AngularJS is a Javascript open-source front-end structural framework that is mainly used to develop single-page web applications(SPAs). It is a continuously growing and expanding framework which provides better ways for developing web applications. It changes the static HTML to dynamic HTML. Its features like dynamic binding and dependency injection eliminate the need for code that we have to write otherwise. AngularJS is rapidly growing and because of this reason, we have different versions of AngularJS with the latest stable being 1.7.9. It is also important to note that Angular is different from AngularJS. It is an open-source project which can be freely used and changed by anyone. It extends HTML attributes with Directives, and data is bound with HTML.

Why use AngularJS?
- **Easy to work with:** All you need to know to work with AngularJS is the basics of HTML, CSS, and Javascript, not necessary to be an expert in these technologies.
- **Time-saving:** AngularJs allows us to work with components and hence we can use them again which saves time and unnecessary code.
- **Ready to use a template:** AngularJs is mainly plain HTML, and it mainly makes use of the plain HTML template and passes it to the DOM and then the AngularJS compiler. It traverses the templates and then they are ready to use.
- **Directives:** AngularJS's directives allow you to extend HTML with custom elements and attributes. This enables you to create reusable components and define custom behaviors for your application. Directives make it easier to manipulate the DOM, handle events, and encapsulate complex UI logic within a single component.

1.a Course Name: Angular JS Module
 Name: Angular Application Setup

Step 1 : Before install angular we need to install node js and vs studion
Step 2 : Open command prompt

Use this commands for install angular

➢ **npm install -g @angular/cli**

above command is used to install the angular

➢ **ng v**

through above command we can check the version of angular

Angular CLI: 16.1.4
Node: 18.15.0
Package Manager: npm 9.5.0
OS: win32 ia32

Creating a Angular Application

F:\Aditya_College_Informations\meanstacklab\Module-2\Angular>ng new myapp

Now let see how build the server

Step1 : place mcart folder in any Drive (C,D,E,F....)

Then use below commands
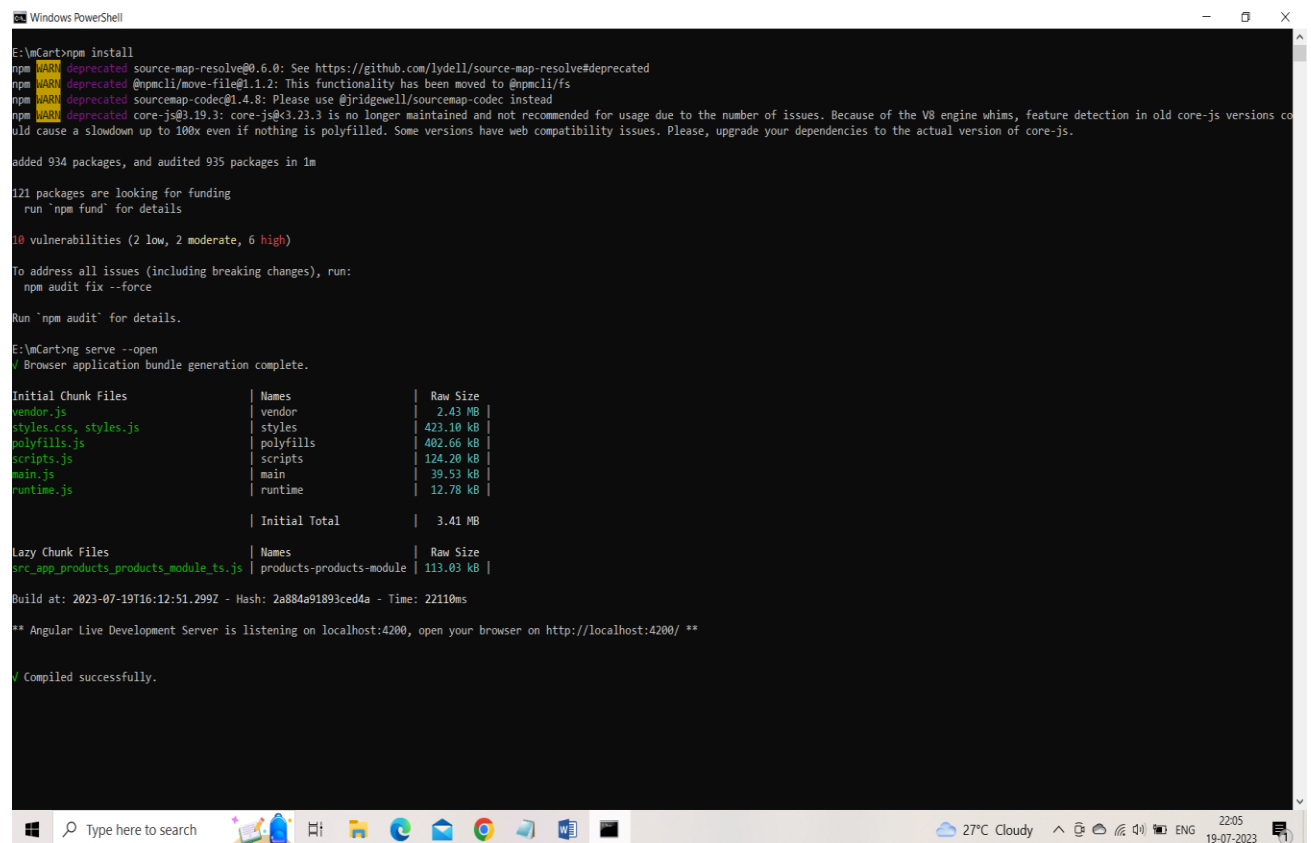
D:mcart > npm install

This will create a folder called node_modules with all the dependencies installed inside it

After complete the installation check all node modules are installed or not

Then run the mcart using below command

D:mcart>ng serve --open

1B . Module Name : Component

Create new component called hello and render hello angular on the page

Angular app – One more modules

Module – One or more components and services

Components – Html + css

Services – Business logic

Module interact and ultimately render the view in the browser

Let's start the angular application is hello-world

Create angular folder

E:\Angular>ng new hello-world

//Above command for create angular application

E:\Angular>ng serve –open

//Execute angular application

Then open it visual studio go to src->app->app.component.ts

Find title property add another called name

Then go to scr->app->app.component.html

Find the `<span>{{ title }} app is running!</span>`

Then add another tag with name property

```
<div>
   <span>{{name}}</span>
 </div>
```

Component.ts

Component.html

What is Component?

For creating user define component we need to use this command

>ng g c test

After execute above we find below files for test component
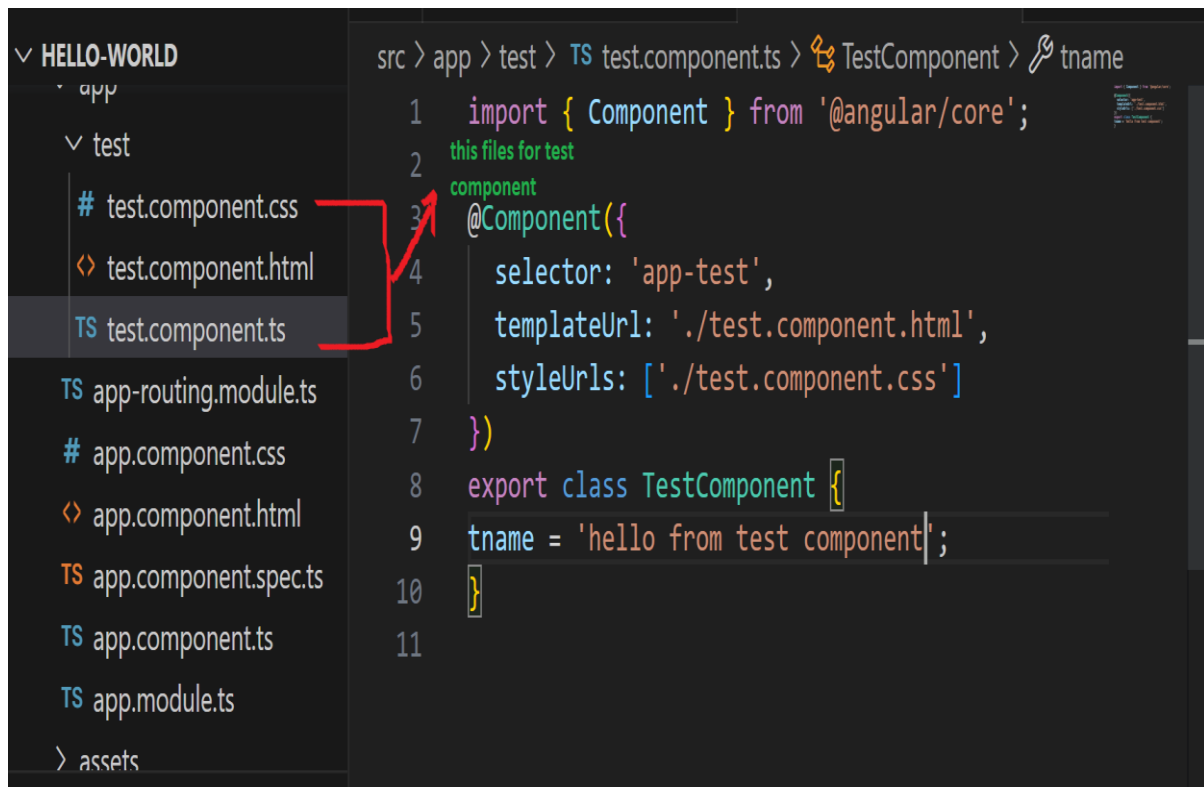
How to render hello from test component

1. declare tname variable in TestComponent class
2. now open test.component.html add below script

```
<p>test Component Works!</p>
<p>{{ tname  }}</p>
```

3. One check test.component.ts for selector . selector we can find in Decorators
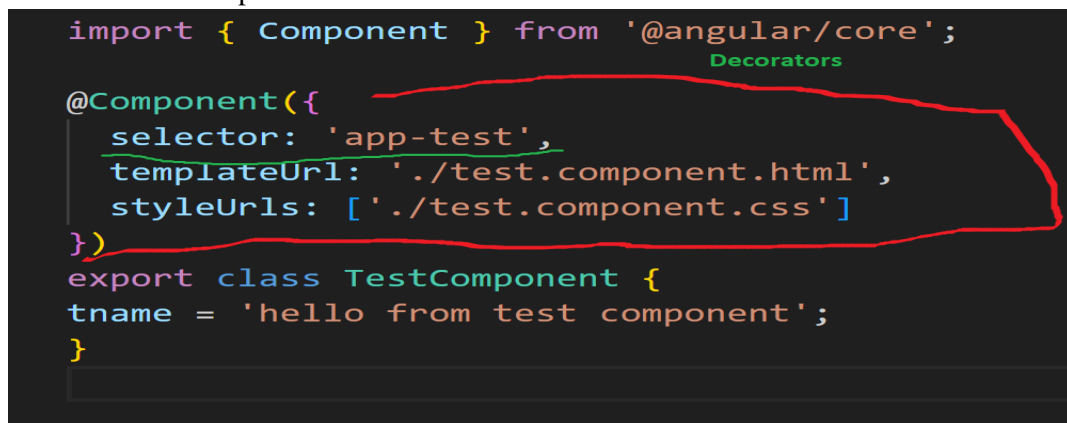
4. Now we have to add that selector in app.component.html

\<div\>

\<span\>{{ title }} app is running!\</span\>

\</div\>

\<br\>

\<div\>

\</div\>

\<app-test\>\</app-test\>

\<router-outlet\>\</router-outlet\>

5. Run the application using below command

E:\Angular\hello-world\>npm start
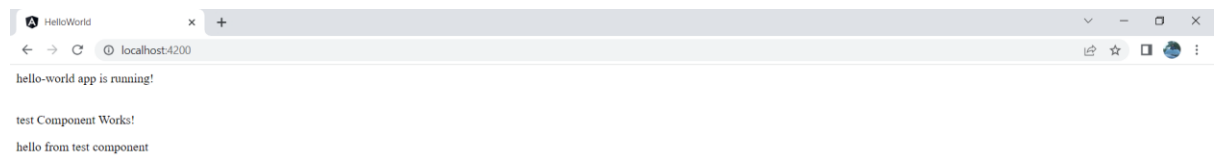
1 c . Add an event to the hello component template and when it is clicked, it should change the courseName.

What is event binding
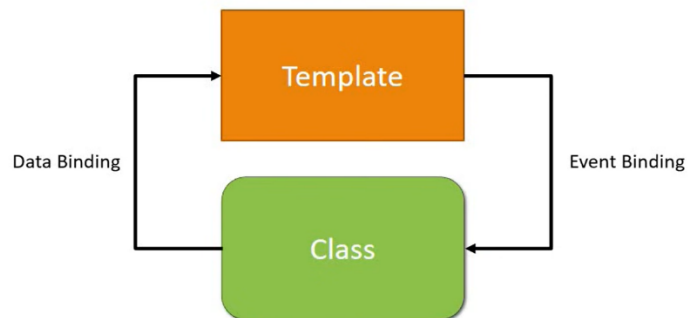
Step on go to test.component .html then create button like below

```
<p>test works!</p>
<button (click)="onclick()"> Click Me... </button>
<button (click)="message='this message from second button'"> Click Me..
</button>
<h2>{{message}} </h2>
```

Then go to test.componet.ts write the function onclick

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

@Component({
  selector: 'app-test',
  templateUrl: './test.component.html',
  styleUrls: ['./test.component.css']
})
export class TestComponent {
      //public name ="abc";
        public message = "";
      onclick()
      {
        this.message="this is my message..."
      }
}
```
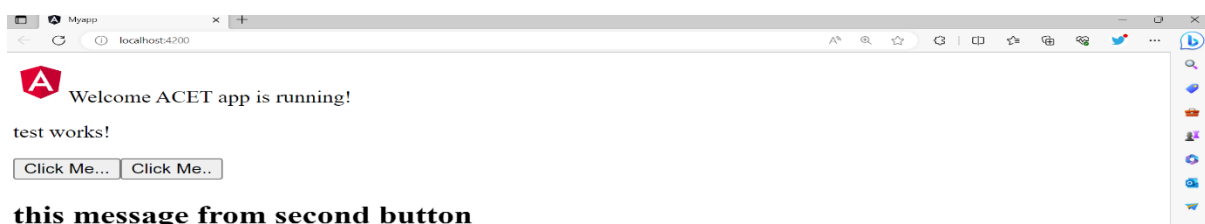
2 a . Course Name : Structural Directives

Structural directives use for add or remove html elements

➔ NgIf
➔ Ngswitch
➔ Ngfor

Create a login form with username and password fields. If the user enters the correct credentials, it should render a "Welcome <>" message otherwise it should render "Invalid Login!!! Please try again…" message

test.component.html

```html
<div *ngIf="!submitted">
    <form>
      <label>User Name</label>
      <input type="text" #username /><br /><br />
      <label for="password">Password</label>
      <input type="password" name="password" #password /><br />
    </form>
    <button (click)="onsubmit(username.value, password.value)">Login</button>
  </div>
  <div *ngIf="submitted">
    <div *ngIf="isAuthenticated; else failureMsg">
      <h4>Welcome {{ username }}</h4>
    </div>
    <ng-template #failureMsg>
      <h4>Invalid Login !!! Please try again...</h4>
    </ng-template>
    <button type="button" (click)="submitted = false">Back</button>
  </div>
```

test.component.ts

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

@Component({
  selector: 'app-test',
  templateUrl: './test.component.html',
  styleUrls: ['./test.component.css']
})
export class TestComponent {
      //public name ="abc";
      isAuthenticated! : boolean;
      submitted = false;
      username! : string;

      onsubmit(name: string ,password: string)
```

```
            {
                    this.submitted=true;
                    this.username=name;
                    if(name=='admin' && password=='admin')
                    {
                        this.isAuthenticated=true;
                    }
                    else
                    {
                        this.isAuthenticated=false;
                    }
            }
}
```

User Name|user1

Password|•••••

Login

Invalid Login !!! Please try again...
Back

ngFor:
ngFor directive is used to iterate over collection of data

2bCreate a courses array and rendering it in the template using ngFor directive in a list format

.html

```
<p>--- ngfor ---</p>
 <ul>
    <li *ngFor="let course of courses;  let i = index">
        {{i}} - {{ course}}
    </li>
  </ul>

  <ul>
    <li *ngFor="let subject of subjects">
       {{subject}}
    </li>
  </ul>
```

```
<h1> names ... </h1>
<ul>
  <li *ngFor="let name of names">
    {{name}}
  </li>
</ul>
```

Output :

.ts

```
export class DirComponent {
   //public directives = "ngif||ngswitch||ngfor";
   displaymessage = true;
 //ng for
 courses: any[]=["Type script","Java Script","Node Js"];
 subjects : any[]=["IOT","CC"];

}
```

--- ngfor ---

- 0 - Type script
- 1 - Java Script
- 2 - Node Js

- IOT
- CC

ngSwitch

ngSwitch adds or remove DOM tree when their expression match the switch expression

2c) Display the correct option based on the value passed to ngSwitch directive.

.ts file

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

@Component({
  selector: 'app-dir',
  templateUrl: './dir.component.html',
  styleUrls: ['./dir.component.css']
})
```

```
export class DirComponent {

 choice=0;

 nextchoice()
 {
   this.choice++;
 }
}
```

`

.html

```
  <h2 class="title">Switch Case..</h2>

<div [ngSwitch]="choice">
<p *ngSwitchCase="1">{{choice}} First Choice </p>
<p *ngSwitchCase="2">{{choice}}  Second Choice </p>
<p *ngSwitchCase="3">{{choice}} Third Choice </p>
<p *ngSwitchDefault>{{choice}} Default Choice </p>
</div>
<button (click)="nextchoice()"> Next Choice </button>
```

Out put

Switch Case..

1 First Choice

Next Choice

Switch Case..

2 Second Choice

Next Choice

2d) Create a custom structural directive called 'repeat' which should repeat the element given a number of times.

repeat.directive.ts

```
import { Directive, TemplateRef, ViewContainerRef,Input } from
'@angular/core';

@Directive({
  selector: '[appRepeat]'
})
export class RepeatDirective {
  constructor(private templateRef: TemplateRef<any>, private viewContainer:
ViewContainerRef) { }
  @Input() set appRepeat(count: number) {
    for (let i = 0; i < count; i++) {
      this.viewContainer.createEmbeddedView(this.templateRef);
    }
  }
}
```

app.component.html

```
<h2> repeat directive </h2>
 <p *appRepeat="5">hello</p>
```

Output:

Structural Directive

I am being repeated...
I am being repeated...
I am being repeated...
I am being repeated...
I am being repeated...

3a) Apply multipule css properties to a paragraph in a component using ngStyle

app.component.ts

```
export class AppComponent {
    title = "ACET";
    isactive = "Active";
    isBordered = true;   }
```

app.component.html

```
<p [ngStyle]="{color:isactive=='Active' ? 'green':'red'}"> Your Account is
{{isactive}}  </p>
```

Out put :

Your Account is Active

3b) Apply multiple css classes to the text using ngClass directive

app.component.html

```
<div [ngClass]="{bordered: isBordered}">
  Border {{ isBordered ? "ON" : "OFF" }}
</div>
```

app.componenet.ts

```
export class AppComponent {
    title = "ACET";
    isactive = "Active";
    isBordered = true;   }
```

3c) Module Name: Custom Attribute Directive

Module Name: Create an attribute directive called 'Show Message' which should display the given message in a paragraph when a user clicks on it and should change the text color to red.

Step1: create directive using below command

  ➢ ng generate directive 'ShowMessage'
                    (or)
  ➢ ng g d 'ShowMessage'

then we find two files with extension .ts and spec.ts

1.ShowMessage.directive.ts

2.ShowMessage.directive.spec.ts

Open ShowMessage.directive.ts the add below code

```
import { Directive,ElementRef,Renderer2,HostListener,Input} from
'@angular/core';

@Directive({
  selector: '[appShowmessage]'
})
export class ShowmessageDirective {
        @Input('appShowmessage') message!:string;
  constructor(private el: ElementRef,private render:Renderer2 )
  {
    render.setStyle(el.nativeElement,'cursor','pointer');

  }
  @HostListener('click') onClick(){
    this.el.nativeElement.innerHTML= this.message;
    this.render.setStyle(this.el.nativeElement,'color','red');
  }

}
```

Now Open the app.component.html then add below statement

```
<h3>College Information</h3>
<p [appShowmessage] = "myMessage">About Cse</p>
```

The run the application belwo command

> ng serve –open

College Information

About Cse

When we click the about cse then text will be change

Like below

College Information

240 Seats in computer science engineering..

4a. Module Name: Property Binding

Module Name: Property Binding

Binding image with class property using property binding

app.component.ts

```
export class AppComponent {

    imageurl ='assets/imgs/v.jpeg';

}
```

app.component.html

```
<img [src]="imageurl"/>
```

Output:

4b. Binding colspan attribute of a table element to the class property

app.component.ts

```
export class AppComponent {
    colspanvalue ="2";
  }
```

app.component.html

```
<table border="1" >
<tr>
  <td [attr.colspan]="colspanvalue"> CSE    </td>
  <td> IT </td></tr>
<tr>
  <td>ECE</td><td>EEE</td><td>MECH</td>
</tr>
</table>
```

Output:

CSE		IT
ECE	EEE	MECH

4c. Binding an element using inline style and user actions like entering text in input fields.

app.component.ts

```
export class AppComponent {

    isvalid=true;

  }
```

app.component.html

```
<button [style.color]="isvalid ? 'blue' : 'red' "> click </button>
<p [style.font-size.px]="isvalid ? 12 : 14"> font sie </p>
```
Output:

```
 click 
```

font size

5a. Display the product code in lowercase and product name in uppercase using built-in pipes

app.component.ts

```
export class AppComponent {
    title = "Product details";
    prodcutcode="prod_001";
    prodcutname="Laptop";
  }
```

app.component.html

```
<h3> {{ title | titlecase}} </h3>
<table style="text-align:left">
<tr><th> Product Code </th>
<td> {{ productcode | lowercase }} </td></tr>
<tr><th> Product Name </th>
<td> {{ productname | uppercase }} </td></tr>
</table>
```

Output:

Product Details

Product Code prod_001
Product Name LAPTOP

5b.Passing Parameters to Pipes. Apply built-in pipes with parameters to display product details

app.component.ts

```
export class AppComponent {
  productCode = 'PROD_P001';
  productName = 'Apple MPTT2 MacBook Pro';
  productPrice = 217021;
  purchaseDate = '1/17/2018';
  productTax = '0.1';
  productRating = 4.92;
  }
```

app.component.html

```
<table style="text-align:left">
    <tr>
        <th> Product Code </th>
        <td> {{ productCode | slice:5:9 }} </td>
    </tr>
    <tr>
        <th> Product Name </th>
        <td> {{ productName | uppercase }} </td>
    </tr>
    <tr>
        <th> Product Price </th>
        <td> {{ productPrice | currency: 'INR':'symbol' }} </td>
    </tr>
    <tr>
        <th> Purchase Date </th>
        <td> {{ purchaseDate | date:'fullDate' | lowercase}} </td>
    </tr>
    <tr>
        <th> Product Tax </th>
        <td> {{ productTax | percent : '.2' }} </td>
    </tr>
    <tr>
        <th> Product Rating </th>
        <td>{{ productRating | number:'1.3-5'}} </td>
    </tr>
</table>
```

OutPut:

Product Details

Product Code	P001
Product Name	APPLE MPTT2 MACBOOK PRO
Product Price	₹217,021.00
Purchase Date	wednesday, january 17, 2018
Product Tax	10.00%
Product Rating	4.920

5c. Nested Components Basics

Load Course List Component in the root component when a user click on the view courses list button.

Step 1: create courselist component

E:\Angular\myapp>ng generate component courselist

CREATE src/app/courselist/courselist.component.html (25 bytes)

CREATE src/app/courselist/courselist.component.spec.ts (587 bytes)

CREATE src/app/courselist/courselist.component.ts (218 bytes)

CREATE src/app/courselist/courselist.component.css (0 bytes)

UPDATE src/app/app.module.ts (886 bytes)

Step2: Open courselist.component.ts

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

@Component({
  selector: 'app-courselist',
  templateUrl: './courselist.component.html',
  styleUrls: ['./courselist.component.css']
})
export class CourselistComponent {
  courses = [{courseid:1,coursename:'nodejs'},
            {courseid:2,coursename:'reactjs'}
];

}
```
Step3: Open courselist.component.html

```
<table border="1">
    <thead>
      <tr>
        <th>Course ID</th>
        <th>Course Name</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let course of courses">
        <td>{{ course.courseid }}</td>
        <td>{{ course.coursename }}</td>
      </tr>
    </tbody>
  </table>
```

Step4:- Open app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
      show!:boolean;
      }
```

Step5:- Open app.component.html

```
<button (click)="show = true">View Courses list</button><br /><br />
<div *ngIf="show">
<app-courselist></app-courselist>
  </div>
```

Step6:- run the application

View Courses list

below table will show
after click the view
courses list (below table
from nested
components)

courselist works!

Course ID	Course Name
1	nodejs
2	reactjs

6.a Create an APPComponent that displays a dropdown with a list of courses as values in it. Create another component called the coursesList component and load it in AppCOmponent which should display the course details . when the user selects a course .

Ans:

Already we create the courselist component

Open courselist.component.ts add below code :

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

@Component({
  selector: 'app-courselist',
  templateUrl: './courselist.component.html',
  styleUrls: ['./courselist.component.css']
})
export class CourselistComponent {
  courses = [{courseid:1,coursename:'NodeJS'},
            {courseid:2,coursename:'ReactJS'},
            {courseid:3,coursename:'AngularJS'}
            ];
            course!: any[];
            @Input() set cName(name: string) {
              this.course = [];
              for (var i = 0; i < this.courses.length; i++) {
                if (this.courses[i].coursename === name) {
                  this.course.push(this.courses[i]);
                }
              }
            }
}
```

Then open courselist.component.html and add below code

```html
<p>courselist works!</p>

<table border="1" *ngIf="course.length > 0">
    <thead>
      <tr>
        <th>Course ID</th>
        <th>Course Name</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let c of course">
        <td>{{ c.courseid }}</td>
        <td>{{ c.coursename }}</td>
```

```
        </tr>
      </tbody>
    </table>
```
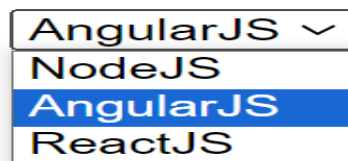
Then open app.component.ts add below property

```
export class AppComponent {
name!: string;
  }
```

Then open app.component.html add below code

```
 Select a course to view
<select #course (change)="name = course.value">
  <option value="NodeJS">NodeJS</option>
  <option value="AngularJS">AngularJS</option>
  <option value="ReactJS">ReactJS</option></select><br /><br />
  <app-courselist [cName]="name"></app-courselist>
<router-outlet></router-outlet>
<app-test></app-test>
<app-dir></app-dir>
```

Select a course to view AngularJS ⌄
                         NodeJS
courselist works!   **AngularJS**
                         ReactJS

Course ID	Course Name
3	AngularJS

when we select
AngularJS . we find
CourseID and Course
Name of AngulaJs

6b) Passing Data from child component to container component

Create an AppComponent that loads another component called the course List Component. Create another component called course list component which should display the courses list in a table along with a register.

Step1: create component course-list

Then open course-list.ts then add below code

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
@Component({
  selector: 'app-courses-list',
  templateUrl: './courses-list.component.html',
  styleUrls: ['./courses-list.component.css']
})
export class CoursesListComponent {
  @Output() registerEvent = new EventEmitter<string>();
  courses = [
    { courseId: 1, courseName: 'Node JS' },
    { courseId: 2, courseName: 'Typescript' },
    { courseId: 3, courseName: 'Angular' },
    { courseId: 4, courseName: 'React JS' }
  ];
  register(courseName: string) {
    this.registerEvent.emit(courseName);
  }
}
```

Step2: now open course-list.component.html

```
<table border="1">
  <thead>
    <tr>
      <th>Course ID</th>
```

```html
    <th>Course Name</th>
    <th></th>
  </tr>
 </thead>
 <tbody><tr *ngFor="let course of courses">
    <td>{{ course.courseId }}</td>
    <td>{{ course.courseName }}</td>
    <td><button (click)="register(course.courseName)">Register</button></td>
  </tr>
 </tbody>
</table>
```

Step3: Then open App.component.html then add below code

```html
<h2>Courses List</h2>
<app-courses-list (registerEvent)="courseReg($event)"></app-courses-list>
<br /><br />
<div *ngIf="message">{{ message }}</div>
```

Step4: Then open App.component.ts then add below code

```typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
 message!: string;
 courseReg(courseName: string) {
   this.message = `Your registration for ${courseName} is successful`;
 }
}
```

Output:

Course ID	Course Name	
1	Node JS	Register
2	Typescript	Register
3	Angular	Register
4	React JS	Register

Your registration for Node JS is successful

6c) Apply Shadow DOM and Node encapsulation modes to component

Create Component Name called Component1. Using below command

➢ ng g c component1

Open component1.css

```
.cmp {
   padding: 6px;
   margin: 6px;
   border: blue 2px solid;
 }
```

Opent Component1.html

```
<p>component1 works!</p>
<div class="cmp">First Component</div>
```

Create Component Name called Component2. Using below command

> ng g c component1

Open component2.css

```
.cmp {
  padding: 6px;
  margin: 6px;
  border: blue 2px solid;
}
```

Opent Component1.html

```
<p>component1 works!</p>
<div class="cmp">Second Component</div>
```

Now open the appcomponent.html add below script

```
<div class="cmp">
  App Component
  <app-component1></app-component1>
  <app-component2></app-component2>
</div>
```

Output:

6d. Override component life cycle hooks and logging the corresponding message to understand the flow

Component life cycle:

What is shadow DOM:-

Shadow DOM is a feature of the Web Components standard that allows you to create encapsulated DOM trees. This means that you can create a DOM tree that is separate from the main DOM tree of the document, and that styles and scripts applied to the main DOM tree will not affect the shadow DOM tree.

Component Life Cycle:

Every component has a life-cycle, a number of different stages it goes through from

Initializing to destroying. There are 8 different stages in the component lifecycle. Every stage is called life cycle hook events so we can use these hook events in different phases of our applications to obtains fine controls on the components.

Since a component is a typescript class, for that reason every component must have a constructor method.

Interface	Hook	Support
OnChanges	ngOnChanges	Directive, Component
OnInit	ngOnInit	Directive, Component
DoCheck	ngDoCheck	Directive, Component
AfterContentInit	ngAfterContentInit	Component
AfterContentChecked	ngAfterContentChecked	Component
AfterViewInit	ngAfterViewInit	Component
AfterViewChecked	ngAfterViewChecked	Component
OnDestroy	ngOnDestroy	Directive, Component
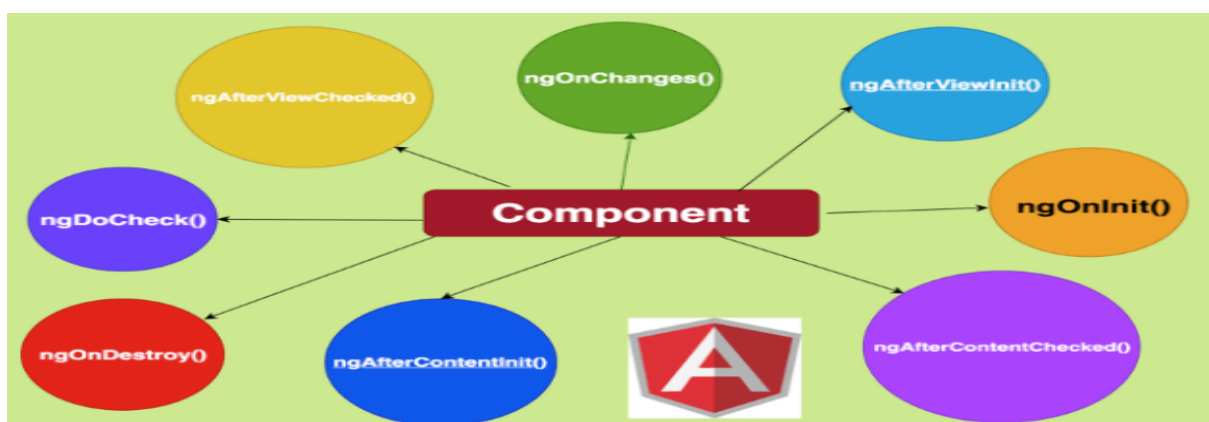
Open app.component add below code :

```
import {
  Component, OnInit, DoCheck, AfterContentInit, AfterContentChecked,
  AfterViewInit, AfterViewChecked,
  OnDestroy
} from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  templateUrl: './app.component.html'
})
export class AppComponent implements OnInit, DoCheck,
  AfterContentInit, AfterContentChecked,
  AfterViewInit, AfterViewChecked,
  OnDestroy {
  data = 'Angular';
  ngOnInit() {
    console.log('Init');
  }
  ngDoCheck(): void {
    console.log('Change detected');
  }
  ngAfterContentInit(): void {
    console.log('After content init');
  }
  ngAfterContentChecked(): void {
    console.log('After content checked');
```

```
  }
  ngAfterViewInit(): void {
    console.log('After view init');
  }
  ngAfterViewChecked(): void {
    console.log('After view checked');
  }
  ngOnDestroy(): void {
    console.log('Destroy');
  }
}
```

2. Write the below-given code in app.component.html

```html
<div>
  <h1>I'm a container component</h1>
  <input type="text" [(ngModel)]="data" />
  <app-child [title]="data"></app-child>
</div>
```

3. Write the below-given code in child.component.ts

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

```
@Component({

  selector: 'app-child',

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

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

})

export class ChildComponent implements OnChanges {

  @Input() title!: string;

  ngOnChanges(changes: any): void {

    console.log('changes in child:' + JSON.stringify(changes));

  }

}
```
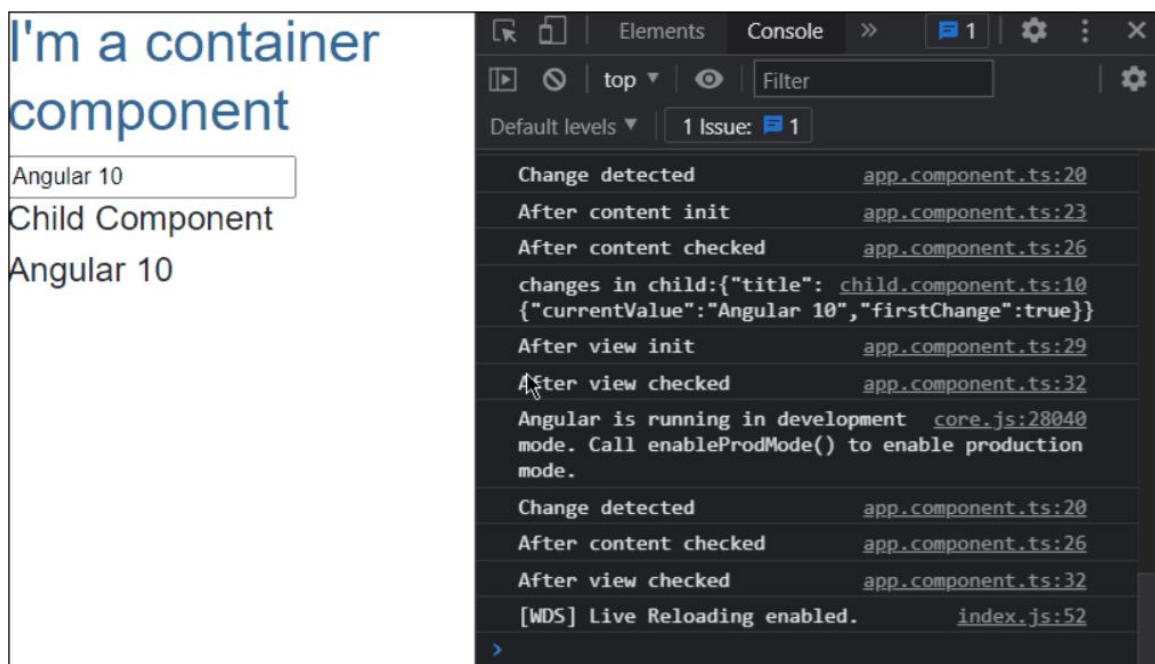
4. Write the below-given code in child.component.html

```
<h2>Child Component</h2>
<h2>{{title}}</h2>
```

Output:

7a) Create a course registration form as a template-driven form.

Add the following code in the registration-form.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
@Component({
  selector: 'app-registration-form',
  templateUrl: './registration-form.component.html',
  styleUrls: ['./registration-form.component.css']
})
export class RegistrationFormComponent implements OnInit {
  registerForm!: FormGroup;
  submitted!:boolean;

  constructor(private formBuilder: FormBuilder) { }
  ngOnInit() {
    this.registerForm = this.formBuilder.group({
      firstName: ['', Validators.required],
      lastName: ['', Validators.required],
      address: this.formBuilder.group({
        street: [],
        zip: [],
        city: []
      })
    });
  }
```

```
    }
```

Line 2: Import FormBuilder class to create a reactive form. Also, import FormGroup class to create a group of form controls and Validators for validation

Line 11: Create a property registerForm of type FormGroup

Line 14: Inject a FormBuilder instance using constructor

Line 17: formBuilder.group() method creates a FormGroup. It takes an object whose keys are FormControl names and values are their definitions

Line 18-24: Create form controls such as firstName, lastName, and address as a subgroup with fields street, zip, and city. These fields are form controls.

For each form control:

you can mention the default value as the first argument and

the list of validators as the second argument:

Validations can be added to the form controls using the built-in validators supplied by the Validators class.

For example: Configure built-in required validator for each control using [' ', Validators.required] syntax.

If multiple validators are to be applied, then use the syntax [' ', [Validators.required, Validators.maxlength(10)]].

registration-form.component.html

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

    <h1>Registration Form</h1>

    <form [formGroup]="registerForm">
```

```html
<div class="form-group">
  <label>First Name</label>
  <input type="text" class="form-control"
  formControlName="firstName">
  <div *ngIf="registerForm.controls['firstName'].errors" class="alert
  alert-danger">
      Firstname field is invalid.
      <p *ngIf="registerForm.controls['firstName'].errors?.['required']">
        This field is required!
      </p>
  </div>
</div>
<div class="form-group">
  <label>Last Name</label>
  <input type="text" class="form-control"
  formControlName="lastName">
  <div *ngIf="registerForm.controls['lastName'].errors" class="alert
  alert-danger">
      Lastname field is invalid.
      <p *ngIf="registerForm.controls['lastName'].errors?.['required']">
        This field is required!
      </p>
  </div>
</div>
<div class="form-group">
  <fieldset formGroupName="address">
  <legend>Address:</legend>
      <label>Street</label>
      <input type="text" class="form-control" formControlName="street">
```

```
        <label>Zip</label>

        <input type="text" class="form-control" formControlName="zip">

        <label>City</label>

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

      </fieldset>

    </div>

    <button type="submit" class="btn btn-primary"
(click)="submitted=true">Submit</button>

  </form>

 <br/>

  <div [hidden]="!submitted">

   <h3> Employee Details </h3>

   <p>First Name: {{ registerForm.get('firstName')?.value }} </p>

   <p> Last Name: {{ registerForm.get('lastName')?.value }} </p>

   <p> Street: {{ registerForm.get('address.street')?.value }}</p>

   <p> Zip: {{ registerForm.get('address.zip')?.value }} </p>

   <p> City: {{ registerForm.get('address.city')?.value }}</p>

  </div>

 </div>
```

Registration Form

First Name

James

Last Name

Gosling

Address:

Street

ABC Street

Zip

457899

City

New York

Submit

Employee Details

First Name: James

Last Name: Gosling

Street: ABC Street

Zip: 457899

7b. Create an employee registration form as a reactive form.

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

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

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

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

import { RegistrationFormComponent } from './registration-form/registration-form.component';

@NgModule({

 declarations: [

  AppComponent,

  RegistrationFormComponent

 ],

```typescript
  imports: [

    BrowserModule,

    ReactiveFormsModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }
```

2. Create a component called RegistrationForm using the following CLI command

D:\MyApp>ng generate component RegistrationForm

3. Add the following code in the registration-form.component.ts file

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

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

@Component({

  selector: 'app-registration-form',

  templateUrl: './registration-form.component.html',

  styleUrls: ['./registration-form.component.css']

})

export class RegistrationFormComponent implements OnInit {

  registerForm!: FormGroup;

  submitted!: boolean;

  constructor(private formBuilder: FormBuilder) { }

  ngOnInit() {

    this.registerForm = this.formBuilder.group({

      firstName: ['', Validators.required],

      lastName: ['', Validators.required],

      address: this.formBuilder.group({
```

```
        street: [],

        zip: [],

        city: []

      })

    });

  }

}
```

4. Write the below-given code in registration-form.component.html

```html
<div class="container">
  <h1>Registration Form</h1>
  <form [formGroup]="registerForm">
    <div class="form-group">
      <label>First Name</label>
      <input type="text" class="form-control" formControlName="firstName" />
      <p
        *ngIf="(registerForm.controls.firstName.dirty ||
            registerForm.controls.firstName.touched ||
            registerForm.controls.firstName.pristine) &&
             registerForm.controls.firstName.errors"
        class="alert alert-danger">
          This field is required!
      </p>
    </div>
    <div class="form-group">
      <label>Last Name</label>
      <input type="text" class="form-control" formControlName="lastName" />
      <p
```

```
      *ngIf="(registerForm.controls.lastName.dirty ||
          registerForm.controls.lastName.touched||
          registerForm.controls.lastName.pristine) &&
          registerForm.controls.lastName.errors"
      class="alert alert-danger">
        This field is required!
    </p>
  </div>
  <div class="form-group">
    <fieldset formGroupName="address">
      <label>Street</label>
      <input type="text" class="form-control" formControlName="street" />
      <label>Zip</label>
      <input type="text" class="form-control" formControlName="zip" />
      <label>City</label>
      <input type="text" class="form-control" formControlName="city" />
    </fieldset>
  </div>
  <button type="submit" class="btn btn-primary" (click)="submitted = true">
    Submit
  </button>
</form>
<br/>
<div [hidden]="!submitted">
  <h3>Employee Details</h3>
  <p>First Name: {{ registerForm.controls.firstName.value }}</p>
  <p>Last Name: {{ registerForm.controls.lastName.value }}</p>
```

```html
<p>Street: {{ registerForm.controls.address.value.street }}</p>
<p>Zip: {{ registerForm.controls.address.value.zip }}</p>
<p>City: {{ registerForm.controls.address.value.city }}</p>
  </div>
</div>
```

5. Write the below-given code in registration-form.component.css

```css
.ng-valid[required]  {
 border-left: 5px solid #42A948; /* green */
}

.ng-invalid:not(form)  {
 border-left: 5px solid #a94442; /* red */
}
```

6. Write the below-given code in app.component.html

```html
<app-registration-form></app-registration-form>
```

Registration Form

First Name

Alex

Last Name

Paul

Address:

Street

Isb road

Zip

500032

City

Hyderabad

Submit

Employee Details

First Name: Alex

Last Name: Paul

Street: Isb road

Zip: 500032

7c) Create a custom validator for an email field in the employee registration form (reactive form)

1. Write a separate function in registration-form.component.ts for custom validation as shown below.

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

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

@Component({

  selector: 'app-registration-form',

  templateUrl: './registration-form.component.html',

  styleUrls: ['./registration-form.component.css']

})

export class RegistrationFormComponent implements OnInit {

  registerForm!: FormGroup;

  submitted!:boolean;


  constructor(private formBuilder: FormBuilder) { }

  ngOnInit() {

    this.registerForm = this.formBuilder.group({

      firstName: ['',Validators.required],

      lastName: ['', Validators.required],

      address: this.formBuilder.group({

        street: [],

        zip: [],

        city: []

      }),
```

```
      email: ['',[Validators.required,validateEmail]]
    });
  }
}
function validateEmail(c: FormControl): any {
  let EMAIL_REGEXP = /^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-
Z]{2,5})$/;
  return EMAIL_REGEXP.test(c.value) ? null : {
    emailInvalid: {
      message: "Invalid Format!"
    }
  };
}
```

2. Add HTML controls for the email field in the registration-form.component.html file as shown below

```
<div class="container">
   <h1>Registration Form</h1>
   <form [formGroup]="registerForm">
    <div class="form-group">
      <label>First Name</label>
      <input type="text" class="form-control" formControlName="firstName">
        <div *ngIf="registerForm.controls['firstName'].errors" class="alert alert-
danger">
        Firstname field is invalid.
        <p *ngIf="registerForm.controls['firstName'].errors?.['required']">
```

```
      This field is required!
    </p>
  </div>
</div>
<div class="form-group">
  <label>Last Name</label>
  <input type="text" class="form-control" formControlName="lastName">
  <div *ngIf="registerForm.controls['lastName'].errors" class="alert alert-
danger">
      Lastname field is invalid.
      <p *ngIf="registerForm.controls['lastName'].errors?.['required']">
        This field is required!
      </p>
  </div>
</div>
<div class="form-group">
  <fieldset formGroupName="address">
    <legend>Address:</legend>
    <label>Street</label>
    <input type="text" class="form-control" formControlName="street">
    <label>Zip</label>
    <input type="text" class="form-control" formControlName="zip">
    <label>City</label>
    <input type="text" class="form-control" formControlName="city">
  </fieldset>
</div>
<div class="form-group">
  <label>Email</label>
```

```html
        <input type="text" class="form-control" formControlName="email" />
        <div *ngIf="registerForm.controls['email'].errors" class="alert alert-danger">
          Email field is invalid.
          <p *ngIf="registerForm.controls['email'].errors?.['required']">
            This field is required!
          </p>
          <p *ngIf="registerForm.controls['email'].errors?.['emailInvalid']">
            {{ registerForm.controls['email'].errors?.['emailInvalid'].message }}
          </p>
        </div>
      </div>
      <button type="submit" class="btn btn-primary" (click)="submitted=true">Submit</button>
    </form>
  <br/>
    <div [hidden]="!submitted">
    <h3> Employee Details </h3>
    <p>First Name: {{ registerForm.get('firstName')?.value }} </p>
    <p> Last Name: {{ registerForm.get('lastName')?.value }} </p>
    <p> Street: {{ registerForm.get('address.street')?.value }}</p>
    <p> Zip: {{ registerForm.get('address.zip')?.value }} </p>
    <p> City: {{ registerForm.get('address.city')?.value }}</p>
    <p>Email: {{ registerForm.get('email')?.value }}</p>
  </div>
</div>
```

Registration Form

First Name

Alex

Last Name

Paul

Address:

Street

lsb road

Zip

500032

City

Hyderabad

Email

alex.paul

Email field is invalid.
Invalid Format!

Submit

8a Create a custom validator for the email field in the course registration form

The code inside login.component.html is present under the login folder. Observe the creation of LoginComponent as a reactive form and addition of validations to it.

```html
<!-- Login form-->
<div class="container container-styles">
    <div class="col-xs-7 col-xs-offset-3">
        <div class="panel panel-primary">
            <div class="panel-heading">Login</div>
            <div class="panel-body padding">
                <form class="form-horizontal" [formGroup]="loginForm">
                    <div class="form-group" >
                        <label for="name" class="col-xs-4 control-label" style="text-align:left">User Name</label>
                        <div class="col-xs-8">
                            <input type="text" class="form-control" [ngClass]="{'valid':loginForm.controls['userName'].valid,
'invalid':loginForm.controls['userName'].invalid &&
!loginForm.controls['userName'].pristine}" formControlName="userName">
                            <div *ngIf="loginForm.controls['password'].errors && loginForm.controls['userName'].dirty">
                                <div *ngIf="loginForm.controls['userName'].errors?.['required']" style="color:red">UserName is required
                                </div>
                            </div>
                        </div>
                    </div>
                    <div class="form-group">
                        <label for="password" class="col-xs-4 control-label" style="text-align:left">Password</label>
                        <div class="col-xs-8">
```

```
                                                <input type="password" class="form-
control" [ngClass]="{'valid':loginForm.controls['password'].valid,
'invalid':loginForm.controls['password'].invalid &&
!loginForm.controls['password'].pristine}" formControlName="password">

                                                <div
*ngIf="loginForm.controls['password'].errors && loginForm.controls['password'].dirty">

                                                <div
*ngIf="loginForm.controls['password'].errors?.['required']" style="color:red">Password is
required

                                                </div>

                                        </div>

                                        </div>

                                </div>

                                <div *ngIf="!valid" class="error">Invalid
Credentials...Please try again...</div>

                                <br />

                                <div class="form-group">

                                        <span class="col-xs-4"></span>

                                        <div class="col-xs-3">

                                                <button (click)="onSubmit()" class="btn
btn-primary" [disabled]="!loginForm.valid">Login</button>

                                        </div>

                                        <span class="col-xs-5" style="top:8px">

                                                <a [routerLink]="['/welcome']"
style="color:#337ab7;text-decoration: underline;">Cancel</a>

                                        </span>

                                </div>

                        </form>

                </div>

        </div>

    </div>

</div>


Line 11: The username textbox is bound to the formControlName userName
```

Line 21: The password textbox is bound to the formControlName password

Line 12-16: Renders the given error message if required validation fails

Line 22-26: Similarly required validation is added to the password field and the corresponding error message will be displayed if the validation fails

Observe the creation of model-driven form within the LoginComponent. Open login.component.ts.

```
import { Component, ElementRef, OnInit, Renderer2, ViewChild } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { Router } from '@angular/router';
import { Login } from './Login';
import { LoginService } from './login.service';
@Component({
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  login = new Login();
  users: Login[] = [];
  valid = true;
  @ViewChild('uname') usernameElement!: ElementRef;
  loginForm!: FormGroup;
  constructor(private router: Router, private formBuilder: FormBuilder,
    private loginService: LoginService, private renderer: Renderer2) {
  }
  ngOnInit() {
```

```
    // Makes a service call to fetch users data from the backend

    this.loginService.getUsers().subscribe({next:users => this.users = users});

    this.loginForm = this.formBuilder.group({

        userName: [this.login.userName, Validators.required],

        password: [this.login.password, Validators.required]

    })

}

// Invoked when user clicks submit in login form

// Validates the credentials with the data fetched from the backend

onSubmit() {


    //fetches the form object containing the values of all the form controls

    this.login = this.loginForm.getRawValue();

    const user = this.users.filter(currUser => currUser.userName === this.login.userName
&& currUser.password === this.login.password)[0];

    if (user) {

        this.loginService.username = this.login.userName;

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

    } else {

        this.valid = false;

    }

}
}
```

8 b) Create a Book Component which fetches book details like id, name and displays them on the page in a list format. Store the book details in an array and fetch the data using a custom service.

1. Create BookComponent by using the following CLI command

D:\MyApp>ng generate component book

2. Create a file with the name book.ts under the book folder and add the following code.

```
export class Book {
   id!: number;
   name!: string;
}
```

3. Create a file with the name books-data.ts under the book folder and add the following code.

```
import { Book } from './book';
export let BOOKS: Book[] = [
   { id: 1, name: 'HTML 5' },
   { id: 2, name: 'CSS 3' },
   { id: 3, name: 'Java Script' },
   { id: 4, name: 'Ajax Programming' },
   { id: 5, name: 'jQuery' },
   { id: 6, name: 'Mastering Node.js' },
   { id: 7, name: 'Angular JS 1.x' },
   { id: 8, name: 'ng-book 2' },
   { id: 9, name: 'Backbone JS' },
   { id: 10, name: 'Yeoman' }
];
```

4. Create a service called BookService under the book folder using the following CLI command

D:\MyApp\src\app\book>ng generate service book

5. Add the following code in book.service.ts

```
import { Injectable } from '@angular/core';
import { BOOKS } from './books-data';
@Injectable({
 providedIn: 'root'
})
export class BookService {
 getBooks() {
  return BOOKS;
 }
}
```

6. Add the following code in the book.component.ts file

```
import { Component, OnInit } from '@angular/core';
import { Book } from './book';
import { BookService } from './book.service';
@Component({
 selector: 'app-book',
 templateUrl: './book.component.html',
 styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {
 books!: Book[];
```

```
  constructor(private bookService: BookService) { }
  getBooks() {
   this.books = this.bookService.getBooks();
  }
  ngOnInit() {
   this.getBooks();
  }
}
```

7. Write the below-given code in book.component.html

```html
<h2>My Books</h2>
<ul class="books">
  <li *ngFor="let book of books">
    <span class="badge">{{book.id}}</span> {{book.name}}
  </li>
</ul>
```

8. Add the following code in book.component.css which has styles for books

```css
.books {
  margin: 0 0 2em 0;
  list-style-type: none;
  padding: 0;
  width: 13em;
}
.books li {
  cursor: pointer;
  position: relative;
  left: 0;
  background-color: #eee;
  margin: 0.5em;
  padding: 0.3em 0;
```

```css
  height: 1.5em;

  border-radius: 4px;

}

.books li:hover {

  color: #607d8b;

  background-color: #ddd;

  left: 0.1em;

}

.books .badge {

  display: inline-block;

  font-size: small;

  color: white;

  padding: 0.8em 0.7em 0 0.7em;

  background-color: #607d8b;

  line-height: 0.5em;

  position: relative;

  left: -1px;

  top: -4px;

  height: 1.8em;

  margin-right: 0.8em;

  border-radius: 4px 0 0 4px;

}
```

9. Add the following code in app.component.html

```html
<app-book></app-book>
```

My Books

1	HTML 5
2	CSS 3
3	Java Script
4	Ajax Programming
5	jQuery
6	Mastering Node.js
7	Angular JS 1.x
8	ng-book 2
9	Backbone JS
10	Yeoman

8c) Create and use an observable in Angular.

app.component.ts

```typescript
import { Component } from '@angular/core';
import { Observable } from 'rxjs';
@Component({
 selector: 'app-root',
 styleUrls: ['./app.component.css'],
 templateUrl: './app.component.html'
})
export class AppComponent {
 data!: Observable<number>;
 myArray: number[] = [];
 errors!: boolean;
 finished!: boolean;
 fetchData(): void {
  this.data = new Observable(observer => {
```

```
    setTimeout(() => { observer.next(11); }, 1000),

    setTimeout(() => { observer.next(22); }, 2000),

    setTimeout(() => { observer.complete(); }, 3000);

  });

  this.data.subscribe((value) => this.myArray.push(value),

    error => this.errors = true,

    () => this.finished = true);

 }

}
```

Line 2: imports Observable class from rxjs module

Line 11: data is of type Observable which holds numeric values

Line 16: fetchData() is invoked on click of a button

Line 17: A new Observable is created and stored in the variable data

Line 18-20: next() method of Observable sends the given data through the stream. With a delay of 1,2 and 3 seconds, a stream of numeric values will be sent. Complete() method completes the Observable stream i.e., closes the stream.

Line 22: Observable has another method called subscribe which listens to the data coming through the stream. Subscribe() method has three parameters. The first parameter is a success callback which will be invoked upon receiving successful data from the stream. The second parameter is an error callback which will be invoked when Observable returns an error and the third parameter is a complete callback which will be invoked upon successful streaming of values from Observable i.e., once complete() is invoked. After which the successful response, the data is pushed to the local array called myArray, if any error occurs, a Boolean value called true is stored in the errors variable and upon complete() will assign a Boolean value true in a finished variable.

app.component.html

```html
<b> Using Observables!</b>
<h6 style="margin-bottom: 0">VALUES:</h6>
<div *ngFor="let value of myArray">{{ value }}</div>
<div style="margin-bottom: 0">ERRORS: {{ errors }}</div>
<div style="margin-bottom: 0">FINISHED: {{ finished }}</div>
<button style="margin-top: 2rem" (click)="fetchData()">Fetch Data</button>
```

Line 4: ngFor loop is iterated on myArray which will display the values on the page

Line 6: {{ errors }} will render the value of errors property if any

Line 8: Displays finished property value when complete() method of Observable is executed

Line 10: Button click event is bound with fetchData() method which is invoked and creates an observable with a stream of numeric values

Output:

Using Observables!

VALUES:
11
ERRORS:
FINISHED:

Fetch Data

9a) Create an application for Server Communication using HttpClient

In the example used for custom services concept, add HttpModule to the app.module.ts to make use of HttpClient class.

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

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

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

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

import { BookComponent } from './book/book.component';

@NgModule({

  imports: [BrowserModule, HttpClientModule],

  declarations: [AppComponent, BookComponent],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }
```

Add the following code in book.service.ts file

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

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

import { Observable, throwError } from 'rxjs';

import { catchError, tap } from 'rxjs/operators';

import { Book } from './book';

@Injectable({

  providedIn:'root'

})

export class BookService {

 booksUrl = 'http://localhost:3020/bookList';

 constructor(private http: HttpClient) { }

 getBooks(): Observable<Book[]> {

  return this.http.get<Book[]>('http://localhost:3020/bookList').pipe(

   tap((data: any) => console.log('Data Fetched:' + JSON.stringify(data))),

   catchError(this.handleError));
```

```typescript
    }
    addBook(book: Book): Observable<any> {
      const options = new HttpHeaders({ 'Content-Type': 'application/json' });
      return this.http.post('http://localhost:3020/addBook', book, { headers: options }).pipe(
        catchError(this.handleError));
    }
    updateBook(book: Book): Observable<any> {
      const options = new HttpHeaders({ 'Content-Type': 'application/json' });
      return this.http.put<any>('http://localhost:3020/update', book, { headers: options }).pipe(
        tap((_: any) => console.log(`updated hero id=${book.id}`)),
        catchError(this.handleError)
      );
    }
    deleteBook(bookId: number): Observable<any> {
      const url = `${this.booksUrl}/${bookId}`;
      return this.http.delete(url).pipe(
        catchError(this.handleError));
    }
    private handleError(err: HttpErrorResponse): Observable<any> {
      let errMsg = '';
      if (err.error instanceof Error) {
        // A client-side or network error occurred. Handle it accordingly.
        console.log('An error occurred:', err.error.message);
        errMsg = err.error.message;
      } else {
        // The backend returned an unsuccessful response code.
        // The response body may contain clues as to what went wrong,
        console.log(`Backend returned code ${err.status}`);
        errMsg = err.error.status;
      }
```

```
    return throwError(()=>errMsg);
  }
}
```

Write the code given below in book.component.ts

```typescript
import { Component, OnInit } from '@angular/core';
import { BookService } from './book.service';
import { Book } from './book';
@Component({
 selector: 'app-book',
 templateUrl: './book.component.html',
 styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {
 title = 'Demo on HttpClientModule';
 books!: Book[];
 errorMessage!: string;
 ADD_BOOK!: boolean;
 UPDATE_BOOK!: boolean;
 DELETE_BOOK!: boolean;
 constructor(private bookService: BookService) { }
 getBooks() {
  this.bookService.getBooks().subscribe({
   next:  books => this.books = books,
   error:error => this.errorMessage = <any>error
  })
 }
 addBook(bookId: string, name: string): void {
  let id=parseInt(bookId)
  this.bookService.addBook({id, name })
   .subscribe({next:(book: any) => this.books.push(book)});
```

```
  }
  updateBook(bookId: string, name: string): void {
   let id=parseInt(bookId)
   this.bookService.updateBook({ id, name })
     .subscribe({next:(book: any) => this.books = book});
  }
  deleteBook(bookId: string): void {
   let id=parseInt(bookId)
   this.bookService.deleteBook(id)
     .subscribe({next:(book: any) => this.books = book});
  }
  ngOnInit() {
   this.getBooks();
  }
}
```

Write the code given below in book.component.html

```html
<h2>{{ title }}</h2>
<h2>My Books</h2>
<ul class="books">
  <li *ngFor="let book of books">
   <span class="badge">{{ book.id }}</span> {{ book.name }}
  </li>
</ul>
<button class="btn btn-primary" (click)="ADD_BOOK = true">Add Book</button> 
<button class="btn btn-primary" (click)="UPDATE_BOOK = true">Update
Book</button> 
<button class="btn btn-primary" (click)="DELETE_BOOK = true">Delete Book</button>
<br />
<div *ngIf="ADD_BOOK">
 <table>
  <tr>
```

```html
      <td>Enter Id of the book:</td>
      <td>
       <input type="number" #id />
      </td>
    </tr>
    <br />
    <tr>
      <td>Enter Name of the Book:</td>
      <td>
       <input type="text" #name />
       <br />
      </td>
    </tr>
    <br />
    <tr>
      <td>
       <button class="btn btn-primary" (click)="addBook(id.value, name.value); ADD_BOOK
= false">
        Add Record
       </button>
      </td>
    </tr>
   </table>
   <br />
</div>
<div *ngIf="UPDATE_BOOK">
   <table>
    <tr>
      <td>Enter Id of the book:</td>
      <td>
       <input type="number" #id />
```

```html
        </td>
      </tr>
      <br />
      <tr>
        <td>Enter Name of the Book:</td>
        <td>
          <input type="text" #name />
          <br />
        </td>
      </tr>
      <br />
      <tr>
        <td>
          <button class="btn btn-primary" (click)="updateBook(id.value, name.value);
UPDATE_BOOK = false">
            Update Record
          </button>
        </td>
      </tr>
    </table>
</div>
<br />
<div *ngIf="DELETE_BOOK">
  <table>
    <tr>
      <td>Enter Id of the book:</td>
      <td>
        <input type="number" #id />
      </td>
    </tr>
    <br />
```

```html
    <tr>

     <td>

       <button class="btn btn-primary" (click)="deleteBook(id.value); DELETE_BOOK =
false">

         Delete Record

       </button>

     </td>

   </tr>

 </table>

</div>

<div class="error" *ngIf="errorMessage">{{ errorMessage }}</div>
```

My Books

1	HTML 5
2	CSS3
3	Java Script
4	Ajax Programming
5	jQuery
6	Node JS
7	Angular JS 1.x
8	ng-book 2
9	Backbone JS
10	Yeoman
11	Vue JS

Add Book	Update Book	Delete Book

9b.Create a custom service called ProductService in which Http class is used to fetch data stored in the JSON files.

Observe the data present inside the JSON files.

mobiles.json

```json
[

    {

        "productId": 1,
```

```json
            "productName": "Samsung Galaxy Note 7",
            "productCode": "MOB-120",
            "description": "64GB, Coral Blue",
            "price": 800,
            "imageUrl": "assets/imgs/samsung_note7_coralblue.jpg",
            "manufacturer": "Samsung",
            "ostype": "Android",
            "rating": 4
    },
    {
            "productId": 2,
            "productName": "Samsung Galaxy Note 7",
            "productCode": "MOB-124",
            "description": "64GB, Gold",
            "price": 850,
            "imageUrl": "assets/imgs/samsung_note7_gold.jpg",
            "manufacturer": "Samsung",
            "ostype": "Android",
            "rating": 4
    },
    ...
]
```

tablets.json

```json
[
```

```json
    {
        "productId": 1,
        "productName": "Apple iPad Mini 2",
        "productCode": "TAB-120",
        "description": "16GB, White",
        "price": 450,
        "imageUrl": "assets/imgs/apple_ipad_mini.jpg",
        "manufacturer": "Apple",
        "ostype": "iOS",
        "rating": 4
    },
    {
        "productId": 2,
        "productName": "Apple iPad Air2",
        "productCode": "TAB-124",
        "description": "64GB, Black",
        "price": 600,
        "imageUrl": "assets/imgs/ipad_air.jpg",
        "manufacturer": "Apple",
        "ostype": "iOS",
        "rating": 3
    },
    ...
]
```

Explore the methods present in the product.service.ts file from the products folder. Observe the code given below:

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

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

import { Observable, throwError } from 'rxjs';

import { catchError, map, tap } from 'rxjs/operators'

import { Product } from './product';

@Injectable()

export class ProductService {

    selectedProducts: any = [];

    products: any = [];

    producttype="tablet";

    username: string = '';

    // Fetches selectedProducts data from the sessionStorage

    constructor(private http: HttpClient) {

        if (sessionStorage.getItem('selectedProducts')) {

            this.selectedProducts =
JSON.parse(sessionStorage.getItem('selectedProducts') + '');

        }

    }

    // Makes a get request to backend to fetch products data

    getProducts(): Observable<Product[]> {

        if (this.producttype === 'tablet') {

            return this.http.get<Product[]>('./assets/products/tablets.json').pipe(

                tap((products) => this.products = products),

                catchError(this.handleError));

        } else if (this.producttype === 'mobile') {
```

```typescript
      return this.http.get<Product[]>('./assets/products/mobiles.json').pipe(

          tap((products) => this.products = products),

          catchError(this.handleError));

    }

    else

     throw new Error();

  }

  // Fetches the selected product details

  getProduct(id: number): Observable<Product> {

    return this.getProducts().pipe(

        map(products => products.filter(product => product.productId ===
id)[0]));

  }

  // Error Handling code

  private handleError(err: HttpErrorResponse) {

    return throwError(() => err.error() || 'Server error');

  }

}
```

Line 2-5: HttpClient, HttpErrorResponse, Observable and its operators are imported

Line 9: @Injectable() decorator makes the class as a service class which can be injected into other classes in the application

Line 18: Injecting HttpClient class to make asynchronous calls to JSON files

Line 25: getProducts() method contains functionality to fetch products data from JSON files

Line 26: Based on the product type selected, it makes an HTTP call to tablets.json or mobiles.json

Line 40-43: getProduct() method is to fetch particular product details from the JSON file

Line 46-48: handleError() is an error-handling method that throws the error back to the component

Now open products.module.ts to explore adding service class to the module

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

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

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

import { ProductsRoutingModule } from './products-routing.module';

import { ProductListComponent } from './product-list/product-list.component';

import { ProductDetailComponent } from './product-detail/product-detail.component';

import { CartComponent } from './cart/cart.component';

import { OrderByPipe } from './product-list/orderby.pipe';

import { RatingComponent } from './rating.component';

import { ProductService } from './product.service';

import { AuthGuardService } from './auth-guard.service';

@NgModule({

 imports: [

  CommonModule,
```

```
    FormsModule,

    ProductsRoutingModule

  ],

  declarations:
[ProductListComponent,ProductDetailComponent,CartComponent,OrderByPip
e,RatingComponent],

  providers:[ProductService,AuthGuardService]

})
export class ProductsModule { }
```

Line 11: Imports ProductService class

Line 21: Add it to the providers' property to make it available in the entire module

Now open the product-list.component.ts file to explore injecting a product service class

```
import { AfterViewInit, Component, ElementRef, OnInit, Renderer2,
ViewChild } from '@angular/core';

import { ProductService } from '../product.service';

import { Cart } from '../cart/Cart';

import { Product } from '../product';

import { LoginService } from 'src/app/login/login.service';

@Component({

  templateUrl: 'product-list.component.html',

  styleUrls: ['product-list.component.css']
```

```typescript
})
export class ProductListComponent implements OnInit, AfterViewInit {
  chkman: any = [];
  chkmanos: any = [];
  rate: number = 0;
  pageTitle = 'mCart';
  imageWidth = 80;
  imageHeight = 120;
  imageMargin = 12;
  showImage = false;
  listFilter: string = '';

  manufacturers = [{ 'id': 'Samsung', 'checked': false },
  { 'id': 'Microsoft', 'checked': false },
  { 'id': 'Apple', 'checked': false },
  { 'id': 'Micromax', 'checked': false }
  ];
  os = [{ 'id': 'Android', 'checked': false },
  { 'id': 'Windows', 'checked': false },
  { 'id': 'iOS', 'checked': false }];
  price_range = [{ 'id': '300-450', 'checked': false },
  { 'id': '450-600', 'checked': false },
  { 'id': '600-800', 'checked': false },
  { 'id': '800-1000', 'checked': false }];
  errorMessage: string = '';
  products: any = [];
  selectedItems: any = 0;
```

```typescript
  cart!: Cart;

  total = 0;

  orderId = 0;

  selectedManufacturers: string[] = [];

  selectedOStypes: string[] = [];

  selectedPrice: string[] = [];

  checkedManufacturers: any[] = [];

  checkedOS: any[] = [];

  checkedPrice: any[] = [];

  sub: any;

  i = 0;

  sortoption = '';

  chkmanosprice: any = [];

  @ViewChild('loginEl')

  loginVal!: ElementRef;

  @ViewChild('welcomeEl')

  welcomeVal!: ElementRef;

  // Fetches the products data from service class

  constructor(private productService: ProductService, private loginService:
LoginService, private renderer: Renderer2) {

  }

  ngAfterViewInit() {

    this.loginVal = this.loginService.loginElement;

    this.welcomeVal = this.loginService.welcomeElement;

    this.renderer.setProperty(this.loginVal.nativeElement, 'innerText',
'Logout');

    this.renderer.setStyle(this.welcomeVal.nativeElement, 'display', 'inline');

    let welcomeText="Welcome "+this.loginService.username+ "  ";
```

```
    this.renderer.setProperty(this.welcomeVal.nativeElement, 'innerText',
welcomeText);

    this.renderer.setStyle(this.welcomeVal.nativeElement, 'color', '#ff0080');

  }

  ngOnInit() {

    this.orderId++;

    this.productService.getProducts()

      .subscribe({

        next:products => {

          this.productService.products = products;

          this.products = this.productService.products;

          this.chkmanosprice =this.products

        },

        error:error => this.errorMessage = error});

    if (this.productService.selectedProducts.length > 0) {

      this.selectedItems = Number(sessionStorage.getItem('selectedItems'));

      this.total = Number(sessionStorage.getItem('grandTotal'));

    }

  }

  checkManufacturers(cManuf: any[], cProducts: any[], chkman: any[]) {

    if (cManuf.length > 0) {

      for (let checkManuf of cManuf) {

        for (let checkProd of cProducts) {

          if (checkProd.manufacturer.toLowerCase() ===
checkManuf.toLowerCase()) {

            this.chkman.push(checkProd);

          }

        }
```

```
        }
      } else {
        this.chkman = cProducts;
      }
    }
    checkOpsystem(cOS: any[], chkman: any[], chkmanos: any[]) {
      if (cOS.length > 0) {
        for (let checkOS of cOS) {
          for (let chkmann of chkman) {
            if (chkmann.ostype.toLowerCase() === checkOS.toLowerCase()) {
              this.chkmanos.push(chkmann);
            }
          }
        }
      } else {
        this.chkmanos = chkman;
      }
    }
    checkPrices(checkedPrice: any[], chkmanosprice: any[], chkmanos: any[]) {
      if (checkedPrice.length > 0) {
        for (let checkPrice of checkedPrice) {
          for (let chkmanfos of chkmanos) {
            if (checkPrice === '300-450') {
              if (chkmanfos.price >= 300 && chkmanfos.price <= 450) {
                this.chkmanosprice.push(chkmanfos);
              }
            }
```

```
                if (checkPrice === '450-600') {
                    if (chkmanfos.price > 450 && chkmanfos.price <= 600) {
                        this.chkmanosprice.push(chkmanfos);
                    }
                }
                if (checkPrice === '600-800') {
                    if (chkmanfos.price > 600 && chkmanfos.price <= 800) {
                        this.chkmanosprice.push(chkmanfos);
                    }
                }
                if (checkPrice === '800-1000') {
                    if (chkmanfos.price > 800 && chkmanfos.price <= 1000) {
                        this.chkmanosprice.push(chkmanfos);
                    }
                }
            }
        }
    } else {
        this.chkmanosprice = chkmanos;

    }
}
// filtering functionality
filter(name: any) {
    let checkedProducts: any[];
    this.chkman = [];
    this.chkmanos = [];
```

```
    this.chkmanosprice = [];

    const index = 0;

    checkedProducts = this.productService.products;



    name.checked = (name.checked) ? false : true;

    this.checkedManufacturers = this.manufacturers.filter(product =>
product.checked).map(product => product.id);

    this.checkedOS = this.os.filter(product => product.checked).map(product
=> product.id);

    this.checkedPrice = this.price_range.filter(product =>
product.checked).map(product => product.id);



    this.checkManufacturers(this.checkedManufacturers, checkedProducts,
this.chkman);

    this.checkOpsystem(this.checkedOS, this.chkman, this.chkmanos);

    this.checkPrices(this.checkedPrice, this.chkmanosprice, this.chkmanos);

    //this.products = [];

    this.products = this.chkmanosprice;

  }
  // Invoked when user clicks on Add to Cart button
  // Adds selected product details to service class variable
  // called selectedProducts
  addCart(id: number) {

    this.cart = new Cart();

    this.selectedItems += 1;

    // fetching selected product details

    const product = this.productService.products.filter((currProduct: any) =>
currProduct.productId === id)[0];
```

```
    this.total += product.price;

    sessionStorage.setItem('selectedItems', this.selectedItems);

    const sp = this.productService.selectedProducts.filter((currProduct: any)
=> currProduct.productId === id)[0];

    if (sp) {

      const index =
this.productService.selectedProducts.findIndex((currProduct: any) =>
currProduct.productId === id);

        this.productService.selectedProducts[index].quantity += 1;

        this.productService.selectedProducts[index].totalPrice += product.price;

    } else {

      this.cart.orderId = 'ORD_' + this.orderId;

      this.cart.productId = id;

      this.cart.userId = sessionStorage.getItem('username') + '';

      this.cart.productName = product.productName;

      this.cart.price = product.price;

      this.cart.quantity = 1;

      this.cart.dateOfPurchase = new Date().toString();

      this.cart.totalPrice = product.price * this.cart.quantity;

      this.productService.selectedProducts.push(this.cart);

      sessionStorage.setItem('selectedProducts',
JSON.stringify(this.productService.selectedProducts));

      this.orderId++;

    }

  }

  // Search box functionality

  // Searches based on manufacturer name

  searchtext() {

    this.products = this.productService.products;
```

```typescript
    if (this.listFilter.length > 0) {
      this.products = this.products.filter((product: Product) =>
        product.manufacturer.toLowerCase().indexOf(this.listFilter) !== -1);
    }
  }
  // Invoked when a tab (Tablets/Mobiles) is clicked
  // Displays tablets or mobiles data accordingly
  tabselect(producttype: string) {
    this.manufacturers = [{ 'id': 'Samsung', 'checked': false },
    { 'id': 'Microsoft', 'checked': false },
    { 'id': 'Apple', 'checked': false },
    { 'id': 'Micromax', 'checked': false }
    ];
    this.os = [{ 'id': 'Android', 'checked': false },
    { 'id': 'Windows', 'checked': false },
    { 'id': 'iOS', 'checked': false }];
    this.price_range = [{ 'id': '300-450', 'checked': false },
    { 'id': '450-600', 'checked': false },
    { 'id': '600-800', 'checked': false },
    { 'id': '800-1000', 'checked': false }];
    this.products = [];
    this.productService.producttype = producttype;
    this.productService.getProducts().subscribe({
      next: products => {
        this.products = products;
        this.sortoption='';
      },
```
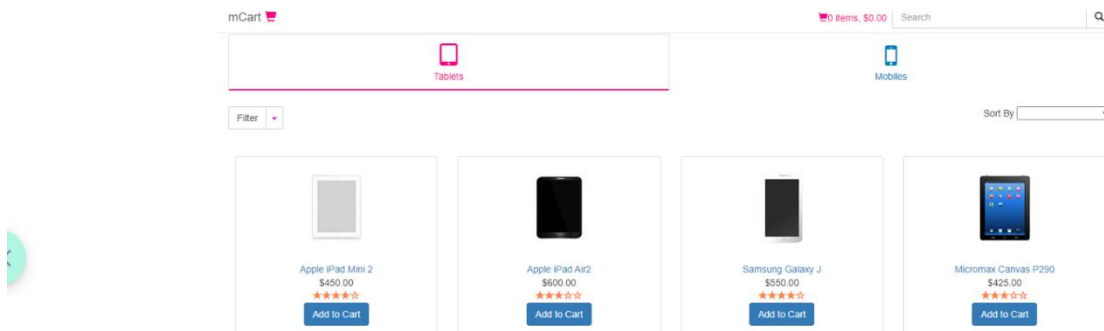
```
        error: error => this.errorMessage = error

    });


  }
  // Invoked when user select an option in sort drop down
  // changes the sortoption value accordingly
  onChange(value: string) {
      this.sortoption = value;
  }
}
```


- Here, the products list is displayed by making HTTP call to the JSON file.

- The JSON files are stored under the assets folder.
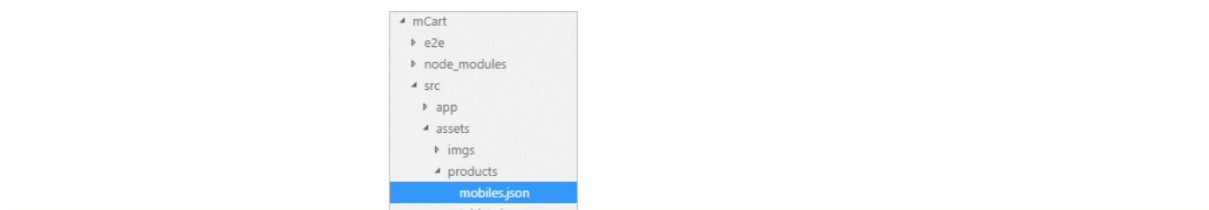
10a) Create multiple components and add routing to provide navigation between them.

1. Consider the example used for the HttpClient concept.

2. Create another component with the name dashboard using the following command

D:\MyApp>ng generate component dashboard

3. Open dashboard.component.ts and add the following code

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

import { Router } from '@angular/router';

import { Book } from '../book/book';

import { BookService } from '../book/book.service';

@Component({

  selector: 'app-dashboard',

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

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

})

export class DashboardComponent implements OnInit {

  books: Book[] = [];

  constructor(

    private router: Router,

    private bookService: BookService) { }

  ngOnInit(): void {

    this.bookService.getBooks()

      .subscribe({next:books => this.books = books.slice(1, 5)});

  }

  gotoDetail(book: Book): void {

    this.router.navigate(['/detail', book.id]);

  }

}
```

4. Open dashboard.component.html and add the following code

```html
<h3>Top Books</h3>
<div class="grid grid-pad">
  <div *ngFor="let book of books" (click)="gotoDetail(book)" class="col-1-4">
    <div class="module book">
      <h4>{{ book.name }}</h4>
    </div>
  </div>
</div>
```

5. Open dashboard.component.css and add the following code

```css
[class*="col-"] {
  float: left;
}
*,
*:after,
*:before {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}
h3 {
  text-align: center;
  margin-bottom: 0;
```

```css
    }
    [class*="col-"] {
      padding-right: 20px;
      padding-bottom: 20px;
    }
    [class*="col-"]:last-of-type {
      padding-right: 0;
    }
    .grid {
      margin: 0;
    }
    .col-1-4 {
      width: 25%;
    }
    .module {
      padding: 20px;
      text-align: center;
      color: #eee;
      max-height: 120px;
      min-width: 120px;
      background-color: #607d8b;
      border-radius: 2px;
    }
    h4 {
      position: relative;
    }
    .module:hover {
      background-color: #eee;
      cursor: pointer;
      color: #607d8b;
```

```
}
.grid-pad {
 padding: 10px 0;
}
.grid-pad > [class*="col-"]:last-of-type {
 padding-right: 20px;
}
@media (max-width: 600px) {
 .module {
  font-size: 10px;
  max-height: 75px;
 }
}
@media (max-width: 1024px) {
 .grid {
  margin: 0;
 }
 .module {
  min-width: 60px;
 }
}
```

6. Create another component called book-detail using the following command

D:\MyApp>ng generate component bookDetail

7. Open book.service.ts and add getbook() method as shown below to fetch specific book details

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

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

import { Observable, throwError } from 'rxjs';

import { catchError, tap, map} from 'rxjs/operators';

import { Book } from './book';

@Injectable({
  providedIn:'root'
})
export class BookService {
 booksUrl = 'http://localhost:3020/bookList';

 private txtUrl = './assets/sample.txt';

 constructor(private http: HttpClient) { }

 getBooks(): Observable<Book[]> {
  return this.http.get<any>(this.booksUrl, {observe:'response'}).pipe(
   tap((data: any) => console.log('Data Fetched:' + JSON.stringify(data))),
   catchError(this.handleError));
 }

 getBook(id: any) {
  return this.getBooks().pipe(
   map((books) => books.find((book) => book.id == id))
  );
 }

 addBook(book: Book): Observable<any> {
  const options = new HttpHeaders({ 'Content-Type': 'application/json' });

  return this.http.post('http://localhost:3020/addBook', book, { headers: options }).pipe(
   catchError(this.handleError));
 }

 updateBook(book: Book): Observable<any> {
  const options = new HttpHeaders({ 'Content-Type': 'application/json' });

  return this.http.put<any>('http://localhost:3020/update', book, { headers: options }).pipe(
```

```
      tap((_: any) => console.log(`updated hero id=${book.id}`)),
      catchError(this.handleError)
    );
  }
  deleteBook(bookId: number): Observable<any> {
   const url = `${this.booksUrl}/${bookId}`;
   return this.http.delete(url).pipe(
     catchError(this.handleError));
  }
  private handleError(err: HttpErrorResponse): Observable<any> {
   let errMsg = '';
   if (err.error instanceof Error) {
     // A client-side or network error occurred. Handle it accordingly.
     console.log('An error occurred:', err.error.message);
     errMsg = err.error.message;
    } else {
     // The backend returned an unsuccessful response code.
     // The response body may contain clues as to what went wrong,
     console.log(`Backend returned code ${err.status}`);
     errMsg = err.error.status;
   }
   return throwError(()=>errMsg);
  }
}


8. Open book-detail.component.ts and add the following code


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

```
import { BookService } from '../book/book.service';
@Component({
  selector: 'app-book-detail',
  templateUrl: './book-detail.component.html',
  styleUrls: ['./book-detail.component.css'],
})
export class BookDetailComponent implements OnInit {
  book!: Book;
  error!: any;
  constructor(
    private bookService: BookService,
    private route: ActivatedRoute
  ) { }
  ngOnInit() {
    this.route.paramsMap.subscribe(params => {
    this.bookService.getBook(params.get('id')).subscribe((book) => {
      this.book = book ?? this.book;
    });
    });
  }
  goBack() {
    window.history.back();
  }
}
```

9. Open book-detail.component.html and add the following code

```html
<div *ngIf="book">
  <h2>{{ book.name }} details!</h2>
  <div><label>id: </label>{{ book.id }}</div>
  <div>
    <label>name: </label> <input [(ngModel)]="book.name" placeholder="name" />
  </div>
  <button (click)="goBack()">Back</button>
</div>
```

10. Open book-detail.component.css and add the following code

```css
label {
  display: inline-block;
  width: 3em;
  margin: 0.5em 0;
  color: #607d8b;
  font-weight: bold;
}
input {
  height: 2em;
  font-size: 1em;
  padding-left: 0.4em;
}
button {
  margin-top: 20px;
  font-family: Arial;
  background-color: #eee;
  border: none;
  padding: 5px 10px;
```

```css
  border-radius: 4px;

  cursor: pointer;

  cursor: hand;

}

button:hover {

  background-color: #cfd8dc;

}

button:disabled {

  background-color: #eee;

  color: #ccc;

  cursor: auto;

}
```

11. Generate PageNotFound component using the following CLI command

D:\MyApp>ng g c PageNotFound

12. Add below code to page-not-found.component.html:

```html
<div>

   <h1>404 Error</h1>

   <h1>Page Not Found</h1>

</div>
```

13. Add the below code to app-routing.module.ts:

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

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

import { BookComponent } from './book/book.component';

import { DashboardComponent } from './dashboard/dashboard.component';

import { BookDetailComponent } from './book-detail/book-detail.component';

import { PageNotFoundComponent } from './page-not-found/page-not-found.component';
```

```
const appRoutes: Routes = [
   { path: 'dashboard', component: DashboardComponent },
   { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
   { path: 'books', component: BookComponent },
   { path: 'detail/:id', component: BookDetailComponent },
   { path: '**', component: PageNotFoundComponent },
];
@NgModule({
  imports: [
     RouterModule.forRoot(appRoutes)
  ],
  exports: [
     RouterModule
  ]
})
export class AppRoutingModule { }
```

14. Write the below-given code in app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
import { DashboardComponent } from './dashboard/dashboard.component';
import { BookDetailComponent } from './book-detail/book-detail.component';
import { AppRoutingModule } from './app-routing.module';
```

```
import { PageNotFoundComponent } from './page-not-found/page-not-found.component';
@NgModule({
  imports: [BrowserModule, HttpClientModule, FormsModule, AppRoutingModule],
  declarations: [AppComponent, BookComponent, DashboardComponent,
BookDetailComponent, PageNotFoundComponent],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

15. Write the below-given code in app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  templateUrl: './app.component.html'
})
export class AppComponent {
  title = 'Tour of Books';
}
```

16. Write the below-given code in app.component.html

```
<h1>{{title}}</h1>
<nav>
    <a [routerLink]='["/dashboard"]' routerLinkActive="active">Dashboard</a>
```

```html
    <a [routerLink]='["/books"]' routerLinkActive="active">Books</a>
</nav>
<router-outlet></router-outlet>
```

17. Open app.component.css and add the following code

```css
/* Master Styles */
h1 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
h2, h3 {
  color: #444;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: lighter;
}
body {
  margin: 2em;
}
body, input[text], button {
  color: #888;
  font-family: Cambria, Georgia;
}
a {
  cursor: pointer;
  cursor: hand;
}
button {
```

```css
  font-family: Arial;

  background-color: #eee;

  border: none;

  padding: 5px 10px;

  border-radius: 4px;

  cursor: pointer;

  cursor: hand;

}
button:hover {

  background-color: #cfd8dc;

}
button:disabled {

  background-color: #eee;

  color: #aaa;

  cursor: auto;

}
/* Navigation link styles */
nav a {

  padding: 5px 10px;

  text-decoration: none;

  margin-right: 10px;

  margin-top: 10px;

  display: inline-block;

  background-color: #eee;

  border-radius: 4px;

}
nav a:visited, a:link {

  color: #607D8B;

}
nav a:hover {
```

```css
  color: #039be5;

  background-color: #CFD8DC;

}

nav a.active {

  color: #039be5;

}

/* everywhere else */

* {

  font-family: Arial, Helvetica, sans-serif;

}
```

18. Open styles.css under the src folder and add the following code

```css
/* You can add global styles to this file, and also import other style files */

body{

    padding:10px;

}
```

19. Open book.component.ts file in book folder and add the following code

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

import { Book } from './book';

import { BookService } from './book.service';

@Component({

  selector: 'app-book',

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

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

})

export class BookComponent implements OnInit {

  books!: Book[];

  errorMessage!: string;

  constructor(private bookService: BookService) { }
```

```
  getBooks() {

   this.bookService.getBooks().subscribe({

     next:  books => this.books = books,

     error:error => this.errorMessage = <any>error

    })

  }


 ngOnInit(): void {

   this.getBooks();

 }

}
```

20. Open book.component.html and update with below code.

```
<h2>My Books</h2>

<ul class="books">

  <li *ngFor="let book of books">

    <span class="badge">{{ book.id }}</span> {{ book.name }}

  </li>

</ul>

<div class="error" *ngIf="errorMessage">{{ errorMessage }}</div>
```
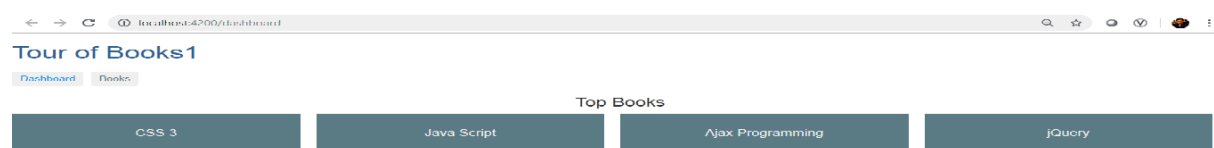
21. Save the files and check the output in the browser.

10b) Considering the same example used for routing, add route guard to BooksComponent. Only after logging in, the user should be able to access BooksComponent. If the user tries to give the URL of Bookscomponent in another tab or window, or if the user tries

Create a LoginComponent using Angular CLI

ng g c Login

Add the following code to the login.component.html file

```
<h3 style="position: relative; left: 60px">Login Form</h3>
<div *ngIf="invalidCredentialMsg" style="color: red">
  {{ invalidCredentialMsg }}
</div>
<br />
<div style="position: relative; left: 20px">
  <form [formGroup]="loginForm" (ngSubmit)="onFormSubmit()">
    <p>User Name <input formControlName="username" /></p>
    <p>
      Password
      <input
        type="password"
        formControlName="password"
        style="position: relative; left: 10px"
      />
    </p>
    <p><button type="submit">Submit</button></p>
  </form>
```

```html
</div>
```

Line 2: div tag will render error message for incorrect credentials

Line 7-18: A reactive form with two fields username and password is displayed

Add the following code to the login.component.ts file

```typescript
import { Component } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';
import { Router } from '@angular/router';
import { LoginService } from './login.service';
@Component({
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css'],
})
export class LoginComponent {
  invalidCredentialMsg!: string;
  loginForm!: FormGroup;
  constructor(
    private loginService: LoginService,
    private router: Router,
    private formbuilder: FormBuilder
  ) {
    this.loginForm = this.formbuilder.group({
      username: [],
```

```
    password: [],
  });
 }
 onFormSubmit(): void {
  const uname = this.loginForm.value.username;
  const pwd = this.loginForm.value.password;
  this.loginService
   .isUserAuthenticated(uname, pwd)
   .subscribe({next:(authenticated) => {
    if (authenticated) {
      this.router.navigate(['/books']);
    } else {
      this.invalidCredentialMsg = 'Invalid Credentials. Try again.';
    }
   }});
 }
}
```

Line 25: onFormSubmit() method is invoked when the submit button is clicked in Login Form

Line 26-27: Fetching username and password values from the form

Line 28: Invoking isUserAuthenticated method of LoginService class which will check for the validity of username and password values and returns a Boolean value

Line 31-35: If the response is true, it will navigate to BooksComponent else assigns an error message to invalidCredentialMsg property

Add the following code to the user.ts file inside Login folder.

```
export class User {
  constructor(public userId: number, public username: string, public password: string) { }
}
```

Line 1-3: A User model class with three properties userId, username, and password is created

Add the following code to the login.service.ts file present inside login folder.

```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';
import { map } from 'rxjs/operators';
import { User } from './user';
const USERS = [
  new User(1, 'user1', 'user1'),
  new User(2, 'user2', 'user2')
];
const usersObservable = of(USERS);
@Injectable({
  providedIn: 'root'
})
export class LoginService {
```

```
    private isloggedIn = false;
    getAllUsers(): Observable<User[]> {
        return usersObservable;
    }
    isUserAuthenticated(username: string, password: string):
Observable<boolean> {
        return this.getAllUsers().pipe(
            map(users => {
                const Authenticateduser = users.find(user => (user.username ===
username) && (user.password === password));
                if (Authenticateduser) {
                    this.isloggedIn = true;
                } else {
                    this.isloggedIn = false;
                }
                return this.isloggedIn;
            })
        );
    }
    isUserLoggedIn(): boolean {
        return this.isloggedIn;
    }
}
```

Line 4: Imports User model class

Line 6-9: Creates an array called USERS of type User

Line 10: Converts USERS array as an observable type

Line 18-20: getAllUsers() method returns users array in Observable manner

Line 21: isUserAuthenticated method takes username and password values as inputs and returns a Boolean value of type Observable

Line 22-32: Invokes getAllUsers() methods which returns an observable array. After receiving it, it will find the entered credentials exist in the array or not. If the user exists, assigns true value to isloggedIn property otherwise false value to it

Line 34-36: isUserLoggedIn() method returns the value of isloggedIn which we will use in LoginGuardService class

Create another service class called login-guard.service inside login folder and add the following code:

```
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';
import { LoginService } from './login.service';
@Injectable({
  providedIn: 'root'
})
export class LoginGuardService implements CanActivate {
  constructor(private loginService: LoginService, private router: Router) {
}
```

```
    canActivate(): boolean {

      if (this.loginService.isUserLoggedIn()) {

         return true;

      }

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

      return false;

   }

}
```

Line 8: Implements CanActivate interface to LoginGuardService class

Line 10: Overrides canActivate() method

Line 11-15: Invokes isUserLoggedIn method from LoginService class which returns a Boolean value representing whether a user is logged in or not. If the user logs in, canActivate returns true otherwise navigate to the login component asking the user to login first to access BooksComponent

Add the following code in app-routing.module.ts

```
...
const appRoutes: Routes = [

   { path: '', redirectTo: '/login', pathMatch: 'full' },

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

   { path: 'books', component: BookComponent,
canActivate:[LoginGuardService] },

   { path: 'dashboard', component: DashboardComponent},
```

```
    { path: 'detail/:id', component: BookDetailComponent},
    { path: '**', component: PageNotFoundComponent },
];
...
```

Line 5: Binds the LoginGuardService to the books path.

Update app.module.ts as below:


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

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

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

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

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

import { BookComponent } from './book/book.component';

import { DashboardComponent } from
'./dashboard/dashboard.component';

import { BookDetailComponent } from './book-detail/book-
detail.component';

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

import { PageNotFoundComponent } from './page-not-found/page-not-
found.component';

import { LoginComponent } from './login/login.component';

@NgModule({

  imports: [BrowserModule, HttpClientModule,
ReactiveFormsModule,FormsModule, AppRoutingModule],

  declarations: [AppComponent, LoginComponent, BookComponent,
DashboardComponent, BookDetailComponent,
PageNotFoundComponent],

  providers: [],
```

bootstrap: [AppComponent]

})

export class AppModule { }
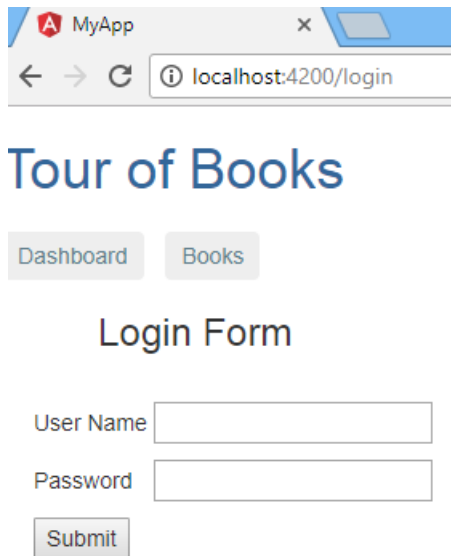
10c) Apply lazy loading to BookComponent. If lazy loading is not added to the demo, it has loaded in 1.14 s. Observe the load time at the bottom of the browser console. Press F12 in the browser and click the Network tab and check the Load time

1. Write the code given below in the book-routing.module.ts file inside book folder.

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

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

import { BookComponent } from './book.component';

import { LoginGuardService } from '../login/login-guard.service';

const bookRoutes: Routes = [

    {
```

```
      path: '',

      component: BookComponent,

      canActivate: [LoginGuardService]

   }

];

@NgModule({

  imports: [RouterModule.forChild(bookRoutes)],

  exports: [RouterModule]

})

export class BookRoutingModule { }
```

2. Create the book.module.ts file inside book folder and add the following code

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

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

import { BookComponent } from './book.component';

import { BookRoutingModule } from './book-routing.module';

@NgModule({

 imports: [CommonModule, BookRoutingModule],

 declarations: [BookComponent]

})

export class BookModule { }
```

3. Add the following code to the app-routing.module.ts file

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

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

import { BookDetailComponent } from './book-detail/book-detail.component';

import { BookComponent } from './book/book.component';

import { DashboardComponent } from './dashboard/dashboard.component';

import { LoginGuardService } from './login/login-guard.service';

import { LoginComponent } from './login/login.component';

import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

const appRoutes: Routes = [

  { path: '', redirectTo: '/login', pathMatch: 'full' },

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

  { path: 'books', loadChildren: () => import('./book/book.module').then(m => m.BookModule) },

  { path: 'dashboard', component: DashboardComponent },

  { path: 'detail/:id', component: BookDetailComponent} ,

  { path: '**', component: PageNotFoundComponent }

];

@NgModule({

  imports: [

    RouterModule.forRoot(appRoutes)

  ],

  exports: [

    RouterModule

  ]

})
```
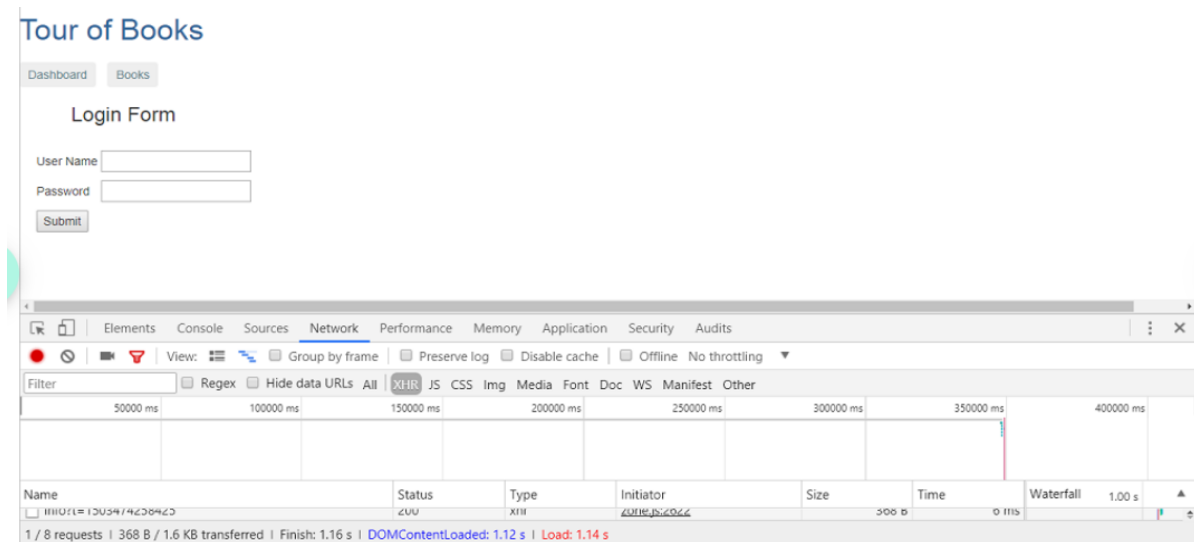
```typescript
export class AppRoutingModule { }
```

4. Add the following code to the app.module.ts file

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

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

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

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

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

import { BookComponent } from './book/book.component';

import { DashboardComponent } from './dashboard/dashboard.component';

import { BookDetailComponent } from './book-detail/book-detail.component';

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

import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

import { LoginComponent } from './login/login.component';

@NgModule({

  imports: [BrowserModule, HttpClientModule,
ReactiveFormsModule,FormsModule, AppRoutingModule],

  declarations: [AppComponent, LoginComponent, DashboardComponent,
BookDetailComponent, PageNotFoundComponent],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }
```

5. Save the files and check the output in the browser

10d) Implement Child Routes to a submodule

1. Write the following code in app.module.ts.

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

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

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

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

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

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

import { LoginComponent } from './login/login.component';

@NgModule({

  imports: [BrowserModule, HttpClientModule,  ReactiveFormsModule, AppRoutingModule],

  declarations: [AppComponent, LoginComponent],

  providers: [],
```

```
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2. Write the following code in app-routing.module.ts.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { LoginComponent } from './login/login.component';
import { PageNotFoundComponent } from './page-not-found/page-not-
found.component';
const appRoutes: Routes = [
    { path: '', redirectTo: '/login', pathMatch: 'full' },
    { path: 'login', component: LoginComponent },
    { path: 'books', loadChildren: () => import('./book/book.module').then(m =>
m.BookModule) },
    { path: '**', component: PageNotFoundComponent }
];
@NgModule({
    imports: [
        RouterModule.forRoot(appRoutes)
    ],
    exports: [
        RouterModule
    ]
})
```

```
export class AppRoutingModule { }
```

3. Write the following code in app.component.html

```
<h1>{{title}}</h1>
<nav>
   <a [routerLink]='["/books"]' routerLinkActive="active">Books</a>
   <a [routerLink]='["/books/dashboard"]'
routerLinkActive="active">Dashboard</a>
</nav>
<router-outlet></router-outlet>
```

4. Write the below code in book.module.ts

```
import { NgModule } from '@angular/core';
import { BookComponent } from './book.component';
import { BookRoutingModule } from './book-routing.module';
import { FormsModule } from '@angular/forms';
import { BookDetailComponent } from '../book-detail/book-detail.component';
import { DashboardComponent } from '../dashboard/dashboard.component';
import { CommonModule } from '@angular/common';
@NgModule({
 imports: [ CommonModule, BookRoutingModule, FormsModule],
```

```
  declarations: [BookComponent, BookDetailComponent,
DashboardComponent]
})
export class BookModule { }
```

5. Write the following code in book-routing.module.ts.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { BookComponent } from './book.component';
import { LoginGuardService } from '../login/login-guard.service';
import { DashboardComponent } from '../dashboard/dashboard.component';
import { BookDetailComponent } from '../book-detail/book-detail.component';
const bookRoutes: Routes = [
  {
    path: '',
    component: BookComponent,
    children: [
      { path: 'dashboard', component: DashboardComponent },
      { path: 'detail/:id', component: BookDetailComponent }
    ],
    canActivate: [LoginGuardService]
  }];
@NgModule({
  imports: [RouterModule.forChild(bookRoutes)],
  exports: [RouterModule]
```

```
})
export class BookRoutingModule { }
```

6. Write the below code in book.component.html

```html
<br/>
      <h2>MyBooks</h2>
      <ul class="books">
       <li *ngFor="let book of books " (click)="gotoDetail(book)">
            <span class="badge">{{book.id}}</span> {{book.name}}
       </li>
      </ul>
      <div>
       <router-outlet></router-outlet>
      </div>
      <div class="error" *ngIf="errorMessage">{{errorMessage}}</div>
```

7. Write the below code in book.component.ts:

```typescript
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { Book } from './book';
import { BookService } from './book.service';
```

```
@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {
  books: Book[]=[];
  errorMessage!: string;
  constructor(private bookService: BookService, private router: Router) { }
  getBooks() {
    this.bookService.getBooks().subscribe({
      next:  books => {console.log(books);this.books = books},
      error:error => this.errorMessage = <any>error
    })
  }
  gotoDetail(book: Book): void {
    this.router.navigate(['/books/detail/', book.id]);
  }
  ngOnInit(): void {
    this.getBooks();
  }
}
```

8. Update book-detail.component.html as below:

```
<div *ngIf="book">
  <h2>{{ book.name }} details!</h2>
```

```html
  <div><label>id: </label>{{ book.id }}</div>
  <div>
    <label>name: </label> <input [(ngModel)]="book.name"
placeholder="name" />
  </div>
  <button (click)="goBack()">Back</button>
</div>
```

9.Update book-detail.component.ts as below:

```typescript
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Book } from '../book/book';
import { BookService } from '../book/book.service';
@Component({
  selector: 'app-book-detail',
  templateUrl: './book-detail.component.html',
  styleUrls: ['./book-detail.component.css'],
})
export class BookDetailComponent implements OnInit {
  book!: Book;
  error!: any;
  constructor(
    private bookService: BookService,
    private route: ActivatedRoute
  ) { }
  ngOnInit() {
```

```
    this.route.paramMap.subscribe(params => {
    this.bookService.getBook(params.get('id')).subscribe((book) => {
     this.book = book ?? this.book;
    });
   });
 }
 goBack() {
  window.history.back();
 }
}
```

10. Update dashboard.component.html with below:

```html
<h3>Top Books</h3>
<div class="grid grid-pad">
  <div *ngFor="let book of books" (click)="gotoDetail(book)" class="col-1-4">
   <div class="module book">
    <h4>{{ book.name }}</h4>
   </div>
  </div>
</div>
```

11. Update dashboard.component.ts with below:

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

```
import { Book } from '../book/book';
import { BookService } from '../book/book.service';
@Component({
 selector: 'app-dashboard',
 templateUrl: './dashboard.component.html',
 styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
 books: Book[] = [];
 constructor(
  private router: Router,
  private bookService: BookService) { }
 ngOnInit(): void {
  this.bookService.getBooks()
   .subscribe(books => this.books = books.slice(1, 5));
 }
 gotoDetail(book: Book): void {
  this.router.navigate(['/books/detail', book.id]);
 }
}
```

12.Save the files and check the output in the browser.

Tour of Books

Books   Dashboard

Login Form

User Name user1

Password [··]   user1
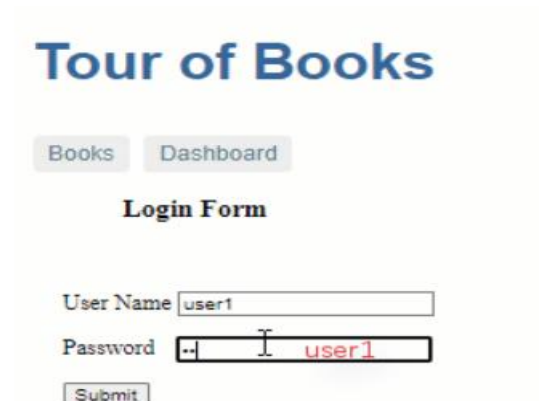
Submit

Click submit button

Tour of Books

Books Dashboard

MyBooks

1	HTML 5
2	CSS 3
3	Java Script
4	Ajax Programming
5	jQuery
6	Mastering Node.js
7	Angular JS 1.x
8	

11a). Install MongoDB and configure ATLAS

MongoDB Atlas

MongoDB Atlas is a cloud service by MongoDB. It is built for developers who'd rather spend time building apps than managing databases. This service is available on AWS, Azure, and GCP.

It is the worldwide cloud database service for modern applications that give best-in-class automation and proven practices guarantee availability, scalability, and compliance with the foremost demanding data security and privacy standards. We can use MongoDB's robust ecosystem of drivers, integrations, and tools to create faster and spend less time managing our database.

Advantages of MongoDB Atlas

- o **Global clusters for world-class applications:** Using MongoDB Atlas, we are free to choose the cloud partner and ecosystem that fit our business strategy.

- o **Secure for sensitive data:** It offers built-in security controls for all our data. It enables enterprise-grade features to integrate with our existing security protocols and compliance standard.

- o **Designed for developer productivity:** MongoDB Atlas moves faster with general tools to work with our data and a platform of services that makes it easy to build, secure, and extend applications that run on MongoDB.

- o **Reliable for mission-critical workload:** It is built with distributed fault tolerance and automated data recovery.

- o **Built for optimal performance:** It makes it easy to scale our databases in any direction. We can get more out of our existing resources with performance optimization tools and real-time visibility into database metrics.

- o **Managed for operational efficiency:** It comes with built-in operational best practices, so we can focus on delivering business value and accelerating application development instead of managing databases.

**Create an Atlas Account and deploying a Free Tier Cluster**

Step 1: Go to https://www.mongodb.com/cloud/atlas to register for an Atlas account to host your data.
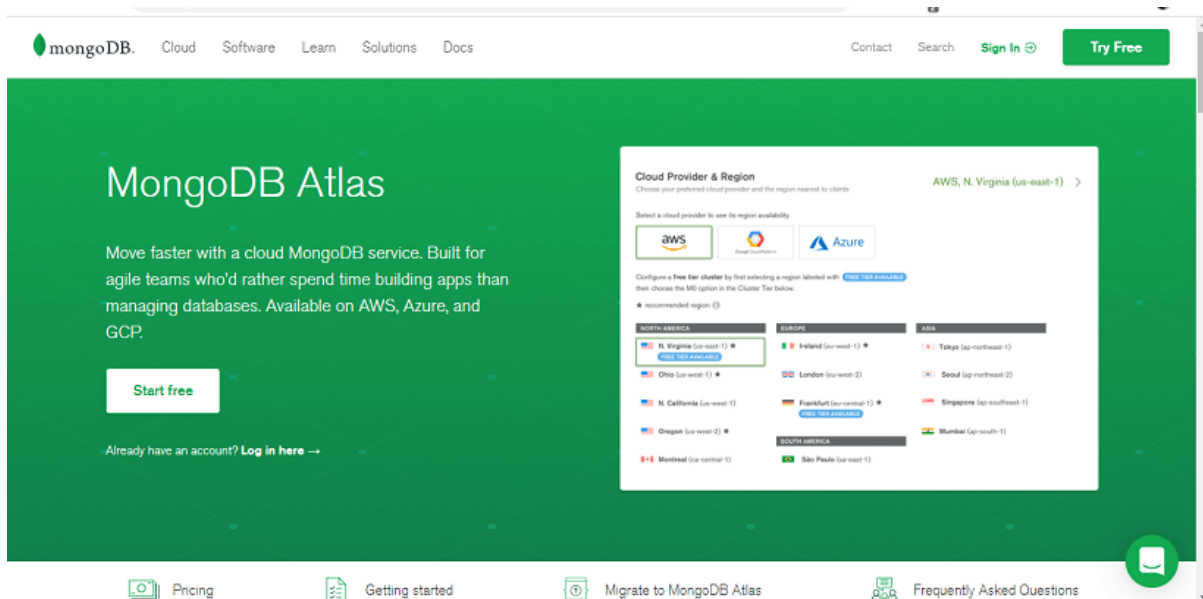
Step 2: When you click on Start Free, you will be redirected to the Registration form for an account on the MongoDB Atlas.
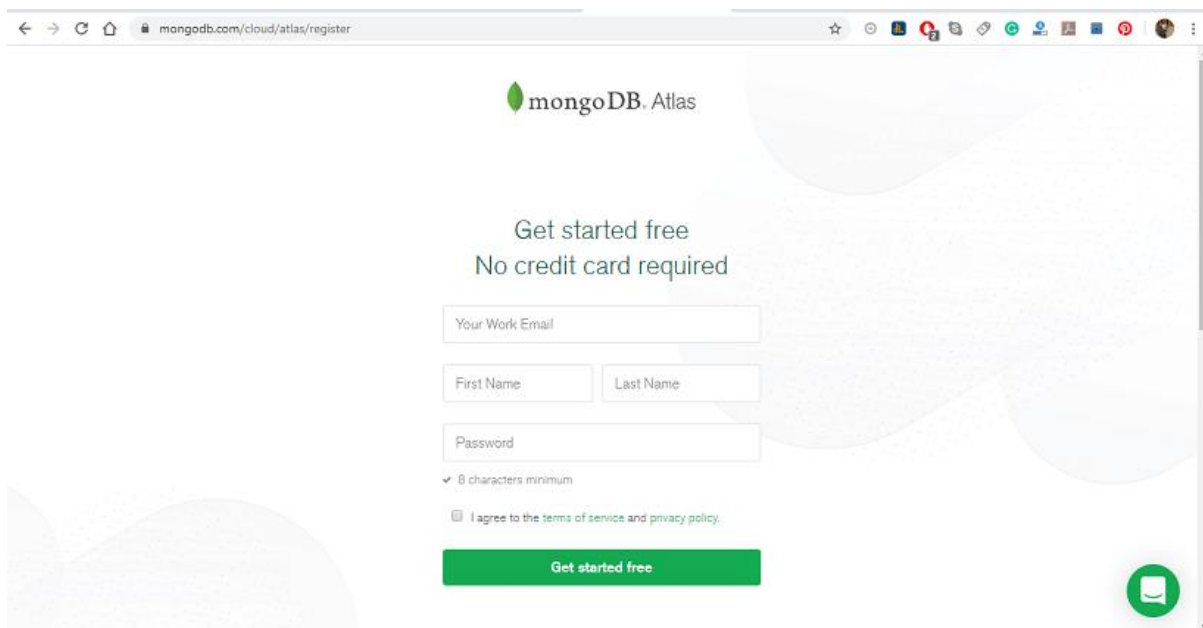

Step 3: Select Starter Clusters and click create a Cluster. The Starter cluster includes the M0, M2, and M5 cluster tiers. These low-cost clusters are suitable for users who are learning MongoDB or developing small proof -of- concept applications.
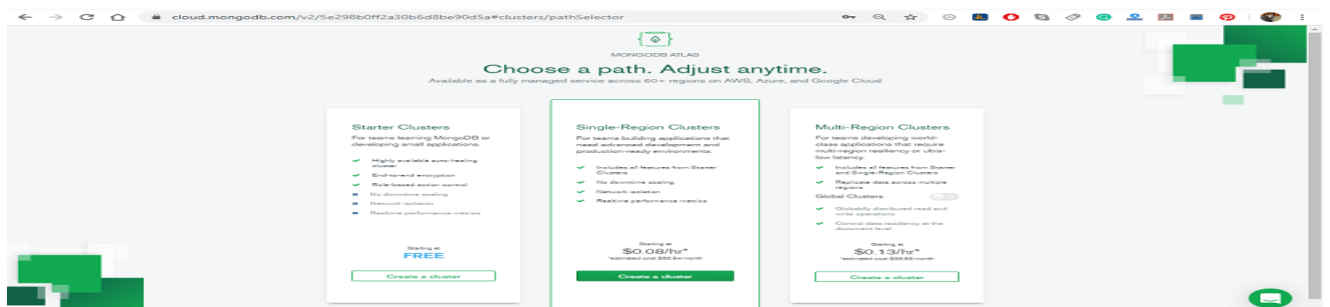
Step 4: Select your preferred Cloud Provider & Region. It supports M0 Free Tier clusters on Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. The regions that support M0 Free tier clusters are marked with the **"*Free Tier Available*"** label.
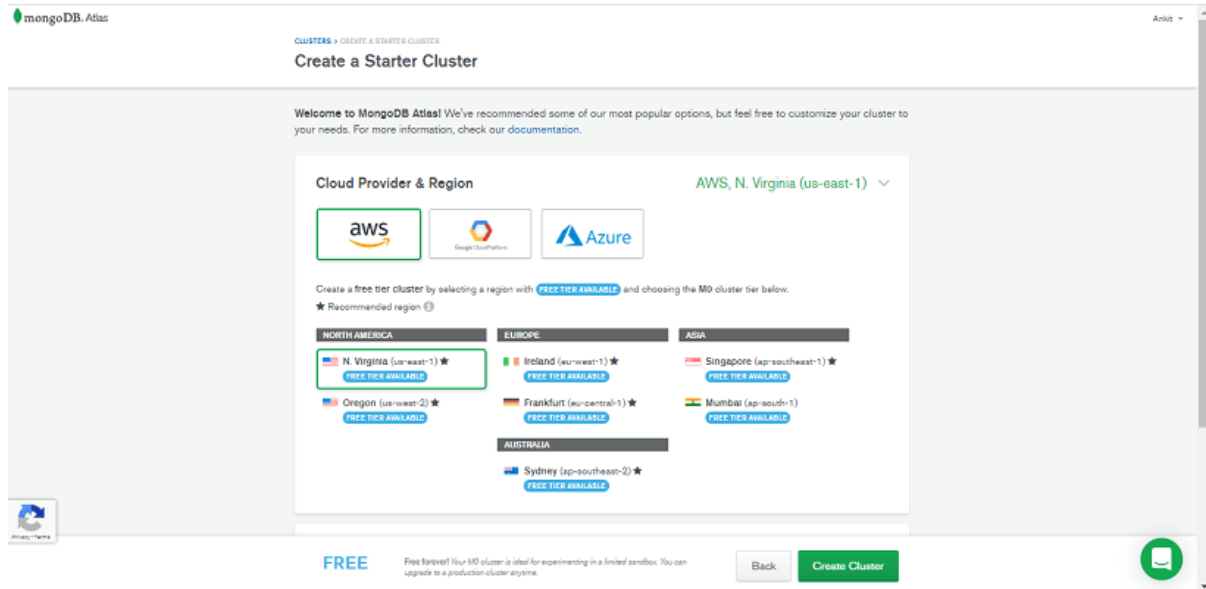
Step 5: Select **M0 Sandbox** for cluster tier: Selecting M0 automatically locks the remaining configuration options. If you can't select the M0 cluster tier, return to the previous step and choose a Cloud Provider & Region that supports M0 Free Tier clusters.
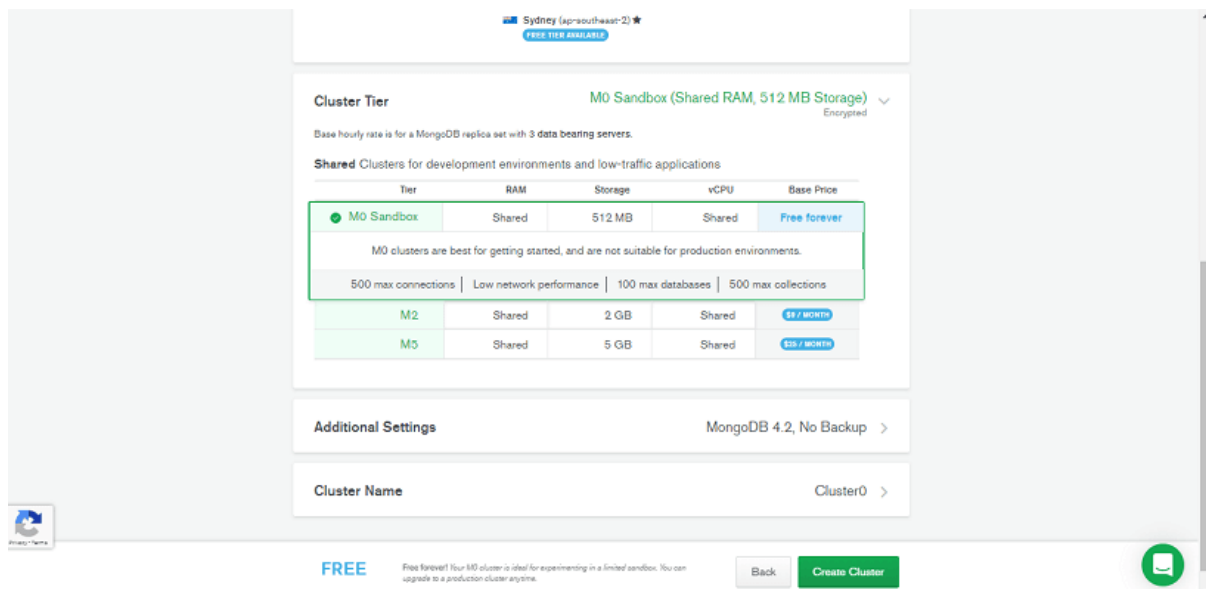
Step 6: Enter a name for your cluster in the Cluster Name field; you can enter any name for your cluster. The cluster name contains ASCII letters, numbers, and hyphens.
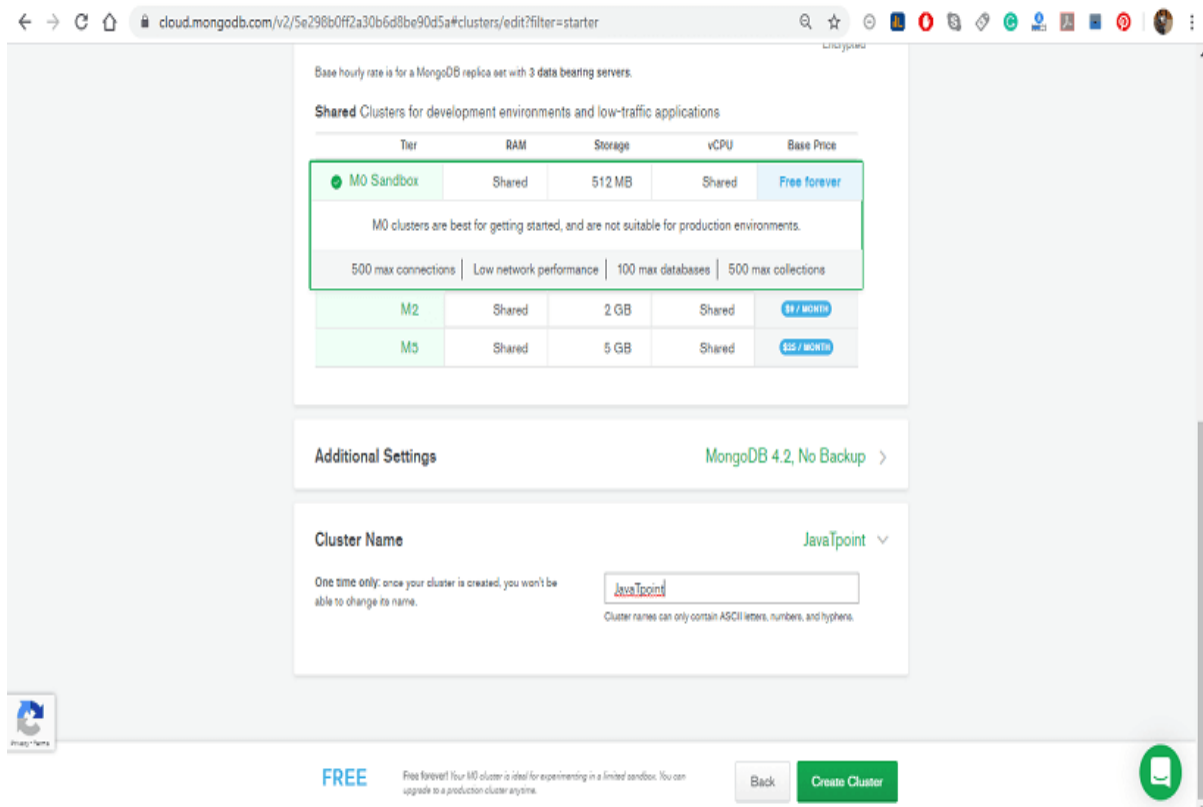
Step 7: Click on Create Cluster to deploy the cluster. Once you deploy your cluster, it can take up to 5-10 min for your cluster to provision and become ready to use.
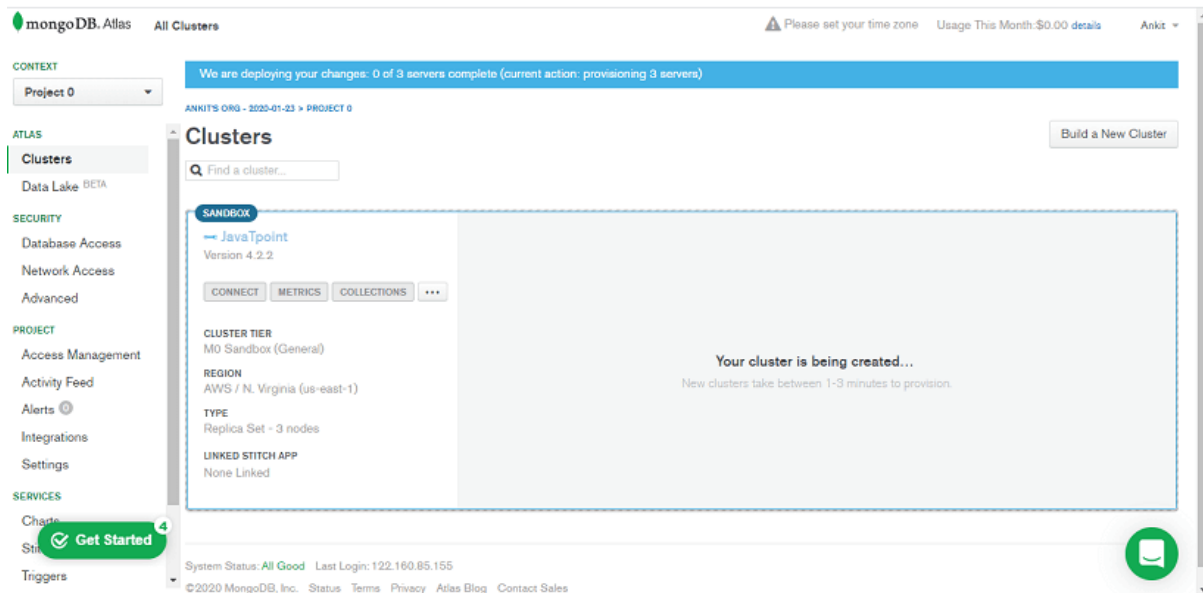
Step 8: Once we register, Atlas automatically creates a default organization and project where we can deploy our first cluster. We can add additional organizations and projects later.

11b) Write MongoDB queries to perform CRUD operations on document using insert(), find(), update(), remove()

1. Inset():In MongoDB, insert operations target a single <u>collection</u>. All write operations in MongoDB are <u>atomic</u> on the level of a single <u>document.</u>

```
db.users.insertOne(          ←——— collection
  {
    name: "sue",             ←——— field: value
    age: 26,                 ←——— field: value      } document
    status: "pending"        ←——— field: value
  }
)
```

2. find():You can specify <u>query filters or criteria</u> that identify the documents to return.

```
db.users.find(           ←—— collection
  { age: { $gt: 18 } },  ←—— query criteria
  { name: 1, address: 1 } ←—— projection
).limit(5)               ←—— cursor modifier
```
<center>click to enlarge</center>

3. Update ():

Update operations modify existing <u>documents</u> in a <u>collection</u>. MongoDB provides the following methods to update documents of a collection:

- <u>db.collection.updateOne()</u> *New in version 3.2*

- <u>db.collection.updateMany()</u> *New in version 3.2*

- <u>db.collection.replaceOne()</u> *New in version 3.2*

In MongoDB, update operations target a single collection. All write operations in MongoDB are <u>atomic</u> on the level of a single document.

You can specify criteria, or filters, that identify the documents to update. These <u>filters</u> use the same syntax as read operations.

```
db.users.updateMany(                    ◄——— collection
    { age: { $lt: 18 } },               ◄——— update filter
    { $set: { status: "reject" } } }    ◄——— update action
)
```

4. Remove()

```
db.collection.remove()
```

Removes documents from a collection.

Returns: A <u>WriteResult</u> object that contains the status of the operation.

```
db.collection.remove(
   <query>,
   {
    justOne: <boolean>,
    writeConcern: <document>,
    collation: <document>,
    let: <document> // Added in MongoDB 5.0
   }
)
```

12a) Write MongoDB queries to Create and drop databases and collections.

Create database :

Creating a Database

In MongoDB, we can create a database using the use command. As shown in the below image

```
> use gfgDB
switched to db gfgDB
>
```

Drop Database :

Steps to Drop Database in MongoDB

In order to drop a database or delete a database in MongoDB, the following steps are followed:

Step 1

List out all the available databases by using the show dbs command.

Syntax

show dbs

Step 2

Select and connect to the database which is to be deleted using the use command in the MongoDB shell.

Syntax

use <database_name>

Step 3

Drop the connected database using the dropDatabase() function.

Syntax

db.dropDatabase()

Step 4

Check the list of databases again now using the show command to confirm the deletion of the database.

Syntax

show dbs

Let us now illustrate this process of dropping a MongoDB database through the implementation of an example.

12b) Write MongoDB queries to work with records using find(), limit(), sort(), createIndex(), aggregate().

Limit()

In MongoDB, the **limit()** method limits the number of records or documents that you want. It basically defines the max limit of records/documents that you want. Or in other words, this method uses on cursor to specify the maximum number of documents/ records the cursor will return. We can use this method after the find() method and find() will give you all the records or documents in the collection. You can also use some conditions inside the find to give you the result that you want.

Example file :

```
> db.gfg.find()
{ "_id" : ObjectId("6005d3158438681f01c53e7f"), "content" : "Data Structure" }
{ "_id" : ObjectId("6005d3258438681f01c53e80"), "content" : "Algorithims" }
{ "_id" : ObjectId("6005d3318438681f01c53e81"), "content" : "Interview Preparation" }
{ "_id" : ObjectId("6005d33d8438681f01c53e82"), "content" : "FANG" }
{ "_id" : ObjectId("6009c642df8008388bd7646a"), "content" : "Competitve Programming" }
{ "_id" : ObjectId("6009c66bdf8008388bd7646b"), "content" : "Development" }
{ "_id" : ObjectId("6009c8e4df8008388bd7646c"), "content" : "coding questions" }
{ "_id" : ObjectId("6009c907df8008388bd7646d"), "content" : "compiler online" }
>
```

Example:
```
db.gfg.find().limit(2)
```

out put of above command

```
> db.gfg.find().limit(2)
{ "_id" : ObjectId("6005d3158438681f01c53e7f"), "content" : "Data Structure" }
{ "_id" : ObjectId("6005d3258438681f01c53e80"), "content" : "Algorithims" }
>
```

Sort()

The **sort()** method specifies the order in which the query returns the matching documents from the given collection. You must apply this method to the cursor before retrieving any documents from the database. It takes a document as a parameter that contains a field: value pair that defines the sort order of the result set. The value is 1 or -1 specifying an ascending or descending sort respectively.

Example :

```
J
> db.student.find().pretty()
{
        "_id" : ObjectId("600f1abb923681e7681ebdce"),
        "name" : "Akshay",
        "age" : 19
}
{
        "_id" : ObjectId("600f1abb923681e7681ebdcf"),
        "name" : "Bablu",
        "age" : 18
}
{
        "_id" : ObjectId("600f1abb923681e7681ebdd0"),
        "name" : "Rakesh",
        "age" : 21
}
{
        "_id" : ObjectId("600f1abb923681e7681ebdd1"),
        "name" : "Gourav",
        "age" : 20
}
> _
```

Now write a command sort the records based on age

```
> db.student.find().sort({age:1})
```

```
.
> db.student.find().sort({age:1})
{ "_id" : ObjectId("6015ba124dabc381f81e53ae"), "name" : "Bablue", "age" : 18 }
{ "_id" : ObjectId("6015ba124dabc381f81e53ad"), "name" : "Akshay", "age" : 19 }
{ "_id" : ObjectId("6015ba124dabc381f81e53b0"), "name" : "Gourav", "age" : 20 }
{ "_id" : ObjectId("6015ba124dabc381f81e53af"), "name" : "Rakesh", "age" : 21 }
>
```

createIndex()

You can use db.collection.createIndex() for deployments hosted in the following

environments:

- MongoDB Atlas: The fully managed service for MongoDB deployments in the cloud

- MongoDB Enterprise: The subscription-based, self-managed version of MongoDB

- MongoDB Community: The source-available, free-to-use, and self-managed version
  of MongoDB

The createIndex() method has the following form:

```
db.collection.createIndex( <keys>, <options>, <commitQuorum>)
```

The createIndex() method takes the following parameters:

aggregate ()

```
db.train.aggregate([{$group : { _id :"$id", total : { $sum : "$fare" }}}])
                        Stage      Expression  Accumulator
```

Example:

```
> db.students.aggregate([{$group: {_id:"$sec", total_st: {$sum:1}, max_age:{$max:"$age"} } }])
{ "_id" : "A", "total_st" : 4, "max_age" : 37 }
{ "_id" : "B", "total_st" : 3, "max_age" : 40 }
>
```