button>
  <button mat-raised-button>Cancel</button>
</div>
</form>
```

Create a class employee inside common folder (ng g class common/employee)
Define the fields which are required as per database
Employee.ts file

```
export class Employee {
    id:number = 0;
    firstName:string='';
    lastName:string = '';
```

```
    email:string = '';
    salary:string = '';
    dob:Date;
    gender:string = '';
    education:string = '';
    company:string = '';
}
```

Crete a service employee service inside services (ng g service services/employee)

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { Employee } from '../common/employee';

@Injectable({
  providedIn: 'root'
})
export class EmployeeService {
  private baseUrl = 'http://localhost:8080/employees';
  constructor(private httpClient:HttpClient) { }
  addEmployee(employee:Employee):Observable<Object>{
    //  return this.httpClient.post(`${this.baseUrl}`,employee);
      return this.httpClient.post(this.baseUrl,employee);
    }
}
```

Use this addEmployee method to save the employee from modal

Update add-employee.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormControl, FormGroup } from '@angular/forms';
import { Employee } from '../../common/employee';
import { EmployeeService } from '../../services/employee.service';
import { DialogRef } from '@angular/cdk/dialog';

@Component({
  selector: 'app-add-employee',
  templateUrl: './add-employee.component.html',
  styleUrl: './add-employee.component.css',
})
export class AddEmployeeComponent implements OnInit {
  employee: Employee = new Employee();
  education: string[] = ['Metric', 'Degree', 'Graduate', 'Post Graduate'];
  signupForm: FormGroup;
  // new approach
  constructor(
    private fb: FormBuilder,
```

```
    private dialog:DialogRef<AddEmployeeComponent>,
    private employeeService: EmployeeService,
  ) {
    this.signupForm = this.fb.group({
      firstName: '',
      lastName: '',
      email: '',
      salary: '',
      dob: '',
      gender: '',
      education: '',
      company: '',
    });
  }
  //old approach
  ngOnInit(): void {}
  //   this.signupForm = new FormGroup({
  //     'firstName': new FormControl(''),
  //     'lastName': new FormControl(''),
  //     'email': new FormControl(''),
  //     'salary': new FormControl(''),
  //     'dob': new FormControl(''),
  //     'gender': new FormControl(''),
  //     'education': new FormControl(''),
  //     'company': new FormControl(''),
  //   });
  // }
  onSubmit() {
    this.employee = this.signupForm.value;
    this.addEmployee();
  }
  addEmployee() {
    this.employeeService.addEmployee(this.employee).subscribe((data) => {
      console.log(data);
      alert("Employee added Successfully");
      this.dialog.close();
    });
  }
}
```

Displaying the list of employees using material table

Use material table with pagination sorting and search
App.module.ts file

```
import {MatTableModule} from '@angular/material/table';
import { MatPaginatorModule } from '@angular/material/paginator';
import { MatSortModule } from '@angular/material/sort';
```

Also to imports array of this file

App.component.ts file

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { MatDialog } from '@angular/material/dialog';
import { AddEmployeeComponent } from './components/add-employee/add-
employee.component';
import { EmployeeService } from './services/employee.service';
import { Employee } from './common/employee';

import { MatPaginator } from '@angular/material/paginator';
import { MatSort } from '@angular/material/sort';
import { MatTableDataSource } from '@angular/material/table';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css',
})
export class AppComponent implements OnInit {
  title = 'employee-management';
      // code form angular material
  displayedColumns: string[] = [
    'id',
    'firstName',
    'lastName',
    'email',
    'salary',
    'dob',
    'gender',
    'education',
    'company',
  ];
  dataSource: MatTableDataSource<any>;
  @ViewChild(MatPaginator) paginator: MatPaginator;
  @ViewChild(MatSort) sort: MatSort;
  constructor(
    private dialog: MatDialog,
    private employeeService: EmployeeService
  ) {}
  openAddEmpFormDialog() {
    this.dialog.open(AddEmployeeComponent);
  }
  ngOnInit(): void {
    this.employeeService.getEmployeeList().subscribe((data) => {
      this.dataSource = new MatTableDataSource(data);
      this.dataSource.sort = this.sort;
      this.dataSource.paginator = this.paginator;
    });
  }
```

```
    // code from angular material
   applyFilter(event: Event) {
    const filterValue = (event.target as HTMLInputElement).value;
    this.dataSource.filter = filterValue.trim().toLowerCase();
    if (this.dataSource.paginator) {
      this.dataSource.paginator.firstPage();
    }
  }
}
```

App.component.html file

```
<mat-toolbar color="primary">
  <span>Employee Management</span>
  <span class="example-spacer"></span>
  <button mat-raised-button (click)="openAddEmpFormDialog()">Add
Employee</button>
</mat-toolbar>
// code is from angular material and customized for the project
<mat-form-field>
  <mat-label>Filter</mat-label>
  <input matInput (keyup)="applyFilter($event)" placeholder="Ex. Mia" #input>
</mat-form-field>
<div class="mat-elevation-z8">
  <table mat-table [dataSource]="dataSource" matSort>
    <!-- ID Column -->
    <ng-container matColumnDef="id">
      <th mat-header-cell *matHeaderCellDef mat-sort-header> ID </th>
      <td mat-cell *matCellDef="let row"> {{row.id}} </td>
    </ng-container>
    <!-- Progress Column -->
    <ng-container matColumnDef="firstName">
      <th mat-header-cell *matHeaderCellDef mat-sort-header> First Name </th>
      <td mat-cell *matCellDef="let row"> {{row.firstName}} </td>
    </ng-container>
    <!-- Name Column -->
    <ng-container matColumnDef="lastName">
      <th mat-header-cell *matHeaderCellDef mat-sort-header> Last Name </th>
      <td mat-cell *matCellDef="let row"> {{row.lastName}} </td>
    </ng-container>
    <!-- Fruit Column -->
    <ng-container matColumnDef="email">
      <th mat-header-cell *matHeaderCellDef mat-sort-header> Email </th>
      <td mat-cell *matCellDef="let row"> {{row.email}} </td>
    </ng-container>
```

```
    <ng-container matColumnDef="salary">
      <th mat-header-cell *matHeaderCellDef mat-sort-header> Salary </th>
      <td mat-cell *matCellDef="let row"> {{row.salary | currency:'INR'}} </td>
    </ng-container>
    <ng-container matColumnDef="dob">
      <th mat-header-cell *matHeaderCellDef mat-sort-header> DOB </th>
      <td mat-cell *matCellDef="let row"> {{row.dob | date:'dd-MM-YYYY'}} </td>
    </ng-container>
    <ng-container matColumnDef="gender">
      <th mat-header-cell *matHeaderCellDef mat-sort-header> Gender </th>
      <td mat-cell *matCellDef="let row"> {{row.gender}} </td>
    </ng-container>
    <ng-container matColumnDef="education">
      <th mat-header-cell *matHeaderCellDef mat-sort-header> Education </th>
      <td mat-cell *matCellDef="let row"> {{row.education}} </td>
    </ng-container>
    <ng-container matColumnDef="company">
      <th mat-header-cell *matHeaderCellDef mat-sort-header> Company </th>
      <td mat-cell *matCellDef="let row"> {{row.company}} </td>
    </ng-container>
    <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
    <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
    <!-- Row shown when there is no matching data. -->
    <tr class="mat-row" *matNoDataRow>
      <td class="mat-cell" colspan="4">No data matching the filter
"{{input.value}}"</td>
    </tr>
  </table>
  <mat-paginator [pageSizeOptions]="[2, 3, 5, 10]" aria-label="Select page of
users"></mat-paginator>
</div>
```

App.component.css file