1.a Course Name: Angular JS Module Name: Angular Application Setup
Observe the link http://localhost:4200/welcome on which the mCart application is running. Perform the below activities to understand the features of the application.

Steps to install Angular CLI

Angular CLI can be installed using Node package manager using the command shown below

D:\> npm install -g @angular/cli

Create an application with the name 'MyApp' using the following CLI command

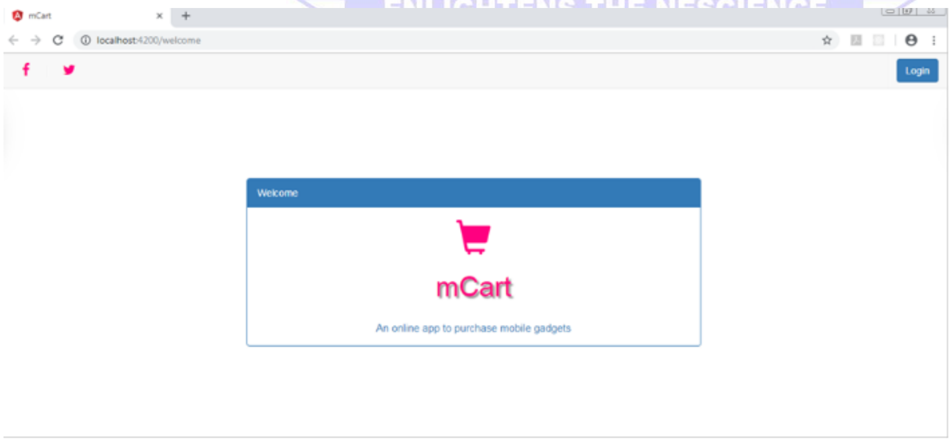
D:\>ng new MyApp

open the Node command prompt and navigate to the mCart folder as shown below, and run the 'npm install' command to install the npm packages.

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

Type the following command to run the application. This will open a browser with the default port as 4200.

D:\mCart>ng serve –open



1.b Course Name: Angular JS Module Name: Components and Modules Create a new component called hello and render Hello Angular on the page.

hello.component.ts

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

@Component({

  selector: 'app-hello',

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

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

})

export class HelloComponent implements OnInit {

  courseName: string = "Angular";

  constructor() { }

  ngOnInit() {

  }

}
```

hello.component.html

```
<p>

  Hello {{ courseName }}

</p>
```

hello.component.css

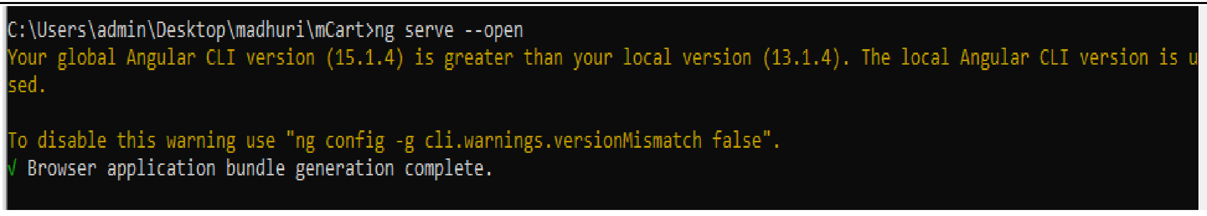
```
p {

    color:blue;

    font-size:20px;

}
```

app.module.ts

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

import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';

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

import { HelloComponent } from './hello/hello.component';

@NgModule({

  imports: [BrowserModule,AppRoutingModule],

  declarations: [AppComponent, HelloComponent],

  providers: [],

  bootstrap: [HelloComponent]

})

export class AppModule { }
```

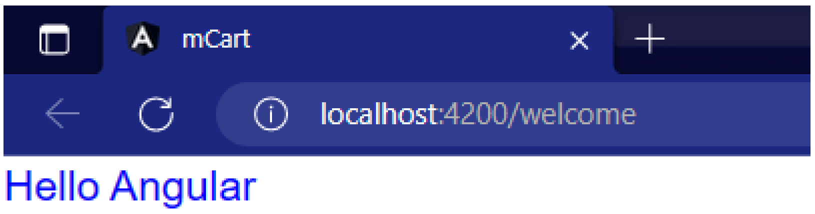
index.html

```
<!doctype html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>MyApp</title>

  <base href="/">

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

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

</head>

<body>

  <app-hello></app-hello>

</body>

</html>
```



```
C:\Users\admin\Desktop\madhuri\mCart>ng serve --open
Your global Angular CLI version (15.1.4) is greater than your local version (13.1.4). The local Angular CLI version is u
sed.

To disable this warning use "ng config -g cli.warnings.versionMismatch false".
√ Browser application bundle generation complete.
```

OUTPUT:



Hello Angular

1.c Course Name: Angular JS Module Name: Elements of Template Add an event to the hello component template and when it is clicked, it should change the courseName.

hello.component.ts

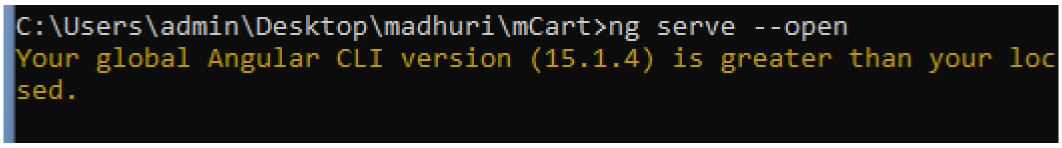
```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-hello',
  templateUrl: "./hello.component.html",
  styleUrls: ['./hello.component.css']
})
export class HelloComponent implements OnInit {
  courseName = "Angular";
  constructor() { }
  ngOnInit() {
  }
  changeName() {
    this.courseName = "TypeScript";
  }
}
```

hello.component.html

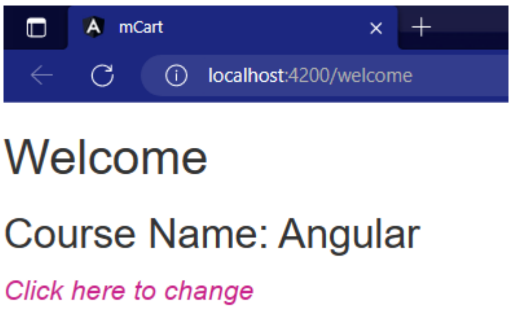
```
<h1>Welcome</h1>
<h2>Course Name: {{ courseName }}</h2>
<p (click)="changeName()">Click here to change</p>
```

hello.component.css

```
p {
    color:rgb(202, 30, 139);
    font-size:20px;
    font-style:italic;

}
```



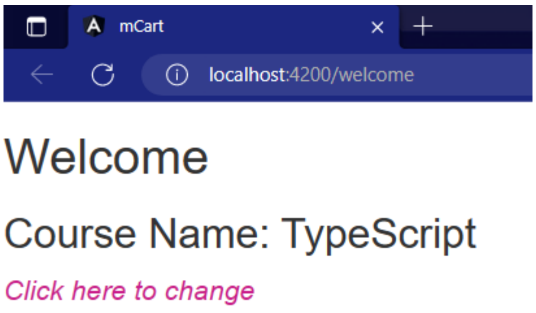
```
C:\Users\admin\Desktop\madhuri\mCart>ng serve --open
Your global Angular CLI version (15.1.4) is greater than your loc
sed.
```



Welcome

Course Name: Angular

Click here to change



Welcome

Course Name: TypeScript

Click here to change

2.a Course Name: Angular JS Module Name: Structural Directives - ngIf 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

app.component.ts

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

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css'],

})

export class AppComponent {

  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;

    }

  }

}
```

app.component.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>
```

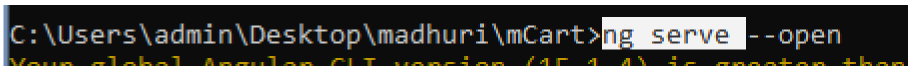
app.module.ts

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

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

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

@NgModule({

  declarations: [

    AppComponent

  ],

  imports: [

    BrowserModule
  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }
```

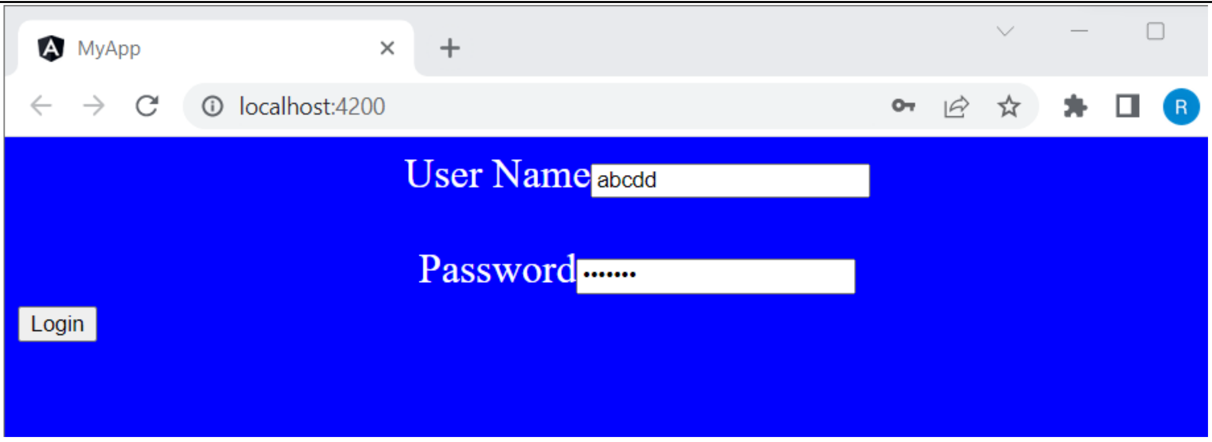
index.html

```
<!doctype html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>MyApp</title>

  <base href="/">

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

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

</head>

<body>

  <app-root></app-root>

</body>

</html>
```

OUTPUT:

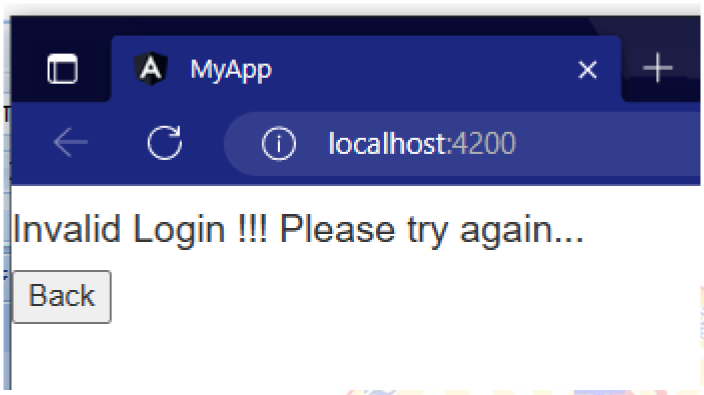


```
C:\Users\admin\Desktop\madhuri\mCart>ng serve --open
Your global Angular CLI version (15.1.4) is greater than
```



User Name abcdd

Password •••••••



Invalid Login !!! Please try again...

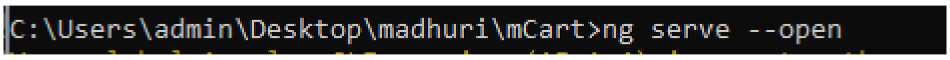
2.b Course Name: Angular JS Module Name: ngFor Create a courses array and rendering it in the template using ngFor directive in a list format.

app.component.ts

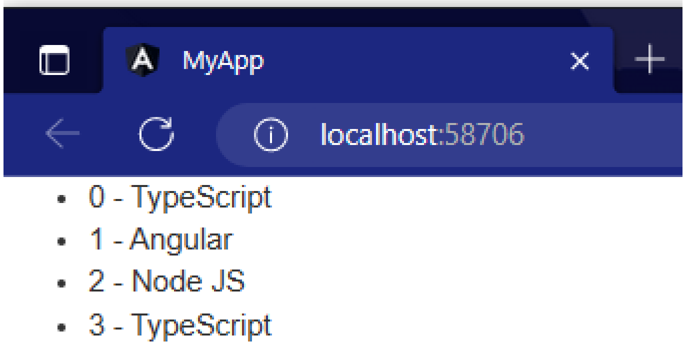
```
import { Component } from '@angular/core';

@Component({

  selector: 'app-root',

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

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

})

export class AppComponent {

  courses: any[] = [

    { id: 1, name: 'TypeScript' },

    { id: 2, name: 'Angular' },

    { id: 3, name: 'Node JS' },

    { id: 1, name: 'TypeScript' }

  ];

}
```

app.component.html

```
<ul>

  <li *ngFor="let course of courses; let i = index">

    {{ i }} - {{ course.name }}

  </li>

</ul>
```



```
C:\Users\admin\Desktop\madhuri\mCart>ng serve --open
```



```
• 0 - TypeScript
• 1 - Angular
• 2 - Node JS
• 3 - TypeScript
```

2.c Course Name: Angular JS Module Name: ngSwitch Display the correct option based on the value passed to ngSwitch directive.

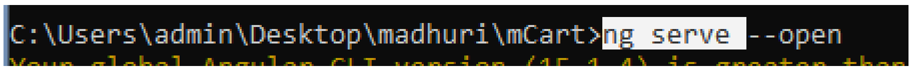
app.component.ts

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

@Component({

  selector: 'app-root',

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

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

})

export class AppComponent {

  choice = 0;

  nextChoice() {

    this.choice++;

  }

}
```

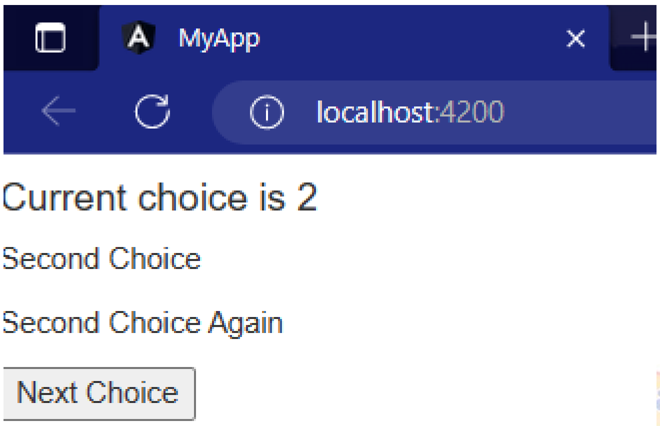
app.component.html

```
<h4>

  Current choice is {{ choice }}

</h4>

<div [ngSwitch]="choice">

  <p *ngSwitchCase="1">First Choice</p>

  <p *ngSwitchCase="2">Second Choice</p>

  <p *ngSwitchCase="3">Third Choice</p>

  <p *ngSwitchCase="2">Second Choice Again</p>

  <p *ngSwitchDefault>Default Choice</p>

</div>

<div>
```

```
  <button (click)="nextChoice()">

                Next Choice

        </button>

</div>
```



```
C:\Users\admin\Desktop\madhuri\mCart>ng serve --open
Your global Angular CLI version (15.1.4) is greater than
```



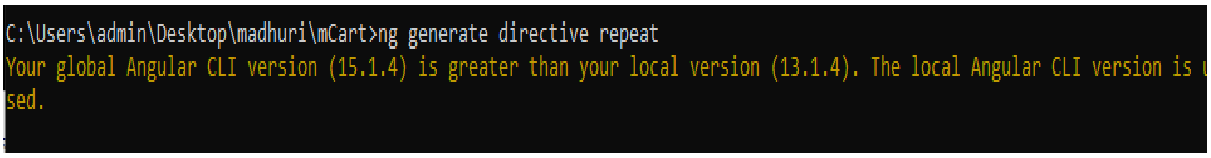
Current choice is 2

Second Choice

Second Choice Again

2.d Course Name: Angular JS Module Name: Custom Structural Directive
Create a custom structural directive called 'repeat' which should repeat the element given a number of times.

Generate a directive called 'repeat' using the following command



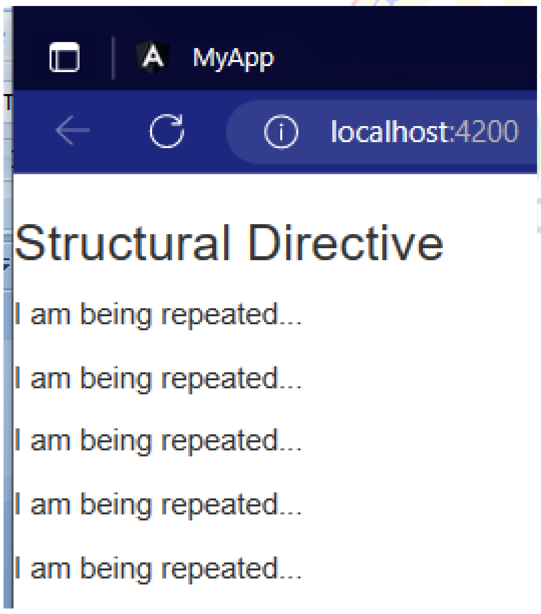
```
C:\Users\admin\Desktop\madhuri\mCart>ng generate directive repeat
Your global Angular CLI version (15.1.4) is greater than your local version (13.1.4). The local Angular CLI version is u
sed.
```

app.module.ts

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

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

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

import { RepeatDirective } from './repeat.directive';

@NgModule({

  declarations: [

    AppComponent,

    RepeatDirective

  ],

  imports: [

    BrowserModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }
```

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);

    }

  }

}

<h3>Structural Directive</h3>

<p *appRepeat="5">I am being repeated...</p>
```



Structural Directive

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

3.a Course Name: Angular JS Module Name: Attribute Directives - ngStyle Apply multiple CSS properties to a paragraph in a component using ngStyle.

Description:
Attribute directives change the appearance/behavior of a component/element.
Following are built-in attribute directives:

ngStyle
ngClass

ngStyle:
This directive is used to modify a component/element's style. You can use the following syntax to set a single CSS style to the element which is also known as style binding.

Program:

App.component.html

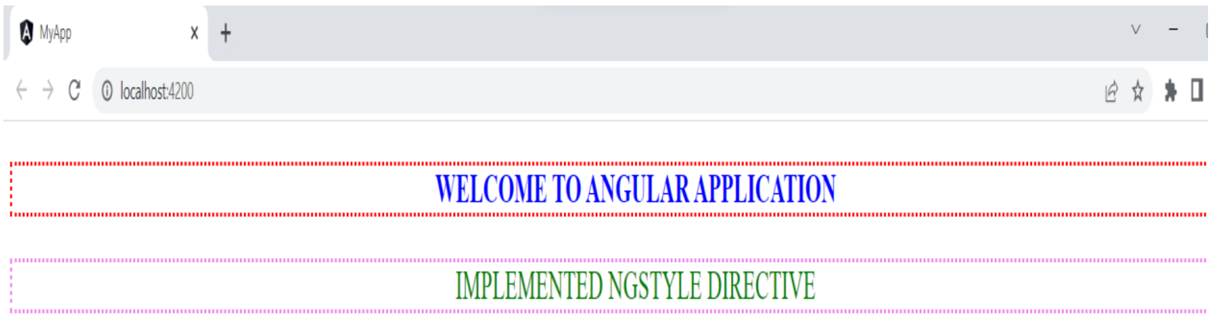
```
<p [ngStyle]="{

  color:colorName1,

  'font-weight':fontWeight,

  border: borderStyle,

  'text-align':align,

  'font-size':fontsize

}">

WELCOME TO ANGULAR APPLICATION

</p>

<p [ngStyle]="{

color:colorName2,

'text-align':align,

'font-size':fontsize,

border:borderStyle2

}">IMPLEMENTED NGSTYLE DIRECTIVE</p>
```

App.component.ts

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

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


export class AppComponent {
  colorName1 = 'blue';
  colorName2='green';
  fontWeight = 'bold';
  borderStyle = '3px dotted red';
  align='center';
  fontsize='25px';
  borderStyle2='3px dotted violet'
}
```

Output:



WELCOME TO ANGULAR APPLICATION

IMPLEMENTED NGSTYLE DIRECTIVE

3.b Course Name: Angular JS Module Name: ngClass Apply multiple CSS classes to the text using ngClass directive.

Description:
Attribute directives change the appearance/behavior of a component/element.
Following are built-in attribute directives:
- ngStyle
- ngClass

ngClass:
ngClass is a directive in Angular that adds and removes CSS classes on an HTML element. In this article, we are talking about ngClass in Angular only, not ng-class in angular.js.

Program:

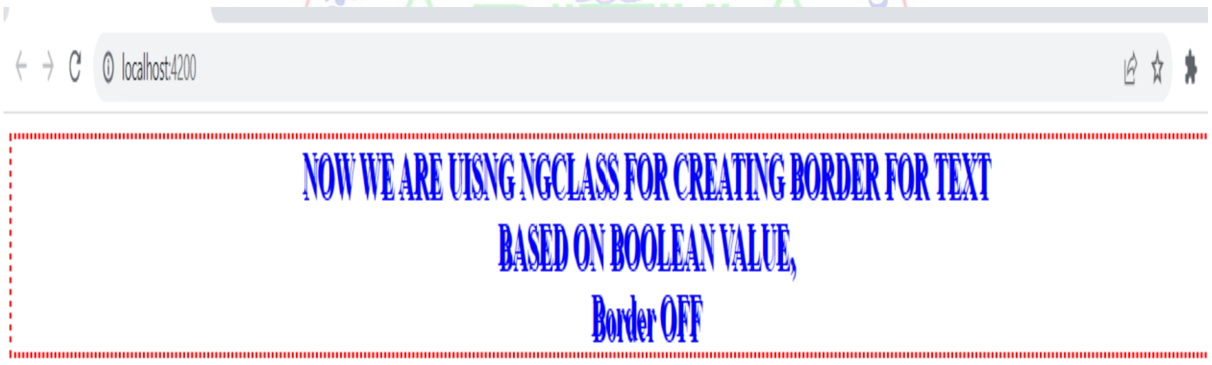
App.component.html

```
<div [ngClass]="{bordered: isBordered}" [ngStyle]="{

  color:colorName1,

  'font-weight':fontWeight,

  border: borderStyle,

  'text-align':align,

  'font-size':fontsize,

  'text-shadow':shadow

}">

NOW WE ARE UISNG NGCLASS FOR CREATING BORDER FOR TEXT<br>

BASED ON BOOLEAN VALUE,<BR>

  Border {{ isBordered ? "ON" : "OFF" }}

</div>
```

App.component.ts

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

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  colorName1 = 'blue';
  colorName2='green';
  fontWeight = 'bold';
  borderStyle = '3px dotted red';
  align='center';
  fontsize='25px';
  borderStyle2='3px dotted violet'
  isBordered=false;
  shadow='2px 2px blue';
}
```

Output:



NOW WE ARE UISNG NGCLASS FOR CREATING BORDER FOR TEXT
BASED ON BOOLEAN VALUE,
Border OFF

Experiment-3c:Course Name: Angular JS **Module Name:** Custom Attribute Directive
Create an attribute directive called 'showMessage' which should display the given message in a paragraph when a user clicks on it and should change the text color to red.

Description:

We can create custom attribute directives and custom structural directives using a @Directive decorator. Using custom attribute directive we can change appearances such as text color, background color and font size of a body of an HTML element that can be called the host element

Program:

App.component.html

```
<h3>Attribute Directive</h3>

<p [appMessage]="myMessage">Click Here TO GET THE TEXT</p>
```

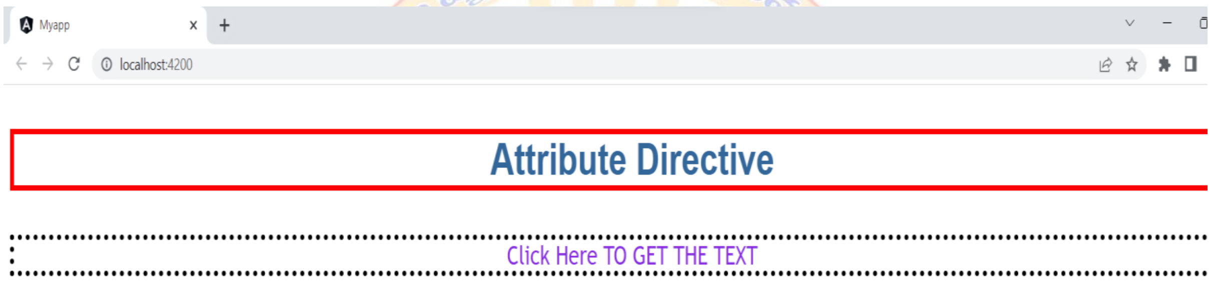
App.component.css

```
h3 {

    color: #369;

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

    font-size: 250%;

    text-align: center;

    border:5px solid red

  }

  p {

    color: blueviolet;

    font-family:Trebuchet MS, Lucida Sans Unicode, Lucida Grande, Lucida
Sans, Arial, sans-serif;

    font-size: 150%;

    text-align: center;

    border: 5px dotted black;

  }
```

App.component.ts

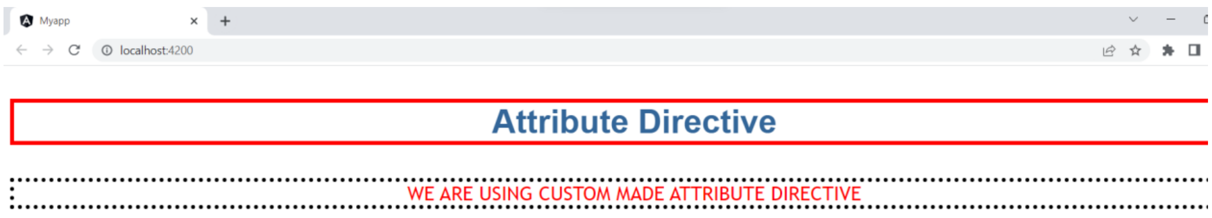
```
import { Component } from '@angular/core';


@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'myapp';
  myMessage = 'WE ARE USING CUSTOM MADE ATTRIBUTE DIRECTIVE';
}
```

Output:



Attribute Directive

Click Here TO GET THE TEXT



Attribute Directive

WE ARE USING CUSTOM MADE ATTRIBUTE DIRECTIVE

4.a Course Name: Angular JS

Module Name: Property Binding

Binding image with class property using property binding.

Description:

Data Binding is a mechanism where data in view and model are in sync. Users should be able to see the same data in a view which the model contains.

Property binding in Angular helps you set values for properties of HTML elements or directives. Use property binding to do things such as toggle button features, set paths programmatically, and share values between components.

Syntax:

[property-target]=”expression”;

Program:

App.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'myapp';
  content:string='using another property binding';
    imgUrl: string = 'assets/rorona.jpg';
  }
```

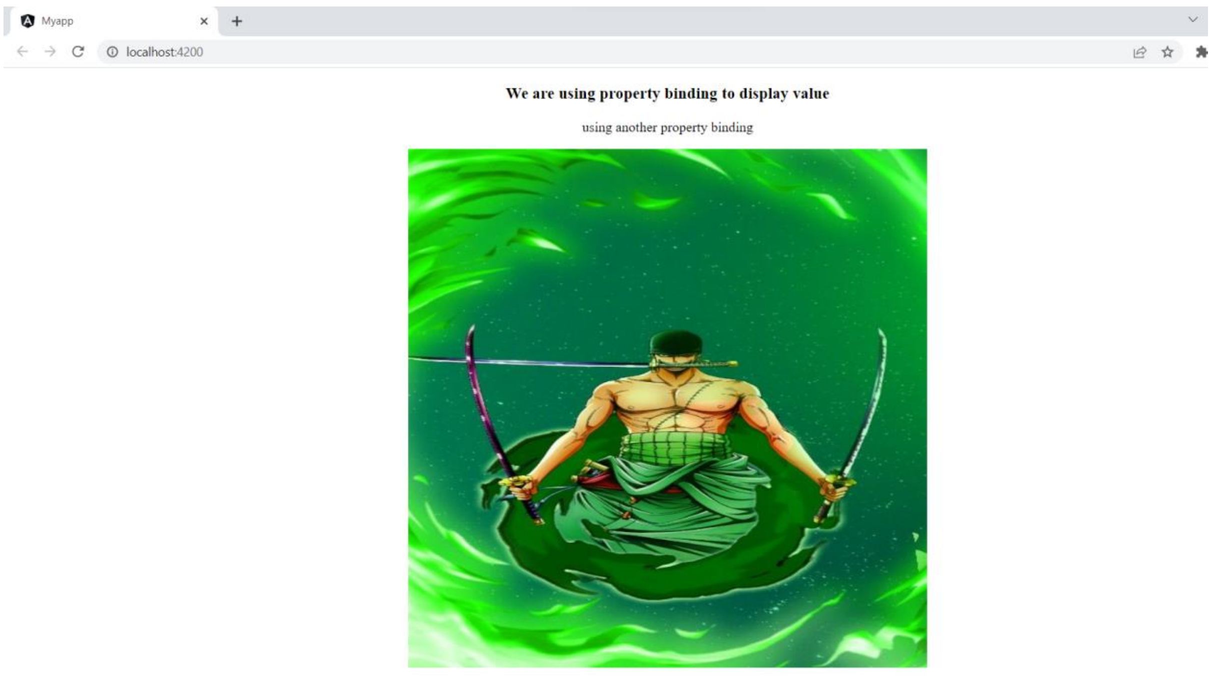
App.component.html

```
<div class="alin">
    <h3>We are using property binding to display value</h3>
    <p>{{content}}</p>
    <img [src]="imgUrl" alt="reload" height="600px" width="600px">
</div>
```

App.component.css

```
.alin{
  text-align: center;
}
```

Output:



We are using property binding to display value

using another property binding

4.b Course Name: Angular JS
Module Name: Attribute Binding
Binding colspan attribute of a table element to the class property

Description:
Data Binding is a mechanism where data in view and model are in sync. Users should be able to see the same data in a view which the model contains.
Attribute binding is used to bind an attribute property of a view element. Attribute binding is mainly useful where we don't have any property view with respect to an HTML element attribute.
Syntax:

[attr.attribute-you-are-targeting]="expression"

Program:
App.component.html

```
<div class="alin">
   <p style="text-align:center">
 <b>  TABLE CONTENT FOR STUDENT ATTENDANCE</b></p>

 <br>

   <table border="1">
    <tr>
        <td>NAME</td>
        <td>ID NUMBER</td>
        <td [attr.colspan]=colspanva>ATTENDANCE
            <tr>
                <td>MORNING</td>
                <td>AFTERNOON</td>
            </tr>
        </td>
    </tr>
    <tr>
        <td>ABC</td>
        <td>1234</td>
        <td>YES</td>
        <td>NO</td>
    </tr>
    <tr>
        <td>DEF</td>
        <td>56A7</td>
        <td>YES</td>
        <td>YES</td>
    </tr>
    <tr>
        <td>GHI</td>
        <td>9012</td>
    <td>NO</td>
    <td>NO</td>
    </tr>
   </table>
</div>
```

App.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'myapp';
colspanva:string="2";
  }
```

App.component.css

```
.alin{
  text-align: center;
}
table{
  margin-left: 600px;
}
```

Output:

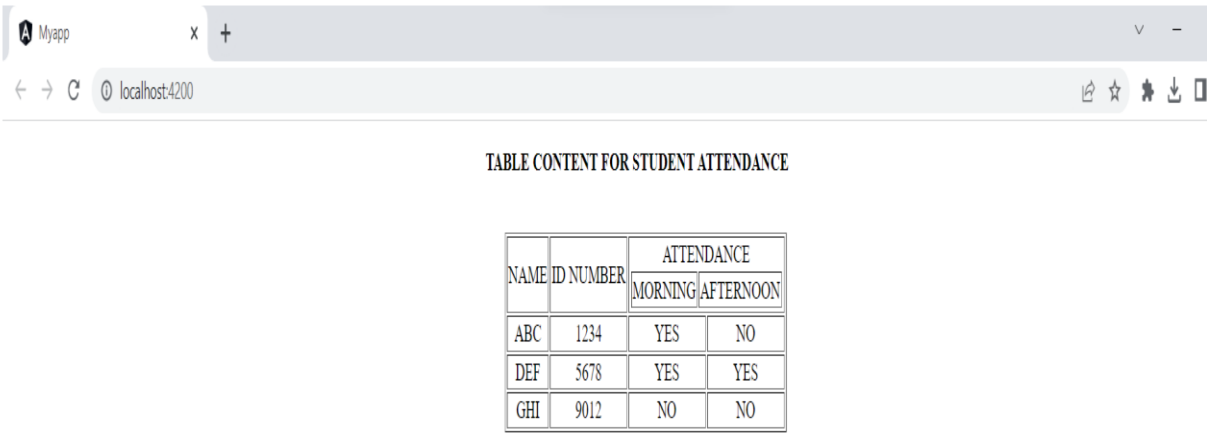


TABLE CONTENT FOR STUDENT ATTENDANCE

NAME	ID NUMBER	ATTENDANCE	
		MORNING	AFTERNOON
ABC	1234	YES	NO
DEF	5678	YES	YES
GHI	9012	NO	NO

4.c Course Name: Angular JS
Module Name: Style and Event Binding
Binding an element using inline style and user actions like entering text in input fields.

Description:

Style binding is used to set inline styles. Syntax starts with prefix style, followed by a dot and the name of a CSS style property.

Syntax:

[style.styleproperty]

User actions such as entering text in input boxes, picking items from lists, button clicks may result in a flow of data in the opposite direction: from an element to the component.

Event binding syntax consists of a target event with ( ) on the left of an equal sign and a template statement on the right.

Syntax:

```
<button (click)="onSave()">Save</button>
```

Program:

App.component.hmtl

```
<div id="id2">

    <div *ngIf = "!submitted" id="id1">

        <div [style.color] = "'blue'"

        [style.font-size.px]="'24'" >Student Login

        </div><br/>

        <form>

        <label>Student Name : </label>

        <input type="text" #studentname /><br/><br/>

        <button (click)="login(studentname.value)"
id="id3">Submit</button>

        </form>
       </div>

       <div *ngIf = "submitted" id="id1">

        <h4>Welcome ,{{ userName }}!,to Mcart</h4>

       </div>

</div>
```

App.component.css

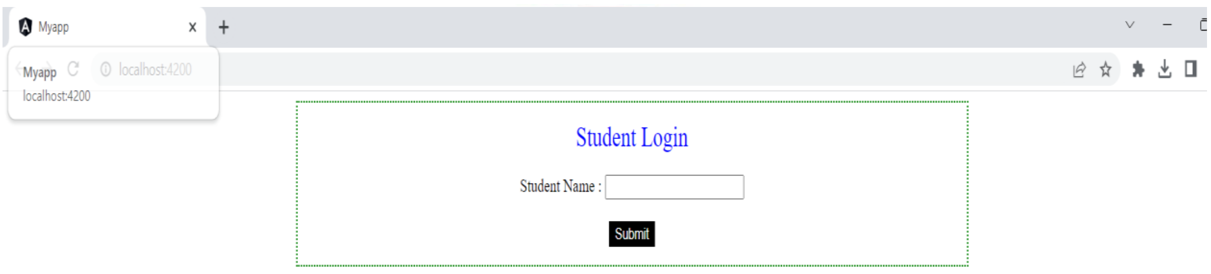
```
#id2{

  text-align: center;

}

#id1{

  border: 2px dotted green;

  margin-left: 350px;

  margin-right: 350px;

  padding: 15px;

}

#id3{

  color:white;

  background-color:black;

}
```

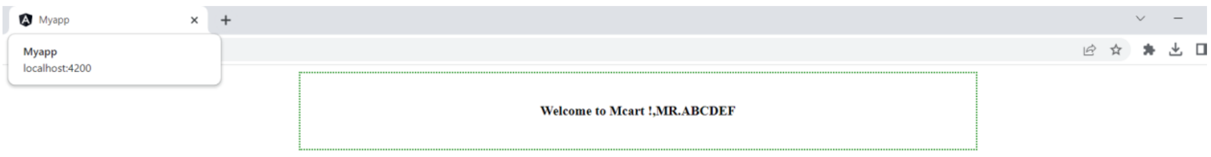
App.component.ts

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


@Component({

  selector: 'app-root',

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

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

})
```

```
export class AppComponent {

  userName: string = '';

  submitted = false;

  login(name: string) {

  this.userName = name;

  this.submitted = true;

  }


  }
```

Output:





Welcome to Mcart !,MR.ABCDEF

5.a Course Name: Angular JS

Module Name: Built in Pipes

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

Description:

Pipes provide a beautiful way of transforming the data inside templates, for the purpose of display.

Pipes in Angular take an expression value as an input and transform it into the desired output before displaying it to the user. It provides a clean and structured code as you can reuse the pipes throughout the application, while declaring each pipe just once.

Syntax:

{{ expression | pipe }}

Program:

App.component.html

```
<div class="alin" *ngIf = "!submitted" id="id1">
    <h2>Enter Product Details</h2>
    <form>
        <label>Product_Id : </label>
        <input type="text" #productid /><br/><br/>
        <label id="pname" >Product_Name:</label>
        <input type="text" #productname>
 <br><br>
<button(click)="login(productid.value,productname.value)"id="id3">Su
bmit</button>
        </form>

</div>
<div *ngIf = "submitted" id="id1">
    <h4> Product_Id IN LOWER_CASE : {{ productId | lowercase}}!</h4>

<br><br>
    <h4>Product_name   PRODUCT_NAME  IN  UPPER_CASE  :  {{productName  |
uppercase}}</h4>
</div>
```

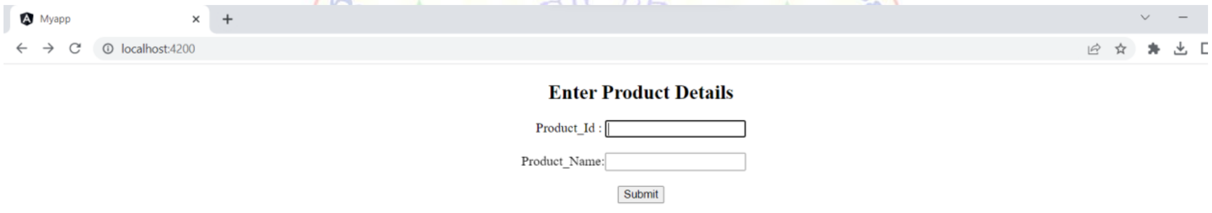
App.component.css

```
.alin{
  text-align: center;
}
#pname{
  margin-left: -18px;
}
```

App.component.ts

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

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

  productName: string = '';
productId:string='';
  submitted = false;
  login(id:string,name: string) {
    this.productId=id;
  this.productName = name;
  this.submitted = true;
  }


  }
```

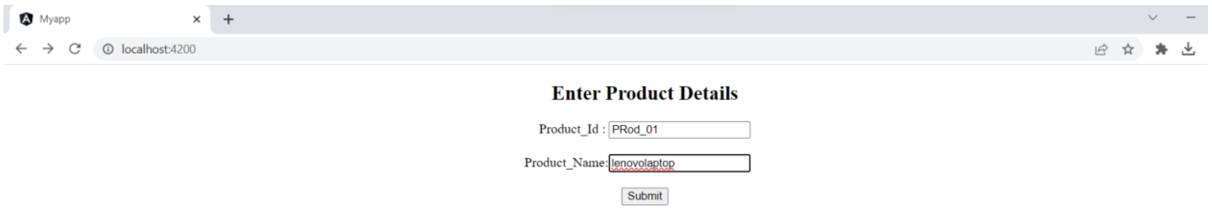
Output:



Enter Product Details

Product_Id :

Product_Name:



Enter Product Details

Product_Id : PRod_01

Product_Name: lenovolaptop

5.b Course Name: Angular JS
Module Name: Passing Parameters to Pipes
Apply built-in pipes with parameters to display product details.
Description:
A pipe can also have optional parameters to change the output. To pass parameters, after a pipe name add a colon(:) followed by the parameter value.
Syntax:

pipename : parametervalue

A pipe can also have multiple parameters as shown below.
Syntax:

pipename : parametervalue1:parametervalue2

Program:
App.component.html

```
<div class="alin">
    <h3 style="color:greenyellow"> {{ title | titlecase}} </h3>
   <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':'':'fr' }}
</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>
   </div>
```

App.component.css

```
.alin{
  text-align: center;
}
table{
  color:white;
  border:1px dotted yellow;
  margin-left:550px;
  margin-right: 550px;
  padding:100px;

}
h3{
  border:1px dotted red;
  margin-left: 500px;
  margin-right: 500px;
}
```

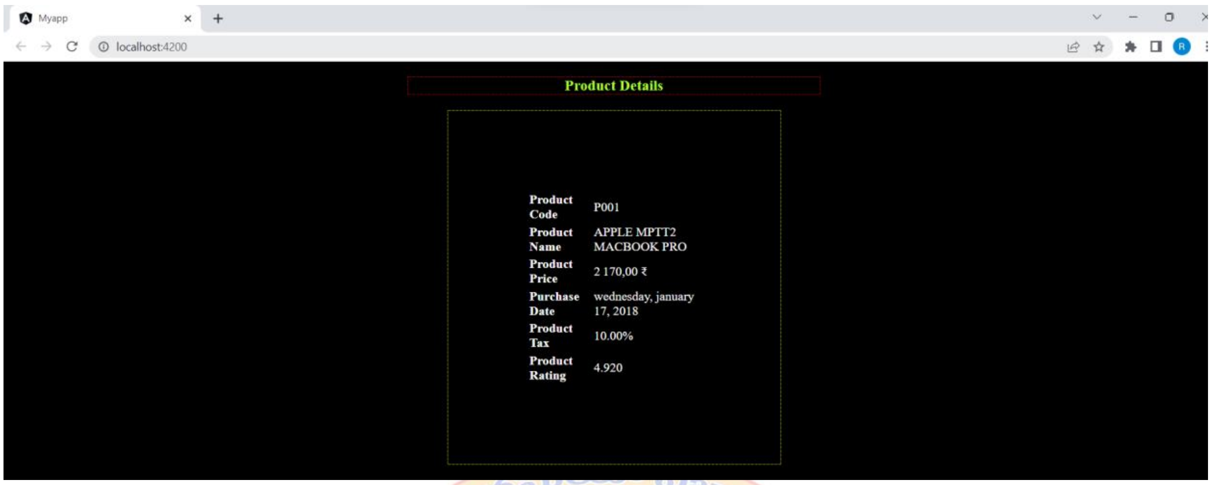
App.component.ts

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

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

  title = 'product details';
  productCode = 'PROD_P001';
  productName = 'Apple MPTT2 MacBook Pro';
  productPrice = 2170;
  purchaseDate = '1/17/2018';
  productTax = '0.1';
  productRating = 4.92;



  }
```

Style.css

```
body{
    background-color: black;
}
```

Output:



Product Details

Product Code	P001
Product Name	APPLE MPTT2 MACBOOK PRO
Product Price	2 170,00 ₹
Purchase Date	wednesday, january 17, 2018
Product Tax	10.00%
Product Rating	4.920

5.c Course Name: Angular JS
Module Name: Nested Components Basics Load CourseslistComponent in the root component when a user clicks on the View courses list button.
Description:

Nested component is a component that is loaded into another component

The component where another component is loaded onto is called a container component/parent component.

The root component is loaded in the index.html page using its selector name. Similarly, to load one component into a parent component, use the selector name of the component in the template i.e., the HTML page of the container component**.**

Program:
App.component.html

```
<h2>Popular Courses</h2>
<button (click)="show = true">View Courses list</button><br /><br />
<div *ngIf="show">
  <app-courses-list></app-courses-list>
</div>
```

App.component.css

App.component.ts

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

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


  show!: boolean;

  }
```

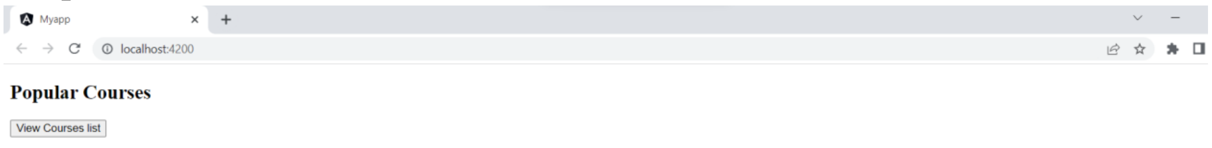
Course-list.component.html

```
<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>
```

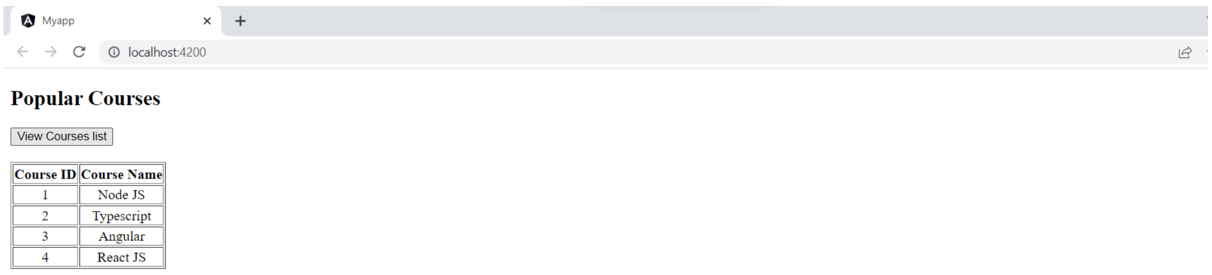
Course-list.component.ts

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-courses-list',
  templateUrl: './courses-list.component.html',
  styleUrls: ['./courses-list.component.css']
})
export class CoursesListComponent {
  courses = [
    { courseId: 1, courseName: "Node JS" },
    { courseId: 2, courseName: "Typescript" },
    { courseId: 3, courseName: "Angular" },
    { courseId: 4, courseName: "React JS" }
  ];
}
```

Output:



Popular Courses



Popular Courses

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

6.a Course Name: Angular JS Module Name: Passing data from Container Component to Child Component 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 from the dropdown, corresponding course details should be loaded.

Program:

App.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  templateUrl: './app.component.html'
})
export class AppComponent {
  name!: string;
}
```

App.component.html

```

<h2>Course Details</h2>
Select a course to view
<select #course (change)="name = course.value">
  <option value="Node JS">Node JS</option>
  <option value="Typescript">Typescript</option>
  <option value="Angular">Angular</option>
  <option value="React JS">React JS</option></select><br /><br />
<app-courses-list [cName]="name"></app-courses-list>
```

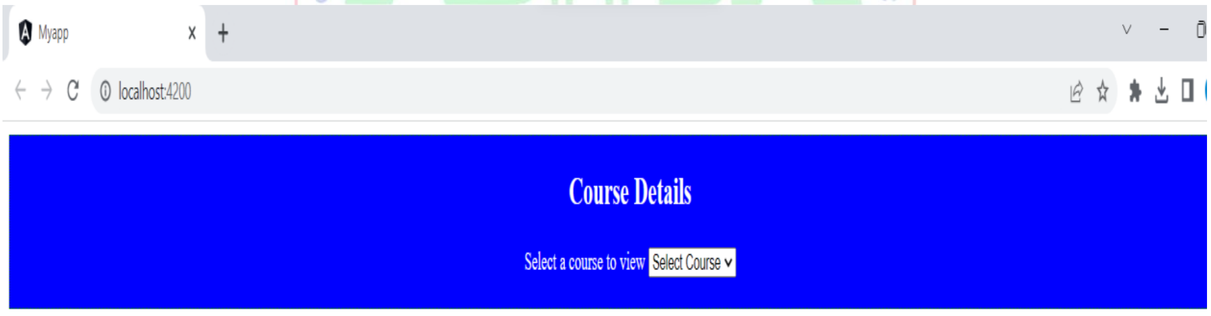
Course-list.component.html

```
  <table border="1" *ngIf="course.length > 0" style="align-items:
center;">
    <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>
```

Course-list.component.ts

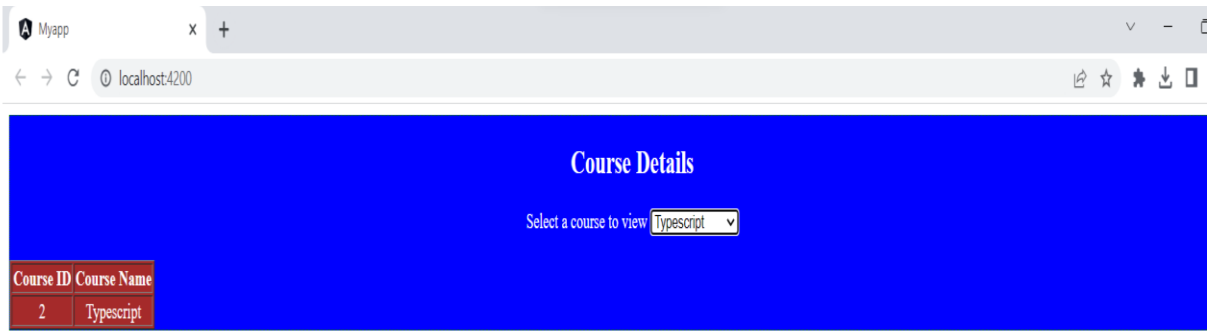
```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'app-courses-list',
  templateUrl: './courses-list.component.html',
  styleUrls: ['./courses-list.component.css'],
  exportAs: 'courselist'
})
export class CoursesListComponent {
  courses = [
    { courseId: 1, courseName: 'Node JS' },
    { courseId: 2, courseName: 'Typescript' },
    { courseId: 3, courseName: 'Angular' },
    { courseId: 4, courseName: 'React JS' }
  ];
  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]);
      }
    }
  }
}
```

Output:



Course Details

Select a course to view Select Course



Course Details

Select a course to view Typescript

Course ID	Course Name
2	Typescript

6.b Course Name: Angular JS Module Name: Passing data from Child Component to ContainerComponent Create an AppComponent that loads another component called the CoursesList component. Create another component called CoursesListComponent which should display the courses list in a table along with a register .button in each row. When a user clicks on the register button, it should send that courseName value back to AppComponent where it should display the registration successful message along with courseName

Program:

Course-list.component.html

```
<table border="1">

  <thead>

    <tr>

      <th>Course ID</th>

      <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>
```

Course-list.component.ts

```
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);

  }

}
```

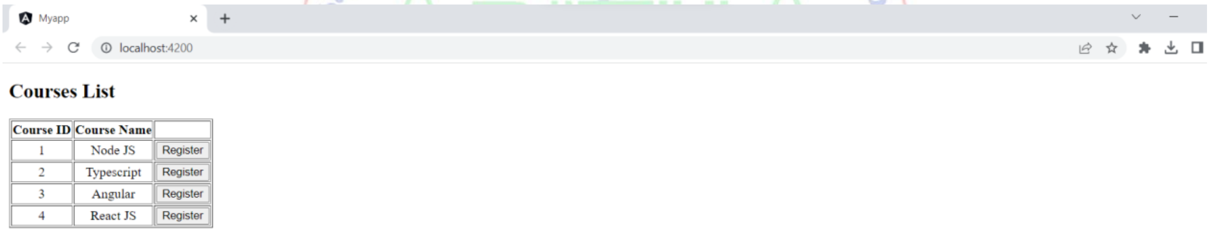
App.component.html

```
<h2>Courses List</h2>

<app-courses-list (registerEvent)="courseReg($event)"></app-courses-
list>

<br /><br />

<div *ngIf="message">{{ message }}</div>
```

App.component.ts

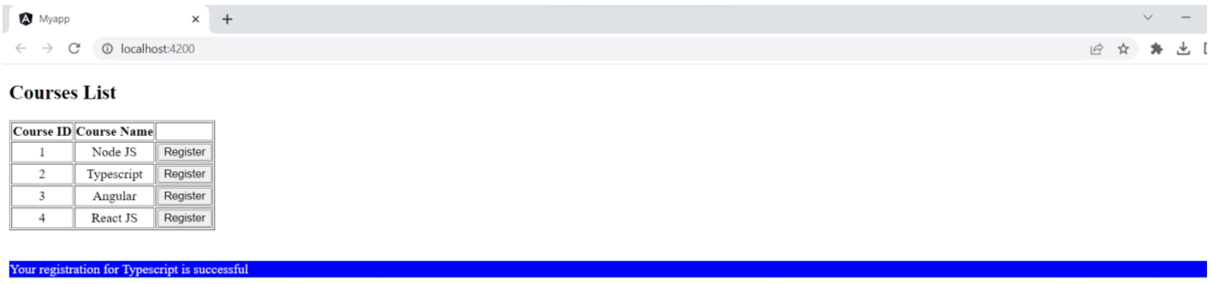
```
import { Component } from '@angular/core';

@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:



Courses List

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



Courses List

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

Your registration for Typescript is successful

6.c Course Name: Angular JS Module Name: Shadow DOM Apply ShadowDOM and None encapsulation modes to components.

Program:

SHADOWDOM::

App.component.html

```
<h3>CSS Encapsulation with Angular</h3>

<div class="cmp">

    App Component

    <app-first></app-first>

    <app-second></app-second>

</div>
```

App.component.ts

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

@Component({

  selector: 'app-root',

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

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

})

export class AppComponent {

}
```

App.component.css

```
.cmp {

          padding: 8px;

          margin: 6px;

          border: 2px solid red;

  }
```

App.module.ts

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

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


import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import {FirstComponent} from './first/first.component';
import {SecondComponent} from './second/second.component'



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

first.component.css

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

```
  }
```

First.component.html

```
<div class="cmp">First Component</div>
```

First.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-first',
  templateUrl: './first.component.html',
  styleUrls: ['./first.component.css']
})
export class FirstComponent {
}
```

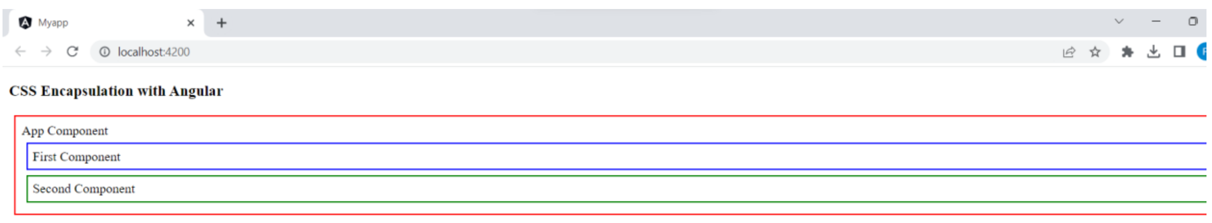
Second.component.css

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

Second.component.html

```
<div class="cmp">Second Component</div>
```

Output:



CSS Encapsulation with Angular

App Component

First Component

Second Component

None encapsulation

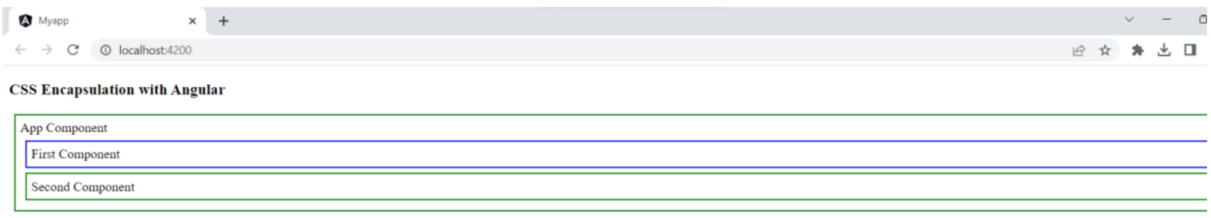
Second.component.ts

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

@Component({

  selector: 'app-second',

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

  styleUrls: ['./second.component.css'],

  encapsulation: ViewEncapsulation.None

})

export class SecondComponent {

}
```

App.component.ts

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

@Component({

  selector: 'app-root',

  styleUrls: ['./app.component.css'],

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

  encapsulation: ViewEncapsulation.None

})

export class AppComponent {

}
```

OutPut:



CSS Encapsulation with Angular

App Component

First Component

Second Component

7.b Course Name: Angular JS Module Name: Model Driven Forms or Reactive Forms Create an employee registration form as a reactive form.

Description:

Forms in Angular:

Angular has two different approaches in dealing with forms: reactive forms and template-driven forms.Both reactive forms and template-driven forms:can capture user-provided data, can capture user input events, can validate the user input, etc. have their own approaches of processing and managing the form data:In template-driven forms, you will create the form completely in the template and need to rely on directives to create and manipulate the underlying form object model. Since the template-driven forms do not scale that well, they are more suitable only when you want to add a simple small form to the application. For example: a signup form.

In reactive forms, you can control the form completely from the component class and hence you will get direct, explicit access to the underlying forms object model. Hence, reactive forms are also known as 'model-driven forms'. As reactive forms are more robust and scalable, they are more suitable for creating all kind of forms in an application, irrespective of the size of form.

Program:

//app.module.ts

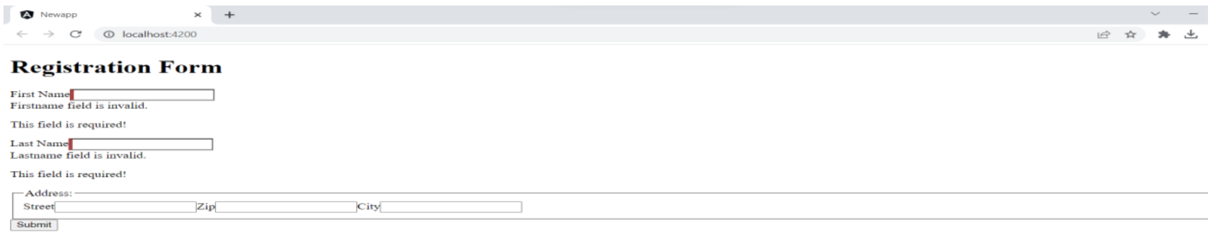
```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { RegistrationFormComponent } from './registration-
form/registration-form.component'
import { ReactiveFormsModule } from '@angular/forms';
@NgModule({
  declarations: [
    AppComponent,
    RegistrationFormComponent
  ],
  imports: [
    BrowserModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

//registration-form.component.ts
```
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: []
      })
    });  }}
```
//app.component.html
```
<app-registration-form></app-registration-form>
```
//registration-form.component.css
```
.ng-valid[required]  {
    border-left: 5px solid #42A948; /* green */
  }
  .ng-invalid:not(form)  {
    border-left: 5px solid #a94442; /* red */
  }
```
//registration-form.component.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">
            <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>
```

<u>OUTPUT:</u>



Newapp
localhost:4200

Registration Form

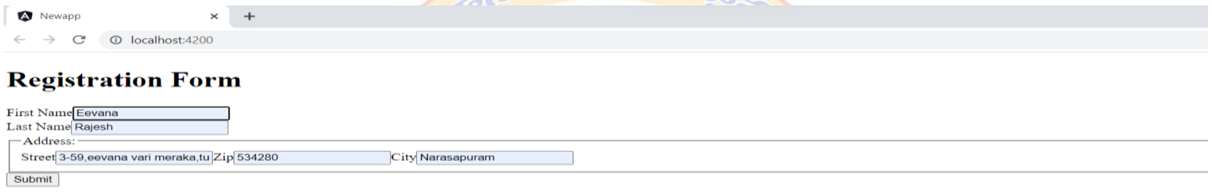
First Name
Firstname field is invalid.

This field is required!

Last Name
Lastname field is invalid.

This field is required!

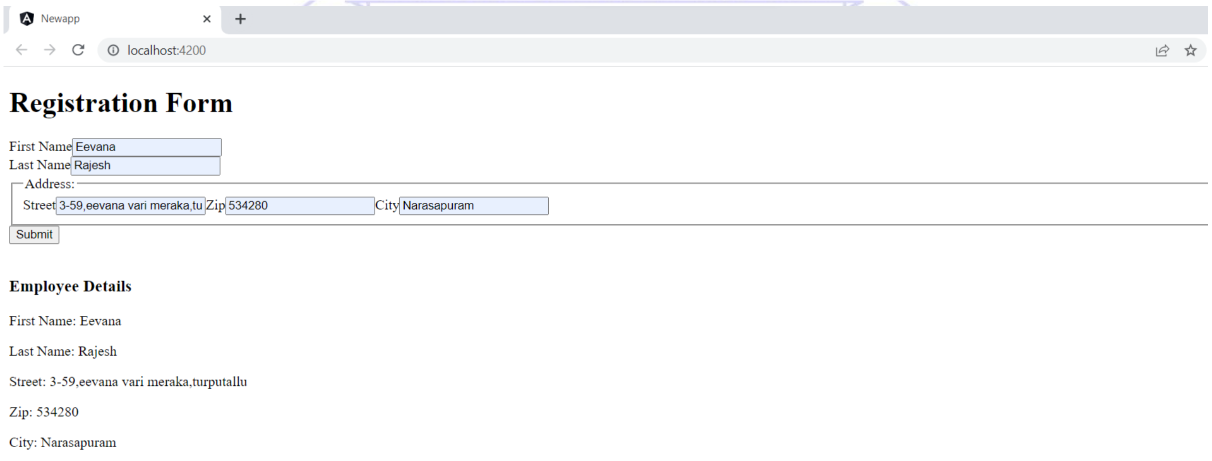
Address:
Street Zip City
Submit



Newapp
localhost:4200

Registration Form

First Name Eevana
Last Name Rajesh
Address:
Street 3-59,eevana vari meraka,tu Zip 534280 City Narasapuram
Submit



Newapp
localhost:4200

Registration Form

First Name Eevana
Last Name Rajesh
Address:
Street 3-59,eevana vari meraka,tu Zip 534280 City Narasapuram
Submit

Employee Details

First Name: Eevana

Last Name: Rajesh

Street: 3-59,eevana vari meraka,turputallu

Zip: 534280

City: Narasapuram

7.c Course Name: Angular JS Module Name: Custom Validators in Reactive Forms Create a custom validator for an email field in the employee registration form (reactive form).

Program:

//app.module.ts

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

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

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

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

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

@NgModule({

  declarations: [

    AppComponent,

    RegistrationFormComponent

  ],

  imports: [

    BrowserModule,

    ReactiveFormsModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }
```

//registration-form.component.ts

```
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: ['', 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!"

    }

  };

}
```

//app.component.html

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

//registration-form.component.css

```
.ng-valid[required]  {

    border-left: 5px solid #42A948; /* green */

  }

  .ng-invalid:not(form)  {

    border-left: 5px solid #a94442; /* red */

  }
```

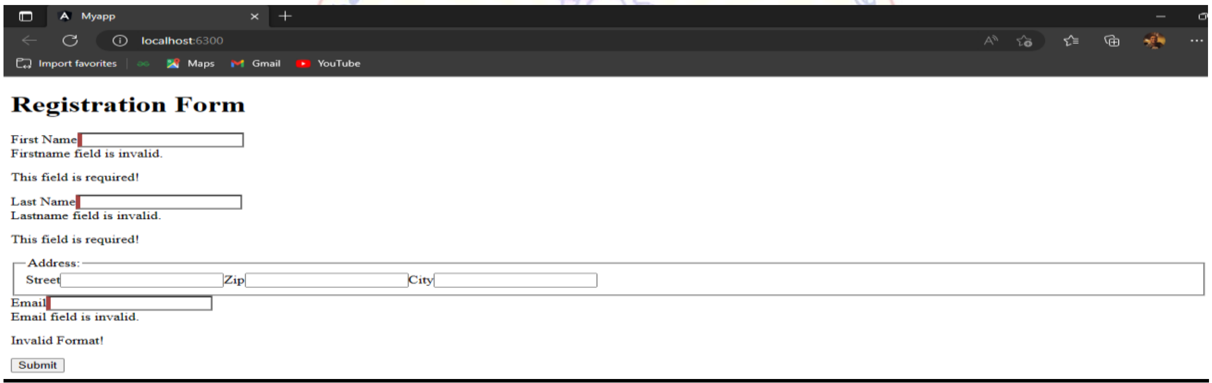
//registration-form.component.html

```
<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">

          <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>

      <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'].messag
e }}

        </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>
```

OUTPUT:



Registration Form

First Name
Firstname field is invalid.

This field is required!

Last Name
Lastname field is invalid.

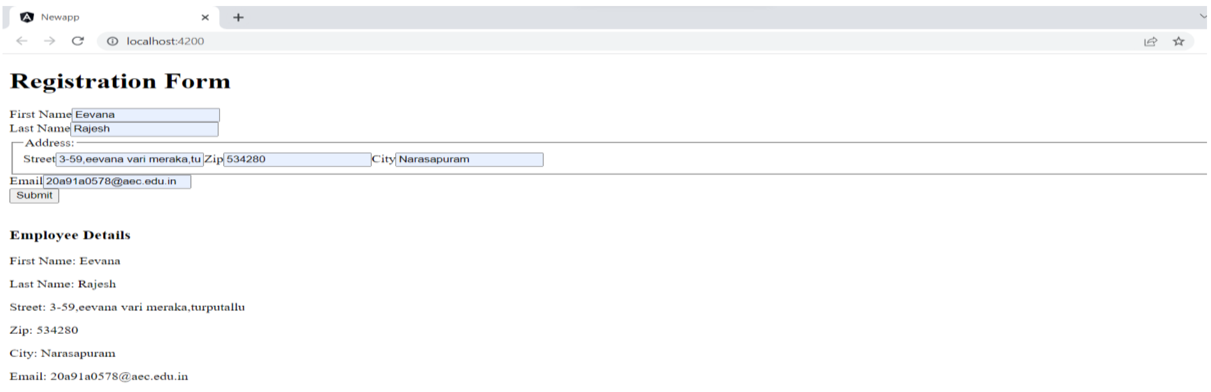
This field is required!

Address:
Street Zip City

Email
Email field is invalid.

Invalid Format!

Submit



Registration Form

First Name Eevana
Last Name Rajesh
Address:
Street 3-59,eevana vari meraka,tu Zip 534280 City Narasapuram
Email 20a91a0578@aec.edu.in
Submit

Employee Details

First Name: Eevana

Last Name: Rajesh

Street: 3-59,eevana vari meraka,turputallu

Zip: 534280

City: Narasapuram

Email: 20a91a0578@aec.edu.in

7.a Course Name: Angular JS Module Name: Template Driven Forms Create a course registration form as a template-driven form.

Description:

In Angular, there are two approaches for creating forms - template-driven and reactive.Template-driven forms are based on two-way data binding between the form controls in the template and the model object in the component class. This means that any changes made to the form controls will be immediately reflected in the model object, and vice versa.To create a template-driven form in Angular, we first need to create a form template in the component's HTML file. This form template contains the form controls and their corresponding data bindings to the model object. We can also add validation rules to the form controls using built-in or custom validators.

Program:

registration-form.component.html

```
<for#registrationForm="ngForm"
(ngSubmit)="onSubmit(registrationForm)">

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

  <input type="text" id="name" name="name" ngModel required>


  <label for="email">Email:</label>

  <input type="email" id="email" name="email" ngModel required email>


  <label for="course">Course:</label>

  <select id="course" name="course" ngModel required>

    <option *ngFor="let course of courses"
[value]="course">{{course}}</option>

  </select>


  <button type="submit"
[disabled]="!registrationForm.valid">Register</button>

</form>

<p *ngIf="submitted">
```

```
  Name: {{registrationForm.value.name}}<br>
  Email: {{registrationForm.value.email}}<br>
  Course: {{registrationForm.value.course}}
</p>
```

registration-form.component.ts

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


@Component({
  selector: 'app-registration-form',
  templateUrl: './registration-form.component.html'
})
export class RegistrationFormComponent {
  courses = ['Angular', 'React', 'Vue'];

  submitted = false;

  onSubmit(form: NgForm) {
    this.submitted = true;
  }
}
```

app.module.ts

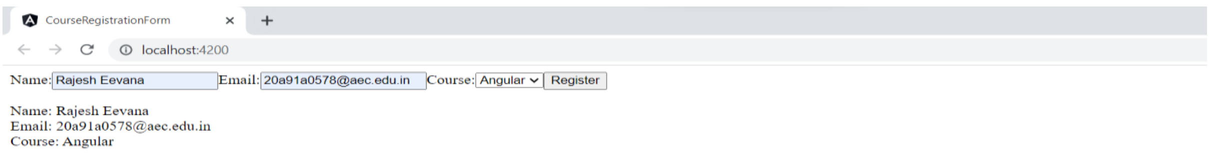
```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { RegistrationFormComponent } from './registration-form/registration-form.component';
```

```
@NgModule({
  declarations: [
    AppComponent,
    RegistrationFormComponent
  ],
  imports:[
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

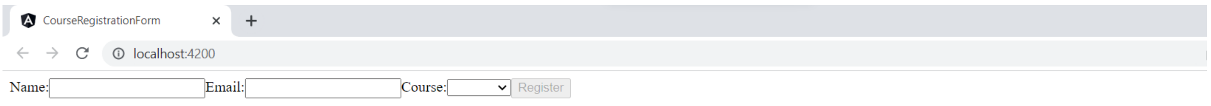
app.component.html

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

Output:


CourseRegistrationForm
localhost:4200
Name: Rajesh Eevana Email: 20a91a0578@aec.edu.in Course: Angular Register

```
Name: Rajesh Eevana
Email: 20a91a0578@aec.edu.in
Course: Angular
```


CourseRegistrationForm
localhost:4200
Name: Email: Course: Register

8a) Course Name: Angular JS
Module Name: Custom Validators in Template Driven forms Create a custom validator for the email field in the course registration form.

Aim: To Create a custom validator for the email field in the course registration form.
Description: While creating forms, there can be situations for validations for which built-in validators are not available. Few such examples include validating a phone number, validating if the password and confirm password fields matches or not, etc., In such situations, we can create custom validators to implement the required functionality.

Program:
Course.ts
```
export class Course {
  courseId:number;
  courseName:string;
  duration:string;
  email:string;
  constructor(courseId:      number,courseName:      string,duration:
string,email:string) {
    this.courseId=courseId;
    this.courseName=courseName;
    this.duration=duration;
    this.email=email;
  }
}
```
Course-form.component.css
```
input.ng-valid[required]  {
    border-left: 5px solid #42A948;
  }
  input.ng-dirty.ng-invalid:not(form)  {
    border-left: 5px solid #a94442;
  }
```

Course-form.component.html
```
<div class="container">
    <div [hidden]="submitted">
    <h1>Course Form</h1>
    <form (ngSubmit)="onSubmit()" #courseForm="ngForm">
    <div class="form-group">
    <label for="id">Course Id</label>
    <input type="text" class="form-control" required
   [(ngModel)]="course.courseId" name="id" #id="ngModel">
    <div [hidden]="id.valid || id.pristine" class="alert
   alert-danger">
    Course Id is required</div>
    </div>
    <div class="form-group">
    <label for="name">Course Name</label>
    <input type="text" class="form-control" required
   [(ngModel)]="course.courseName"
    minlength="4" name="name" #name="ngModel">
    <div *ngIf="name.errors && (name.dirty ||
   name.touched)" class="alert alert-danger">
    <div [hidden]="!name.errors['required']">Name is
   required</div>
   <div [hidden]="!name.errors['minlength']">Name must
    be at least 4 characters long.</div>
     </div>
     </div>
     <div class="form-group">
     <label for="duration">Course Duration</label>
     <input type="text" class="form-control" required
    [(ngModel)]="course.duration"
     name="duration" #duration="ngModel">
     <div [hidden]="duration.valid || duration.pristine"
    class="alert alert-danger">Duration is required</div>
     </div>
     <div class="form-group">
     <label for="email">Author Email</label>
     <input type="text" class="form-control" required
    [(ngModel)]="course.email"
     name="email" #email="ngModel" validateEmail>
     <div *ngIf="email.errors && (email.dirty ||
    email.touched)" class="alert alert-danger">
     <div [hidden]="!email.errors['required']">Email is
    required</div>
     <div
    [hidden]="!email.errors['emailInvalid']">{{email.errors['email
    Invalid']}}</div>
     </div>
     </div>

     <button type="submit" class="btn btn-default"
    [disabled]="!courseForm.form.valid">Submit</button>
     <button type="button" class="btn btn-default"
    (click)="courseForm.reset()">New Course</button>
     </form>
     </div>
     <div [hidden]="!submitted">
     <h2>You submitted the following:</h2>
     <div class="row">
     <div class="col-xs-3">Course ID</div>
     <div class="col-xs-9 pull-left">{{ course.courseId
    }}</div>
     </div>
     <div class="row">
     <div class="col-xs-3">Course Name</div>
     <div class="col-xs-9 pull-left">{{ course.courseName
    }}</div>
     </div>
     <div class="row">
     <div class="col-xs-3">Duration</div>
     <div class="col-xs-9 pull-left">{{ course.duration
    }}</div>
     </div>
     <div class="row">
        <div class="col-xs-3">Email</div>
        <div class="col-xs-9 pull-left">{{ course.email }}</div>
        </div>
        <br>
        <button class="btn btn-default"
       (click)="submitted=false">Edit</button>
        </div>
       </div>
```

Course-form.component.ts

```
  import { Component } from '@angular/core';
import { Course } from './course';
@Component({
 selector: 'app-course-form',
 templateUrl: './course-form.component.html',
 styleUrls: ['./course-form.component.css']
})
export class CourseFormComponent {
 course: Course = new Course(1, 'Angular 2', '4 days',
'devikan4353@gmail.com');
 submitted = false;
 onSubmit() { this.submitted = true; }
}
```

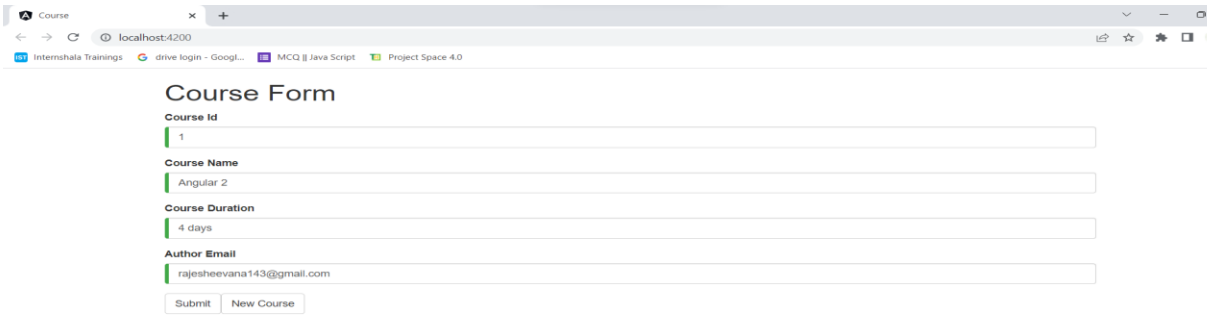
Email.validator.ts

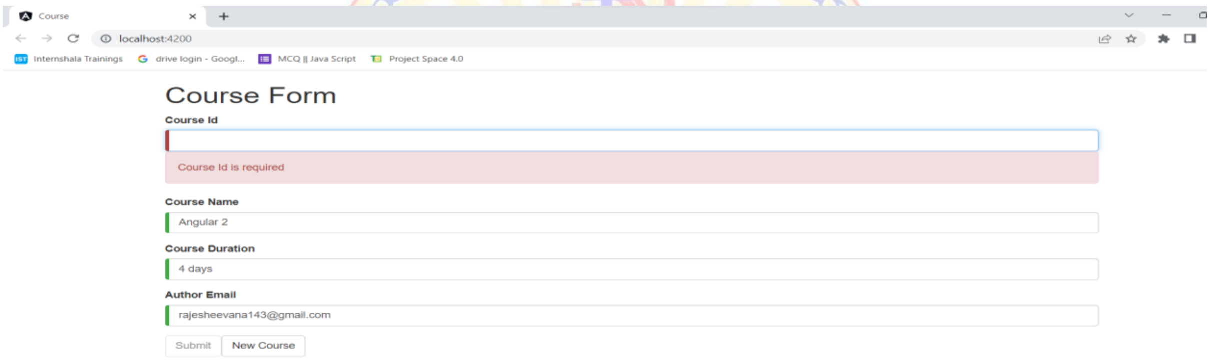
```
import { Directive } from '@angular/core';
import { NG_VALIDATORS, FormControl, Validator } from
'@angular/forms';
@Directive({
  selector: '[validateEmail]',
  providers: [
    { provide: NG_VALIDATORS, useExisting: EmailValidator, multi:
true },
  ],
})
export class EmailValidator implements Validator {
  validate(control: FormControl): any {
    const emailRegexp =
      /^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$/;
    if (!emailRegexp.test(control.value)) {
      return { emailInvalid: 'Email is invalid' };
    }
    return null;
  }
}
```

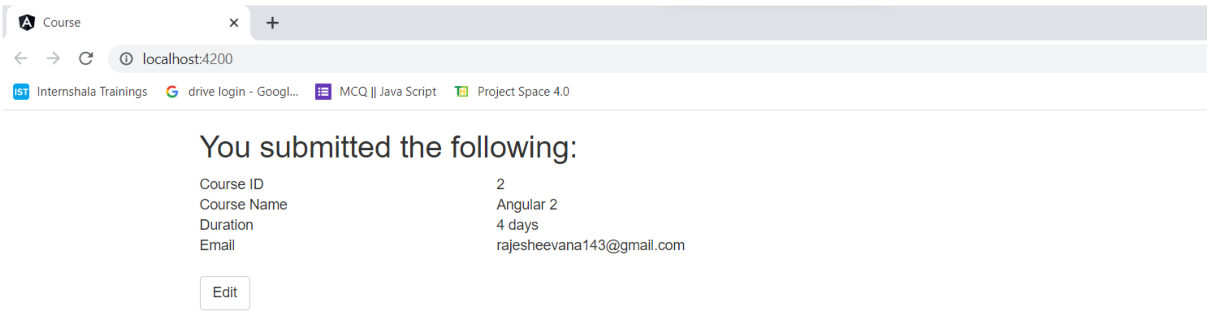
App.component.html

```
<app-course-form></app-course-form>
```

OUTPUT:


Course Form
Course Id
1
Course Name
Angular 2
Course Duration
4 days
Author Email
rajesheevana143@gmail.com


Course Form
Course Id
Course Id is required
Course Name
Angular 2
Course Duration
4 days
Author Email
rajesheevana143@gmail.com


You submitted the following:

```
Course ID          2
Course Name        Angular 2
Duration           4 days
Email              rajesheevana143@gmail.com
```

8b) Course Name: Angular JS

Module Name: Services Basics

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.

Aim: To 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.

Description:

A service in Angular is a class that contains some functionality that can be reused across the application. A service is a singleton object. Angular services are a mechanism of abstracting shared code and functionality throughout the application.

Angular Services come as objects which are wired together using dependency injection. Angular provides a few inbuilt services also can create custom services.

Services can be used to:

1.share the code across components of an application.

2.make HTTP requests.

Creating a Service

To create a service class, use the following command:

Command: ng generate service book

Program:

Book.ts

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

Books-data.ts

```
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: 'Node Js' },
 { id: 5, name: 'Typescript' },
 { id: 6, name: 'Angular Js' }
];
```

Book.service.ts

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

Book.component.ts

```
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();
 }
}
```

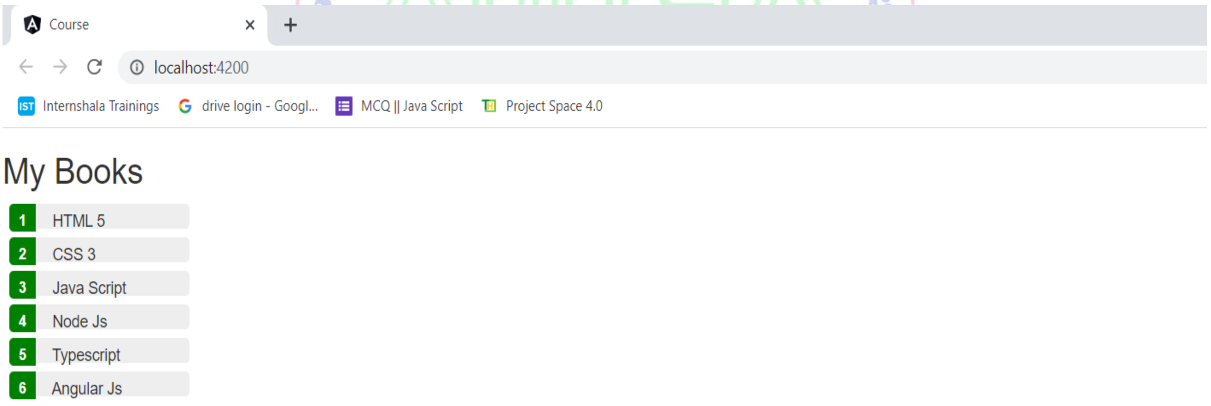
Book.component.html

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

Book.component.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;
    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: green;
    line-height: 0.5em;
    position: relative;
    left: -1px;
    top: -4px;
    height: 1.8em;
    margin-right: 0.8em;
    border-radius: 4px 0 0 4px;
    }
```

Output:



My Books

```
1  HTML 5
2  CSS 3
3  Java Script
4  Node Js
5  Typescript
6  Angular Js
```

8c) Course Name: Angular JS

Module Name: RxJS Observables

Create and use an observable in Angular.

Aim: To create and use an observable in Angular.

Description: RxJS Reactive Extensions for JavaScript (RxJS) is a third-party library used by the Angular team. RxJS is a reactive streams library used to work with asynchronous streams of data. Observables, in RxJS, are used to represent asynchronous streams of data. Observables are a more advanced version of Promises in JavaScript

Program:

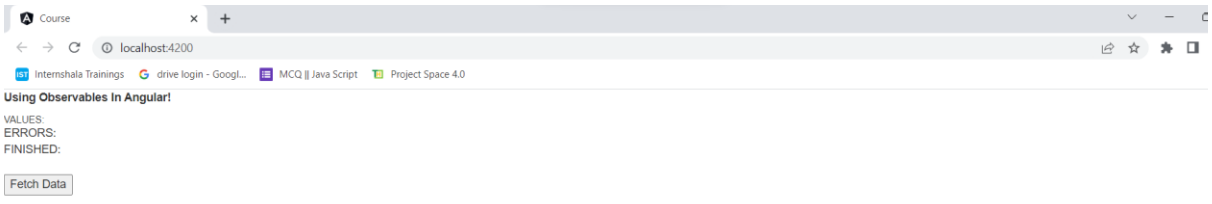
app.component.ts:

```
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(43); }, 1000),
 setTimeout(() => { observer.next(54); }, 2000),
 setTimeout(() => { observer.complete(); }, 3000);
 });
 this.data.subscribe((value) => this.myArray.push(value),
 error => this.errors = true,
 () => this.finished = true);
 }
}
```

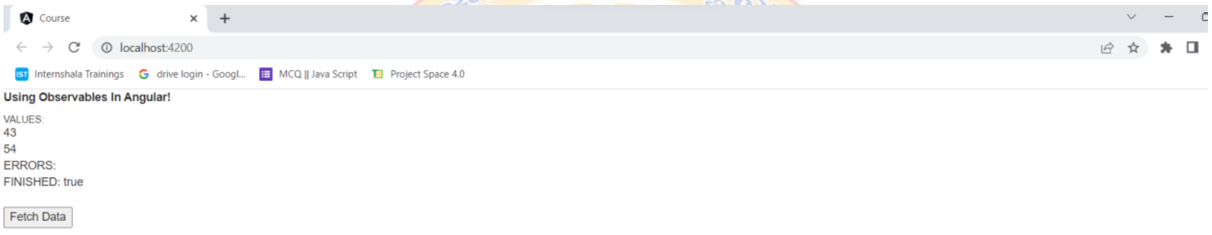
App.component.html

```
<b> Using Observables In Angular!</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>
```

Output:



```
Using Observables In Angular!
VALUES:
ERRORS:
FINISHED:

Fetch Data
```



```
Using Observables In Angular!
VALUES:
43
54
ERRORS:
FINISHED: true

Fetch Data
```

9.a Course Name: Angular JS
Module Name: Server Communication using HttpClient
Create an application for Server Communication using HttpClient

Description:
Most front-end applications communicate with backend services using HTTP Protocol While making calls to an external server, the users must continue to be able to interact with the page. That is, the page should not freeze until the HTTP request returns from the external server. So, all HTTP requests are asynchronous.
HttpClient from @angular/common/http to communicate must be used with backend services.Additional benefits of HttpClient include testability features, typed request and response objects, request and response interception, Observable APIs, and streamlined error handling.
HttpClientModule must be imported from @angular/common/http in the module class to make HTTP service available to the entire module. Import HttpClient service class into a component's constructor. HTTP methods like get, post, put, and delete are made used off.JSON is the default response type for HttpClient.
Program:
Book.component.html
```
<h1>Books Information</h1>
<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>
```
Book.component.ts
```
import { Component } from '@angular/core';
import { 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 {
  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() {
    this.getBooks();
    console.log(this.getBooks());}}
```

Book.service.ts

```
import { Injectable } from '@angular/core';
import   {   HttpClient,   HttpErrorResponse,   HttpHeaders   }   from
'@angular/common/http';
import { catchError, tap } from 'rxjs/operators';
import { Observable, throwError } from 'rxjs';
import { Book } from './book';
@Injectable({
    providedIn: 'root'
})
export class BookService {
  constructor(private http: HttpClient) { }
  getBooks(): Observable<Book[]> {
return this.http.get<Book[]>('http://localhost:3020/bookList').pipe(
      tap((data: any) => data),      catchError(this.handleError));}
  private handleError(err: HttpErrorResponse): Observable<any> {
    let errMsg = '';
    if (err.error instanceof Error) {
      console.log('An error occurred:', err.error.message);
      errMsg = err.error.message;
    } else {
      console.log(`Backend returned code ${err.status}`);
      errMsg = err.error.status;}return throwError(()=>errMsg);}}
```

Book.ts

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

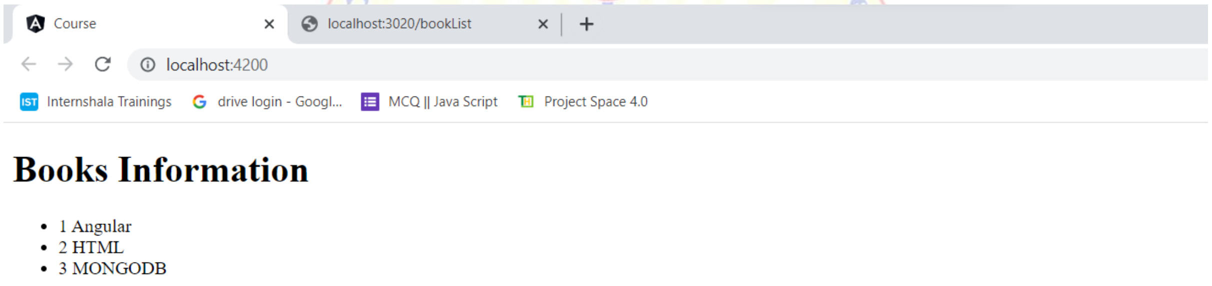
App.module.ts

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

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

Hello.js

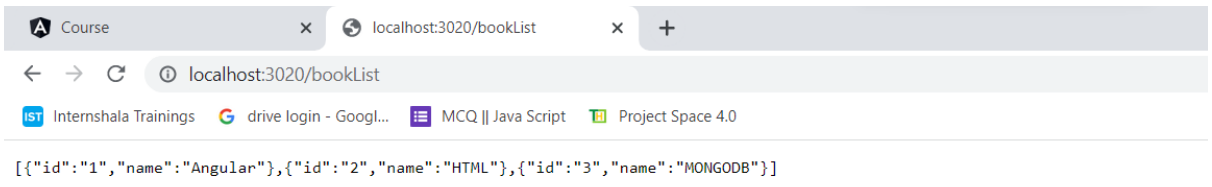
```
const express=require('express')
const cors=require('cors')
const app=express();
const
data=[{id:'1',name:"Angular"},{id:"2",name:'HTML'},{id:'3',name:"MON
GODB"}]
app.use(cors())
app.use(express.json())
app.get('/bookList',(req,res)=>{
    res.send(data);
})
app.listen(3020,()=>{console.log('server ruuning ........')})
```

Output:



Books Information

- 1 Angular
- 2 HTML
- 3 MONGODB



```
[{"id":"1","name":"Angular"},{"id":"2","name":"HTML"},{"id":"3","name":"MONGODB"}]
```

9b) Course Name: Angular JS

Module Name: Communicating with different backend services using Angular HttpClient

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

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

Description:

Communicating with different backend services using Angular HTTP Client Angular can also be used to connect to different services written in different technologies/languages. For example, we can make a call from Angular to Node/Express or Java or Mainframe or .Net services, etc.

Program:

Product.json

```
[
    {
        "productId": 1,
"productName": "Samsung Galaxy Note 7",
"productCode": "MOB-120",
"description": "64GB, Coral Blue",
"price": 800,
"productimg":"https://drop.ndtv.com/TECH/product_database/images/822
01685704PM_635_samsung_galaxy_note7.jpeg",
"manufacturer": "Samsung",
"ostype": "Android",
"rating": 4
    },{
        "productId": 2,
"productName": "Samsung Galaxy Note 7",
"productCode": "MOB-124",
"description": "64GB, Gold",
"price": 850,
"productimg":"https://www.91-img.com/pictures/samsung-galaxy-note-7-
mobile-phone-large-1.jpg",
"manufacturer": "Samsung",
"ostype": "Android",
"rating": 4

    },{
        "productId": 1,
"productName": "Apple iPad Mini 2",
"productCode": "TAB-120",
"description": "16GB, White",
"price": 450,
"productimg":"https://fdn2.mobgsm.com/vv/pics/apple/apple-ipad-
mini2.jpg",
"manufacturer": "Apple",
"ostype": "iOS",
"rating": 4

    },{
        "productId": 2,
"productName": "Apple iPad Air2",
"productCode": "TAB-124",
"description": "64GB, Black",
"price": 600,
"productimg":"https://www.mega.pk/items_images/t_8385.png",
"manufacturer": "Apple",
"ostype": "iOS",
"rating": 3
    }
]
```

Product.service.ts

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

@Injectable({
  providedIn: 'root'
})
export class ProductService {
  private productsUrl = 'assets/products.json';

  constructor(private http: HttpClient) {}

  getProducts(): Observable<any> {
    return this.http.get(this.productsUrl);
  }
}
```

Book.component.html

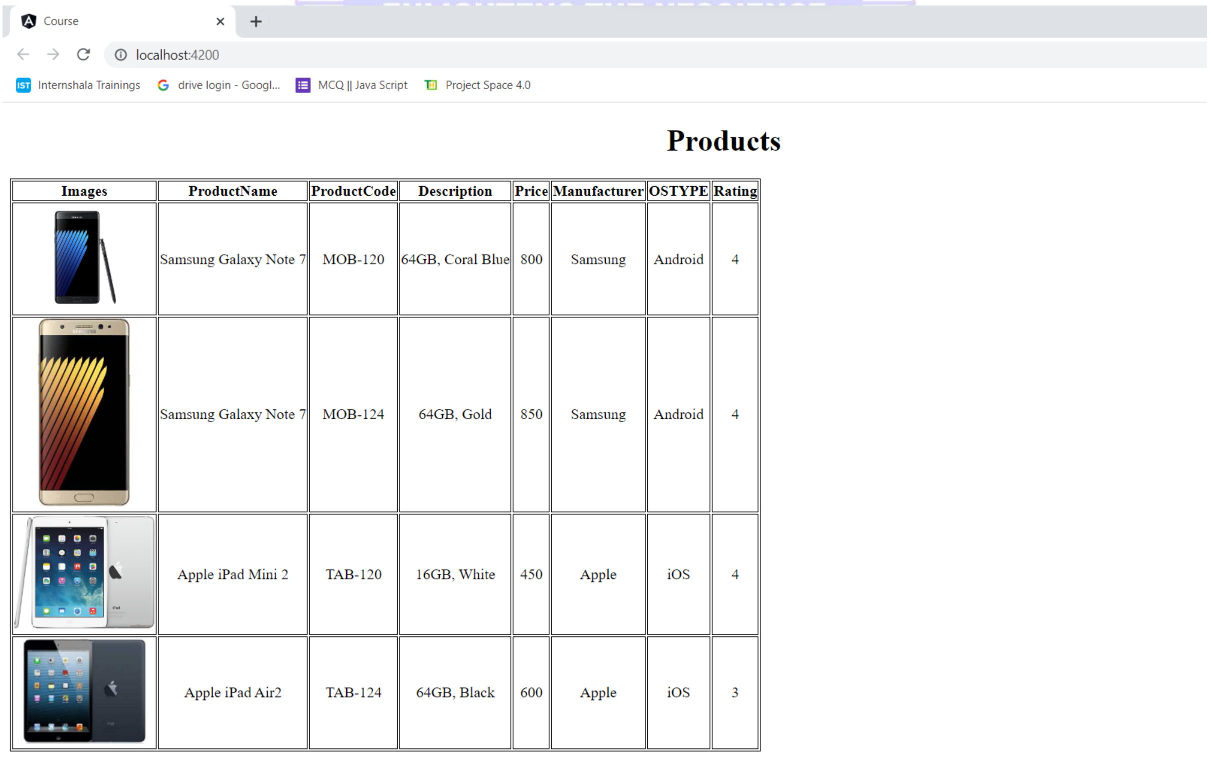
```
  <h1 style="text-align: center;">Products</h1>
<div style="text-align: center;">
  <table id="table">
    <thead>

<th>Images</th><th>ProductName</th><th>ProductCode</th><th>Descripti
on</th><th>Price</th><th>Manufacturer</th><th>OSTYPE</th><th>Rating<
/th>
    </thead>
    <tbody><tr    *ngFor="let    product    of    products"><td><img
src={{product.productimg}}        width="150px"        /></td><td>{{
```

```
product.productName }}</td><td>{{ product.productCode }}</td><td>{{
product.description}}</td>

<td>{{product.price}}</td><td>{{product.manufacturer}}</td><td>{{pro
duct.ostype}}</td><td>{{product.rating}}</td></tr>
    </tbody>
  </table>
</div>
```

Book.component.ts

```
import { Component } from '@angular/core';
import { OnInit } from '@angular/core';
import { ProductService } from './productservice.service';
@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']})
export class BookComponent implements OnInit{
  products: any[]=new Array();
  constructor(private productService: ProductService) {}
ngOnInit() {
    this.productService.getProducts()
      .subscribe(products                                    =>
{this.products=products;console.log(this.products)});
  }}
```

Output



Course
localhost:4200
Internshala Trainings · drive login - Googl... · MCQ || Java Script · Project Space 4.0

Products

Images	ProductName	ProductCode	Description	Price	Manufacturer	OSTYPE	Rating
	Samsung Galaxy Note 7	MOB-120	64GB, Coral Blue	800	Samsung	Android	4
	Samsung Galaxy Note 7	MOB-124	64GB, Gold	850	Samsung	Android	4
	Apple iPad Mini 2	TAB-120	16GB, White	450	Apple	iOS	4
	Apple iPad Air2	TAB-124	64GB, Black	600	Apple	iOS	3

10.a Course Name: Angular JS

Module Name: Routing Basics, Router Links

Create multiple components and add routing to provide navigation between them.

Description:

Routing means navigation between multiple views on a single page.
Routing allows to express some aspects of the application's state in the URL. The full application can be built without changing the URL.
Routing allows to:

- Navigate between the views
- Create modular applications

Configuring Router
Angular uses Component Router to implement routing
A <base> tag must be added to the head tag in the HTML page to tell the router where to start with.

Component1.component.html

```
<h1 style="text-align: center;">This is Component 1</h1>
```

Component2.component.html

```
<h1 style="text-align: center;">This is Component 2</h1>
```

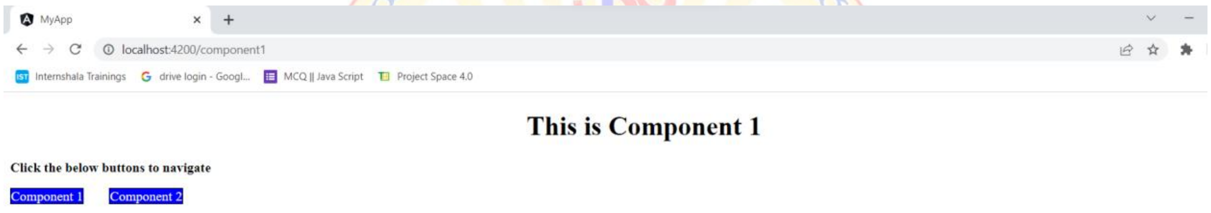
App-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import {Component1Component} from'./component1/component1.component'
import {Component2Component} from'./component2/component2.component'
const routes: Routes = [
  { path: 'component1', component: Component1Component },
  { path: 'component2', component: Component2Component },
];

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

App.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import {Component1Component} from'./component1/component1.component'
import {Component2Component} from'./component2/component2.component'
@NgModule({
  declarations: [
    AppComponent,
    Component1Component,
    Component2Component
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

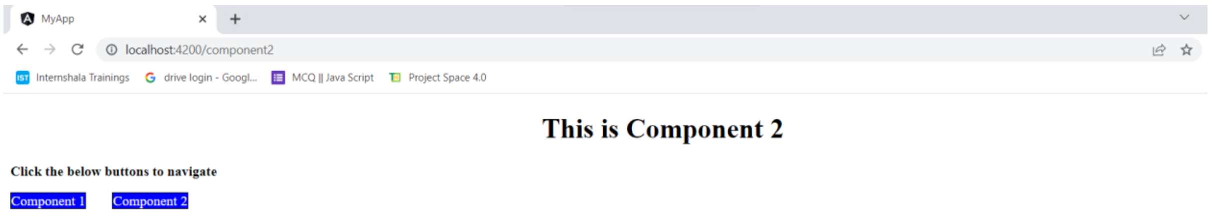
Output:



This is Component 1

Click the below buttons to navigate

Component 1 Component 2



This is Component 2

Click the below buttons to navigate

Component 1 Component 2

10.b Course Name: Angular JS
Module Name: Route Guards
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 to access the BookComponent it sholu give error.

Description:
In the Angular application, users can navigate to any URL directly. That's not the right thing to do always.
Consider the following scenarios

- Users must login first to access a component
- The user is not authorized to access a component
- User should fetch data before displaying a component
- Pending changes should be saved before leaving a component

These scenarios must be handled through route guards.
A guard's return value controls the behavior of the router
If it returns true, the navigation process continues
If it returns false, the navigation process stops
Angular has canActivate interface which can be used to check if a user is logged in to access a component canActivate() method must be overridden in the guard class as shown below:
Using canActivate, access can be permitted to only authenticated users.

Program:
Login.component.html
```
<h2>Login</h2>
<form (ngSubmit)="onSubmit()">
  <div>
    <label for="username">Username:</label>
    <input type="text" id="username" name="username"
[(ngModel)]="usernameValue" required>
  </div>
  <div>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password"
[(ngModel)]="passwordValue" required>
  </div>
  <button type="submit">Login</button>
</form>
```

Login.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from '../auth.service';
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  usernameValue = '';
  passwordValue = '';
  constructor(private authService: AuthService, private router:
Router) {}
  onSubmit(): void {
    if (this.authService.login(this.usernameValue,
this.passwordValue)) {
      this.router.navigate(['/component2']);
    } else {
      // Display an error message if login is unsuccessful
    }
  }
}
```

App-routing.component.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { Component1Component } from
'./component1/component1.component';
import { Component2Component } from
'./component2/component2.component';
import { LoginComponent } from './login/login.component';
import { AuthGuard } from './auth.guard';

const routes: Routes = [
  { path: 'component1', component: Component1Component },
  { path: 'component2', component: Component2Component, canActivate:
[AuthGuard] },
  { path: 'login', component: LoginComponent },
];

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

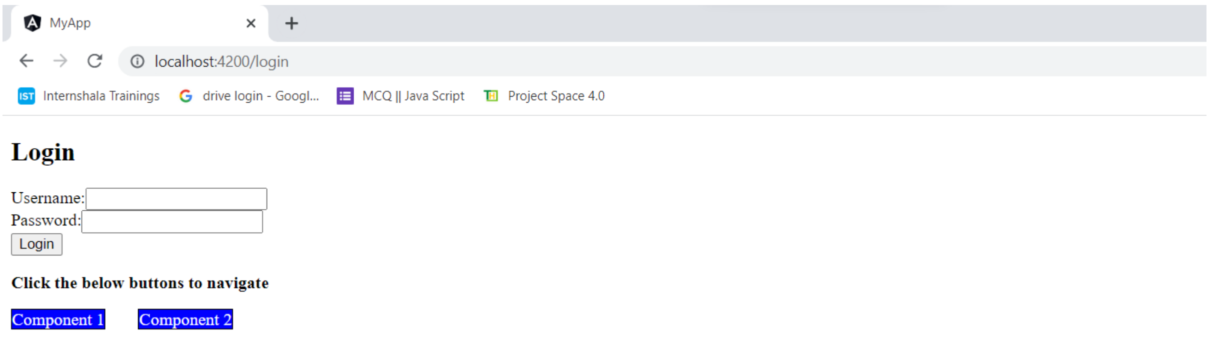
Auth,guard.ts

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

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(private authService: AuthService, private router:
Router) {}

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> |
Promise<boolean | UrlTree> | boolean | UrlTree {
    if (this.authService.isLoggedIn()) {
      return true;
    } else {
      this.router.navigate(['/login']);
      return false;
    }
  }
}
```

Auth.service.ts

```
import { Injectable } from '@angular/core';
@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private loggedIn = false;

  constructor() {}

  login(username: string, password: string): boolean {
    if (username === 'user' && password === 'password') {
      this.loggedIn = true;
      return true;
    }
    return false;
  }

  logout(): void {
    this.loggedIn = false;
  }

  isLoggedIn(): boolean {
    return this.loggedIn;
  }
}
```

Output:

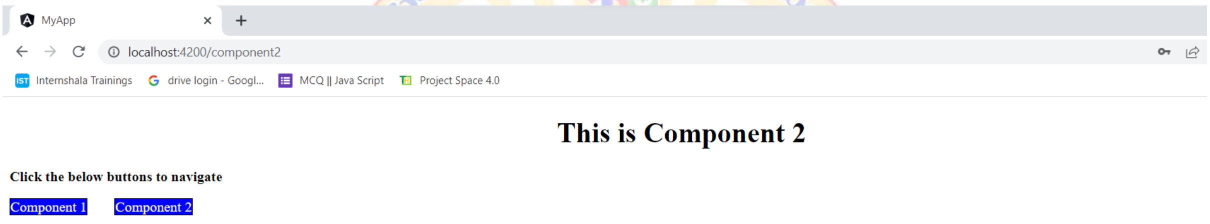

MyApp
localhost:4200/login
Internshala Trainings   drive login - Googl...   MCQ || Java Script   Project Space 4.0

Login

Username:
Password:
Login

Click the below buttons to navigate

Component 1   Component 2


MyApp
localhost:4200/component2
Internshala Trainings   drive login - Googl...   MCQ || Java Script   Project Space 4.0

This is Component 2

Click the below buttons to navigate

Component 1   Component 2

10.c Course Name: Angular JS

Module Name: Asynchronous Routing

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.

Description:

When an Angular application has a lot of components, it will increase the size of the application. In such cases, the application takes a lot of time to load.
To overcome this problem, asynchronous routing is preferred, i.e, modules must be loaded lazily only when they are required instead of loading them at the beginning of the execution

Lazy Loading has the following benefits:

Modules are loaded only when the user requests for it
Load time can be speeded up for users who will be visiting only certain areas of the application

Lazy Loading Route Configuration:

To apply lazy loading on modules, create a separate routing configuration file for that module and map an empty path to the component of that module.

Program:

Login.component.html

```
<h2>Login</h2>
<form (ngSubmit)="onSubmit()">
  <div>
    <label for="username">Username:</label>
    <input type="text" id="username" name="username"
[(ngModel)]="usernameValue" required>
  </div>
  <div>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password"
[(ngModel)]="passwordValue" required>
  </div>
  <button type="submit">Login</button>
</form>
```

Login.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from '../auth.service';
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  usernameValue = '';
  passwordValue = '';
  constructor(private authService: AuthService, private router:
Router) {}
  onSubmit(): void {
    if (this.authService.login(this.usernameValue,
this.passwordValue)) {
      this.router.navigate(['/component2']);
    } else {
      // Display an error message if login is unsuccessful
    }
  }
}
```

App-routing.component.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { Component1Component } from
'./component1/component1.component';
import { LoginComponent } from './login/login.component';
import { AuthGuard } from './auth.guard';

const routes: Routes = [
  { path: 'component1', component: Component1Component },
  { path: 'component2', loadChildren: () =>
import('./component2/component2.module').then(m =>
m.Component2Module), canActivate: [AuthGuard] },
  { path: 'login', component: LoginComponent },
];

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

Auth.guard.ts

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

```
@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(private authService: AuthService, private router:
Router) {}

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> |
Promise<boolean | UrlTree> | boolean | UrlTree {
    if (this.authService.isLoggedIn()) {
      return true;
    } else {
      this.router.navigate(['/login']);
      return false;
    }
  }
}
```

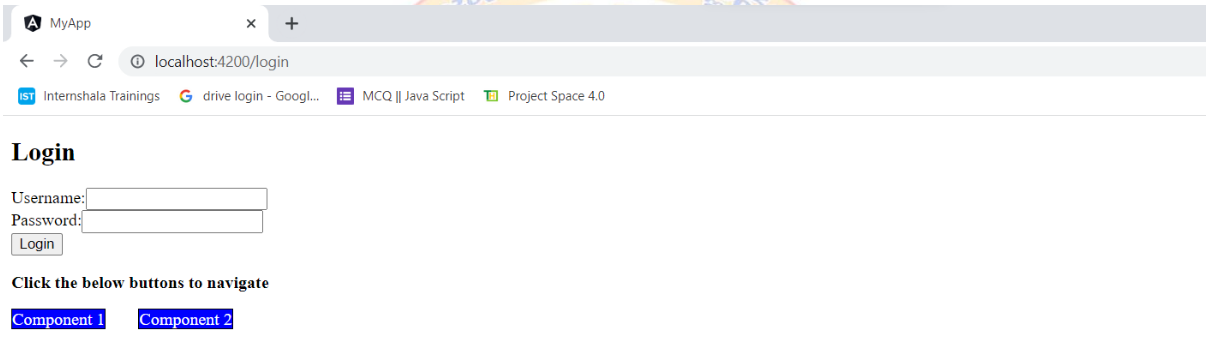
Auth.service.ts

```
import { Injectable } from '@angular/core';
@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private loggedIn = false;

  constructor() {}

  login(username: string, password: string): boolean {
    if (username === 'user' && password === 'password') {
      this.loggedIn = true;
      return true;
    }
    return false;
  }

  logout(): void {
    this.loggedIn = false;
  }

  isLoggedIn(): boolean {
    return this.loggedIn;
  }
}
```

Component.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';
import { Component2Component } from './component2.component';

@NgModule({
  declarations: [Component2Component],
  imports: [
    CommonModule,
    RouterModule.forChild([
      { path: '', component: Component2Component }
    ])
  ],
  exports: [Component2Component]
})
export class Component2Module { }
```

Output:

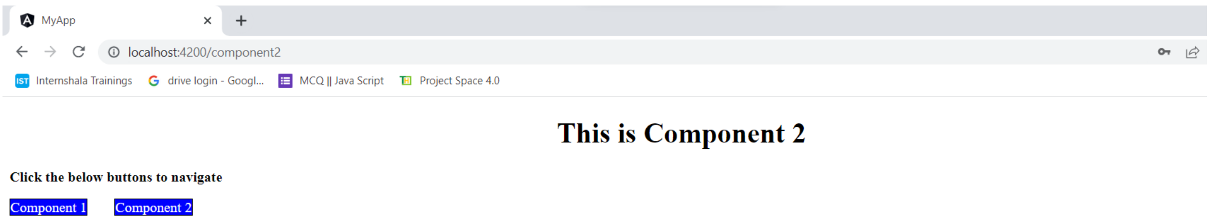


Login

Username:
Password:
Login

Click the below buttons to navigate

Component 1 Component 2



This is Component 2

Click the below buttons to navigate

Component 1 Component 2

10.d Course Name: Angular JS
Module Name: Nested Routes
Implement Child Routes to a submodule.

Description:
Program:
In Angular, nested routes are routes within other routes. They allow us to create a hierarchy of routes and display components within other components. Nested routes are useful when we want to implement the common UI pattern known as Master Detail. To define nested routes, we need to use the **children** property of the **Route** object.

my-submodule.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { MySubmoduleComponent } from './my-submodule.component';

const routes: Routes = [
  { path: '', component: MySubmoduleComponent }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule],
  declarations: [
    MySubmoduleComponent
  ]
})
export class MySubmoduleRoutingModule { }
```

App-routing.module.ts

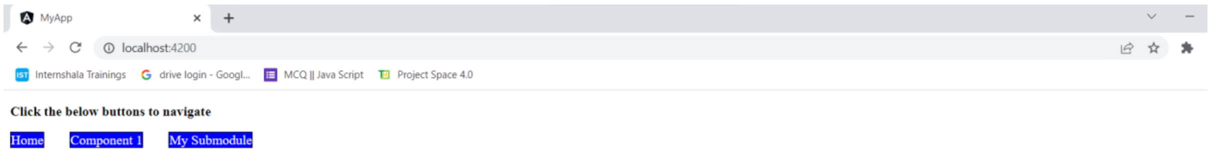
```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import {Component1Component} from'./component1/component1.component'
import {MySubmoduleRoutingModule} from './my-submodule/my-
submodule.module’
const routes: Routes = [
  { path: 'component1', component: Component1Component },
  { path: 'my-submodule', loadChildren: () =>
MySubmoduleRoutingModule },
];

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

```
export class AppRoutingModule {}
```

App.component.ts

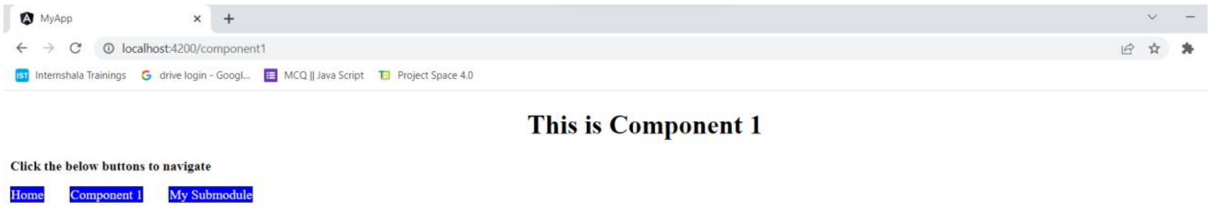
```
<router-outlet></router-outlet>
<p><b>Click the below buttons to navigate</b></p>
<nav>
  <a routerLink="/">Home</a>
  <a routerLink="/component1">Component 1</a>
  <a routerLink="/my-submodule">My Submodule</a>
</nav>
```

Output:



Click the below buttons to navigate

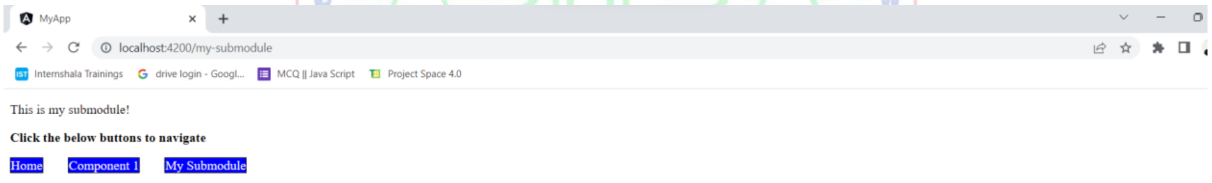
Home Component 1 My Submodule



This is Component 1

Click the below buttons to navigate

Home Component 1 My Submodule



This is my submodule!

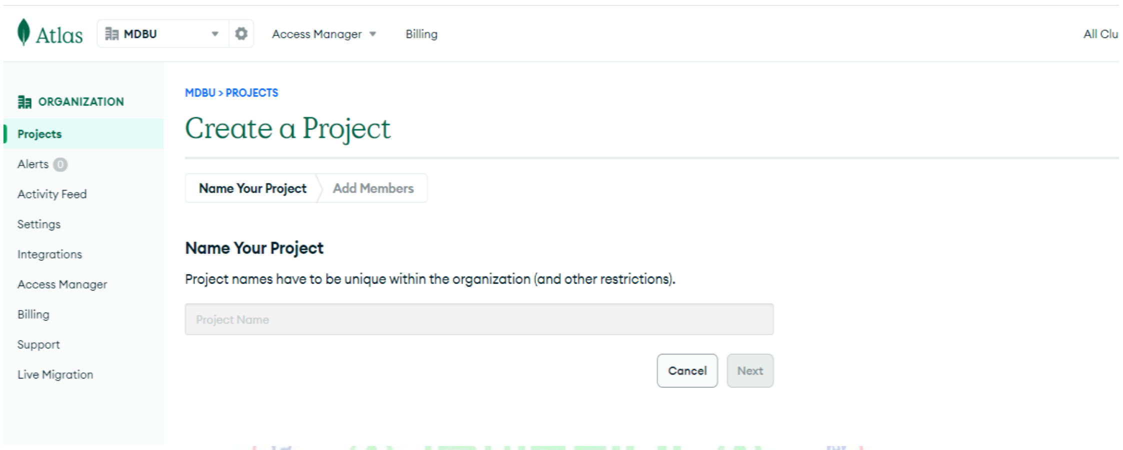
Click the below buttons to navigate

Home Component 1 My Submodule

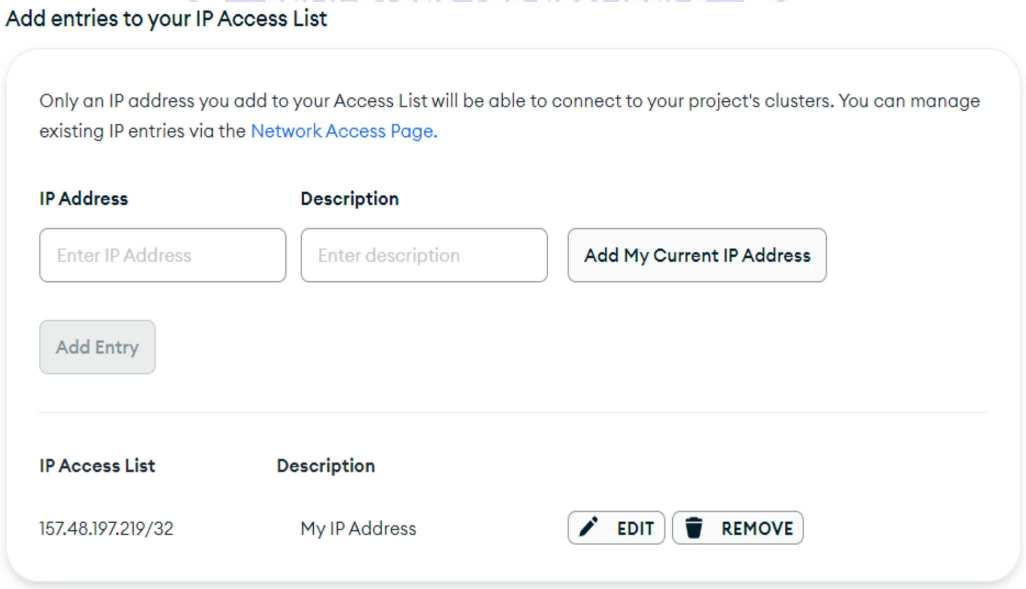
11.a Course Name: MongoDB Essentials - A Complete MongoDB Guide
Module Name: Installing MongoDB on the local computer, Create MongoDB Atlas Cluster Install MongoDB and configure ATLAS.

Program:
To install MongoDB and configure ATLAS , we need to follow the following steps:

- Sign up for a MongoDB Atlas account on the MongoDB website.
- Create a new project and cluster in Atlas.



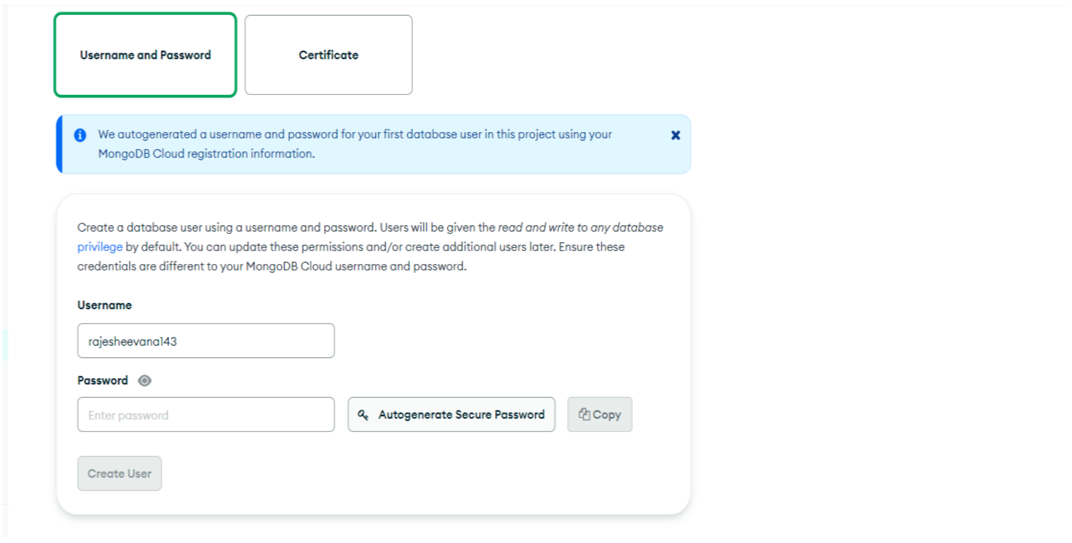
- Whitelist your IP address to allow access to the cluster.


Add entries to your IP Access List

Only an IP address you add to your Access List will be able to connect to your project's clusters. You can manage existing IP entries via the Network Access Page.

IP Access List	Description
157.48.197.219/32	My IP Address

- Create a MongoDB user with appropriate permissions.



Username and Password

Certificate

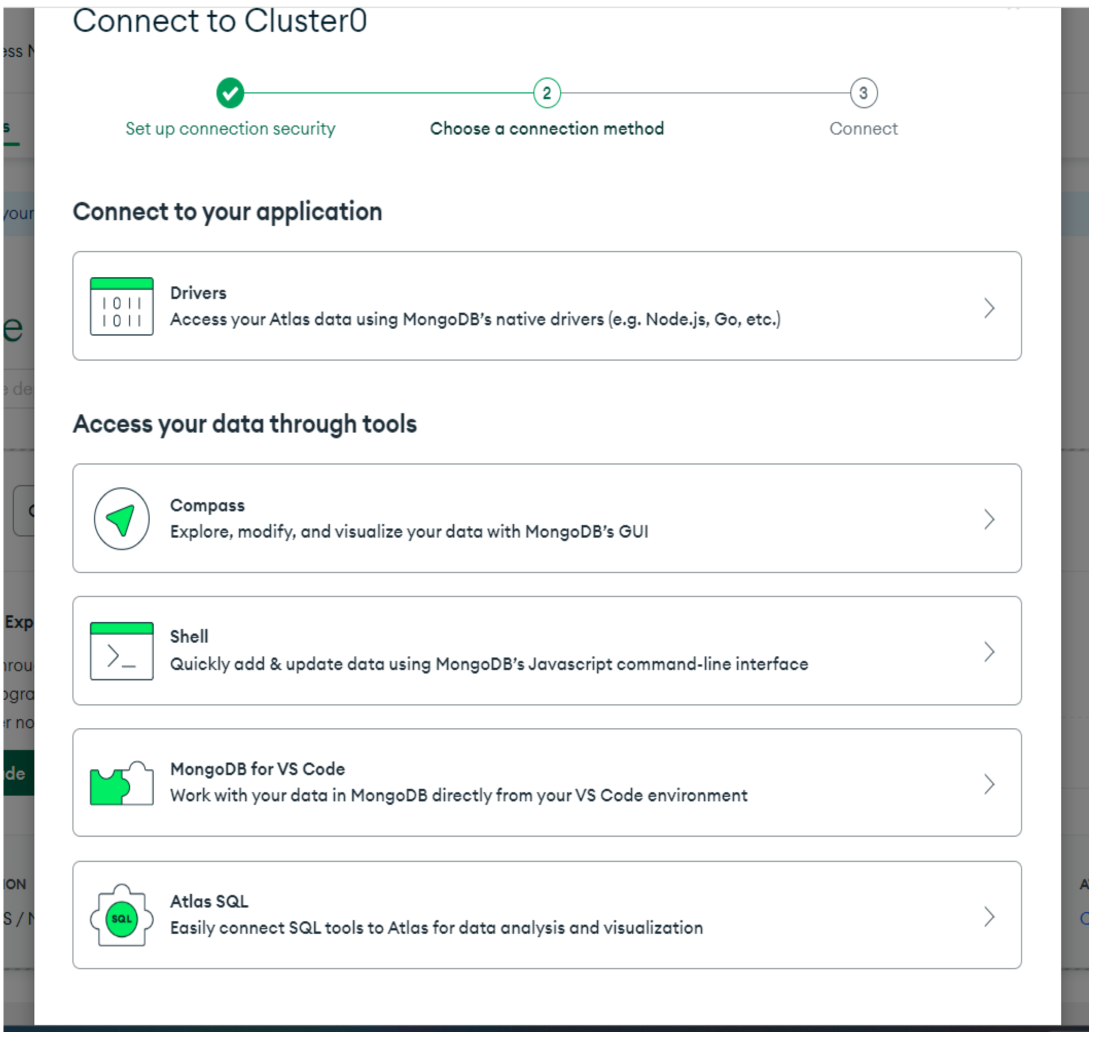
We autogenerated a username and password for your first database user in this project using your MongoDB Cloud registration information.

Create a database user using a username and password. Users will be given the *read and write to any database* privilege by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password.

Username: rajesheevana143

Password: Enter password

- Obtain the connection string for your cluster.



Connect to Cluster0

Connect to your application

Drivers
Access your Atlas data using MongoDB's native drivers (e.g. Node.js, Go, etc.)

Access your data through tools

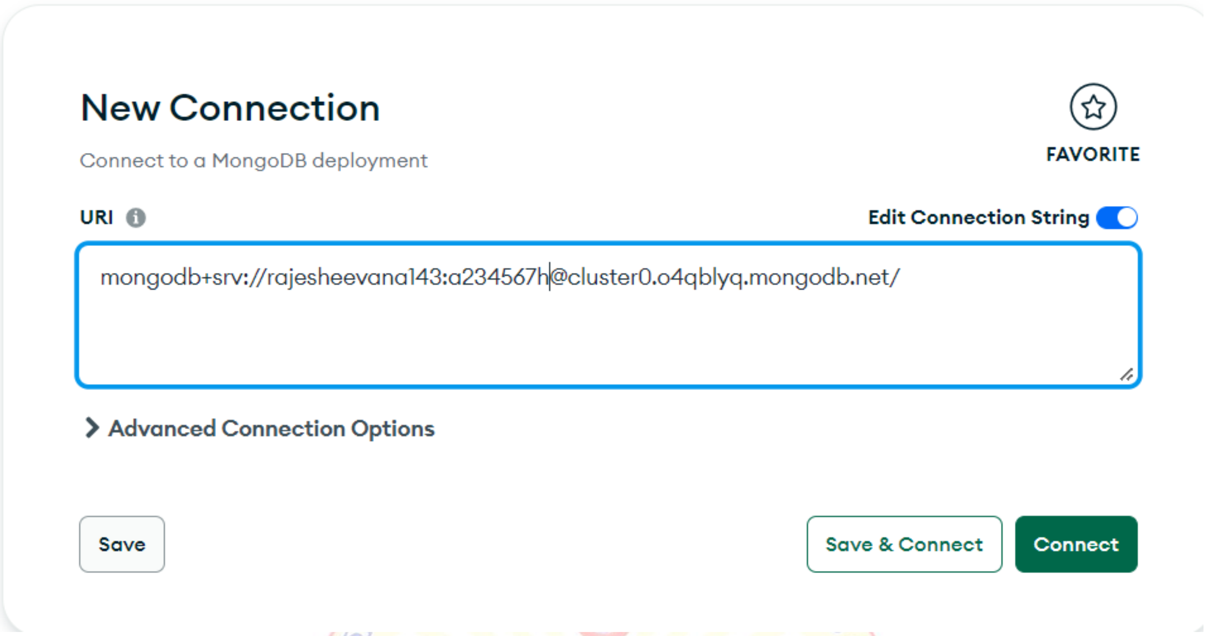
Compass
Explore, modify, and visualize your data with MongoDB's GUI

Shell
Quickly add & update data using MongoDB's Javascript command-line interface

MongoDB for VS Code
Work with your data in MongoDB directly from your VS Code environment

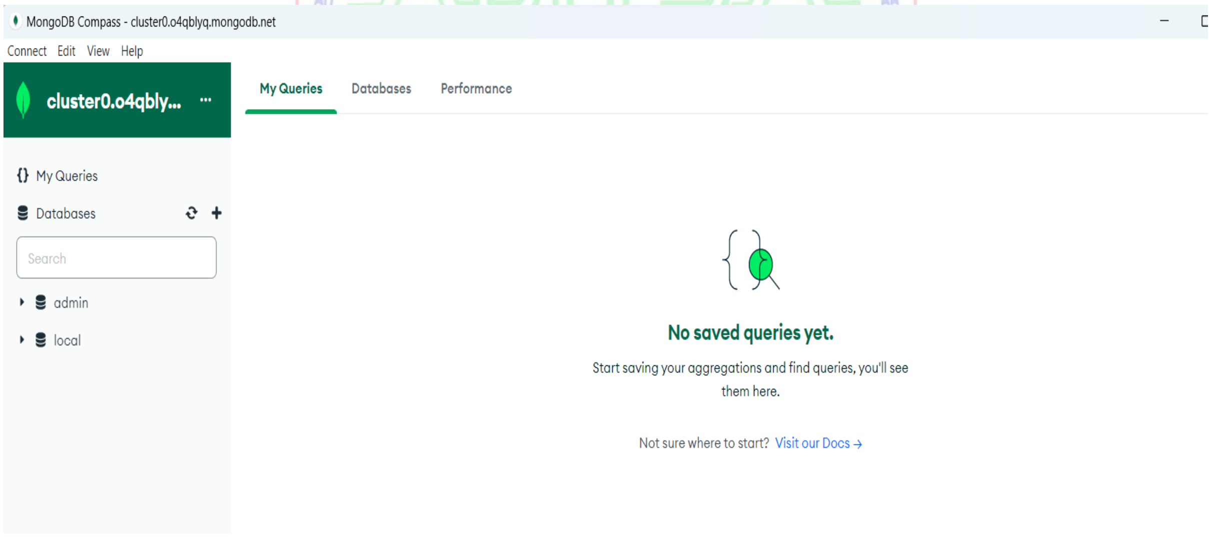
Atlas SQL
Easily connect SQL tools to Atlas for data analysis and visualization

- Install the MongoDB CLI.
- Configure the MongoDB CLI with your Atlas API keys.
- Connect to your Atlas cluster using the MongoDB CLI and the connection string.



mongodb+srv://rajesheevana143:a234567h@cluster0.o4qblyq.mongodb.net/

After successful completion of above process we get the screen as below:



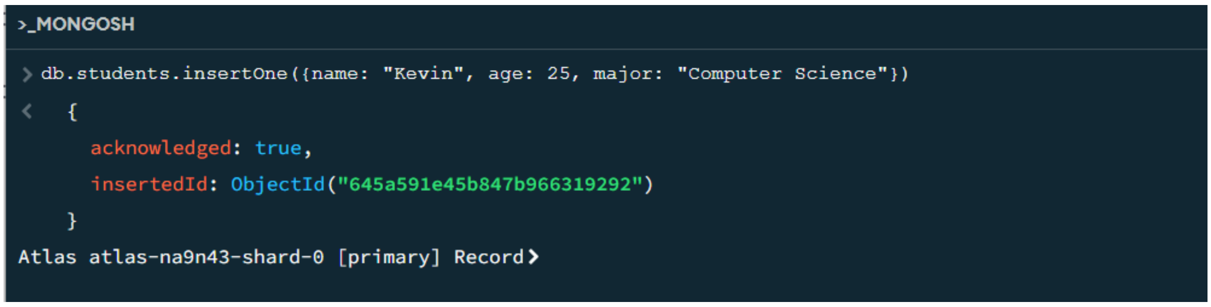
Start saving your aggregations and find queries, you'll see them here.

Not sure where to start?

11.b Course Name: MongoDB Essentials - A Complete MongoDB Guide

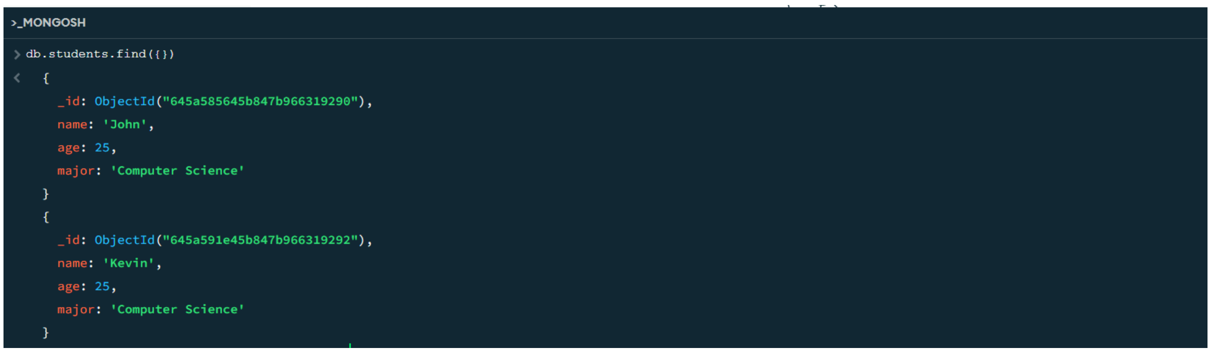
Module Name: Introduction to the CRUD Operations
Write MongoDB queries to perform CRUD operations on document using insert(), find(), update(), remove()
Description:
MongoDB is a document-oriented NoSQL database that provides several methods to manipulate data. Here is a brief description of some of the most commonly used commands in MongoDB:
insert(): The insert() method is used to insert one or more documents into a collection. It takes an array of documents as input and returns a document containing the status of all inserts. If you insert a document in the collection without an _id field, then MongoDB will automatically add an _id field and assign it with a unique ObjectId. If you insert a document with an _id field, then the value of the _id field must be unique to avoid conflicts.
find(): The find() method is used to retrieve documents from a collection. It takes a query object as input and returns a cursor to the documents that match the query. You can use various operators to construct the query object, such as $eq, $gt, $lt, $in, $and, $or, etc.
update(): The update() method is used to modify one or more documents in a collection. It takes two objects as input: a query object to select the documents to update, and an update object to specify the modifications. You can use various update operators to modify the documents, such as $set, $unset, $inc, $push, $pull, etc.
remove(): The remove() method is used to delete one or more documents from a collection. It takes a query object as input to select the documents to remove. If you want to remove all documents from a collection, you can pass an empty query object {} .

INSERT( ):
Command:
```
db.students.insertOne({name: "John", age: 25, major: "Computer Science"})
```

output:

```
>_MONGOSH
> db.students.insertOne({name: "Kevin", age: 25, major: "Computer Science"})
<   {
      acknowledged: true,
      insertedId: ObjectId("645a591e45b847b966319292")
    }
Atlas atlas-na9n43-shard-0 [primary] Record>
```
FIND( ):

Command: db.students.find({})

Output:

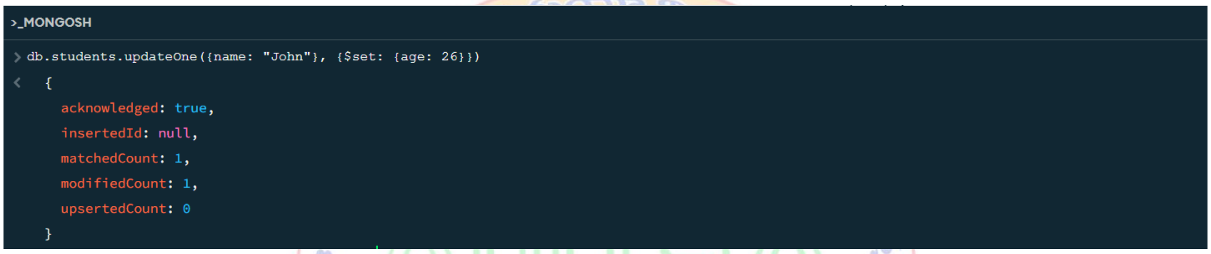


```
>_MONGOSH
> db.students.find({})
< {
    _id: ObjectId("645a585645b847b966319290"),
    name: 'John',
    age: 25,
    major: 'Computer Science'
  }
  {
    _id: ObjectId("645a591e45b847b966319292"),
    name: 'Kevin',
    age: 25,
    major: 'Computer Science'
  }
```

UPDATE( ):

Command: db.students.updateOne({name: "John"}, {$set: {age: 26}})

Output:

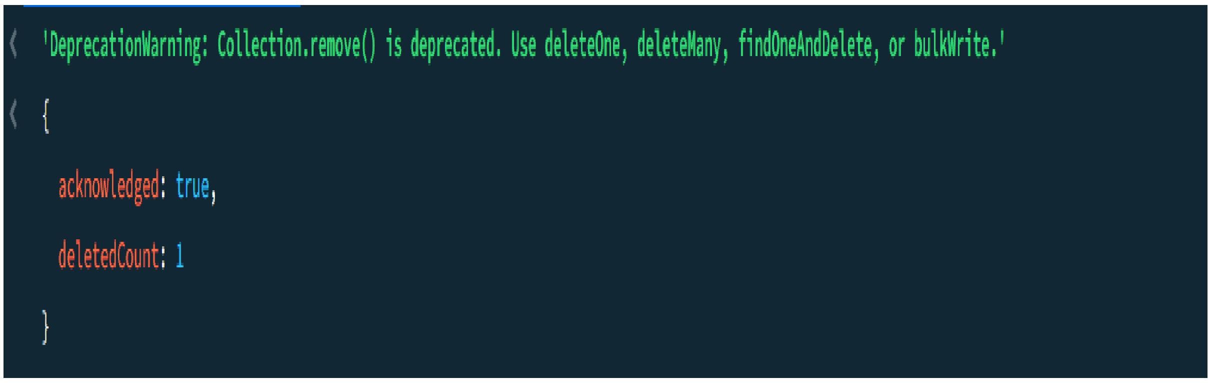


```
>_MONGOSH
> db.students.updateOne({name: "John"}, {$set: {age: 26}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

REMOVE( ):

Command: db.students.remove({name:"John"})

Output:



```
< 'DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDelete, or bulkWrite.'
< {
    acknowledged: true,
    deletedCount: 1
  }
```

12.a Course Name: MongoDB Essentials - A Complete MongoDB Guide Module Name: Create and Delete Databases and Collections Write MongoDB queries to Create and drop databases and collections.

Description:

To create and delete databases and collections in MongoDB, follow these steps:

Create a database: To create a new database in MongoDB, use the use <db> statement in the mongosh shell. For example, to create a new database named "myDB", use the following command: use myDB. If the database does not exist, MongoDB will create the database when you first store data for that database

Create a collection: To create a new collection in MongoDB, you can either create collections implicitly or explicitly. To create a collection explicitly, use the createCollection() method followed by the name of the collection. For example, to create a new collection named "students", use the following command: db.createCollection("students"). If you insert a document into a collection that does not exist yet, MongoDB will automatically create the collection

Delete a database: To delete a database in MongoDB, use the dropDatabase() method followed by the name of the database. For example, to delete the "myDB" database, use the following command: db.dropDatabase()

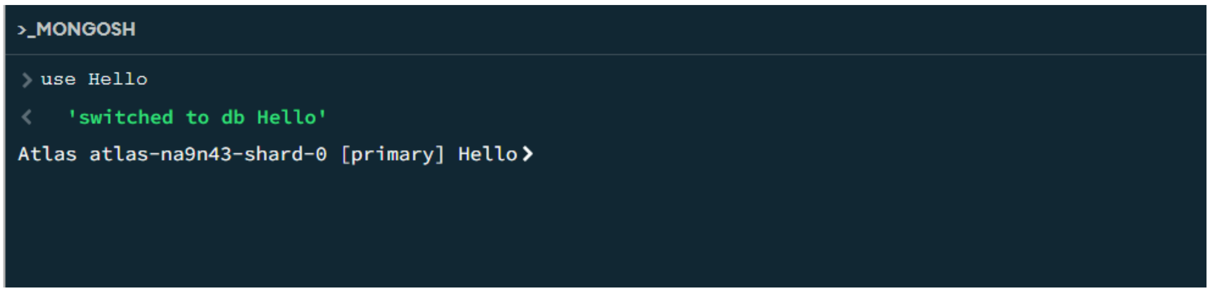
Delete a collection: To delete a collection in MongoDB, use the drop() method followed by the name of the collection. For example, to delete the "students" collection, use the following command: db.students.drop()

Program:

1.Create Database:

Command: use Students

Output:

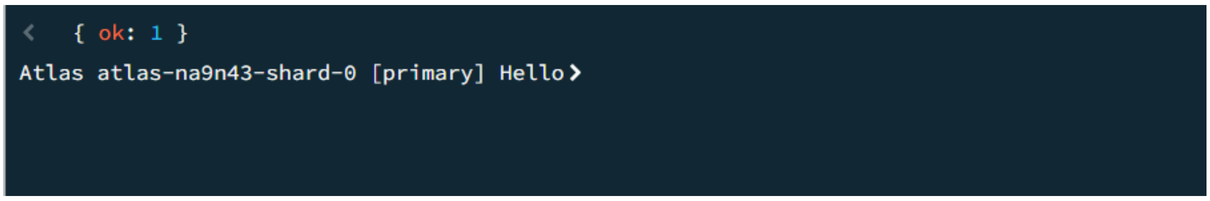


```
>_MONGOSH
> use Hello
<   'switched to db Hello'
Atlas atlas-na9n43-shard-0 [primary] Hello >
```

2.Create Collection:

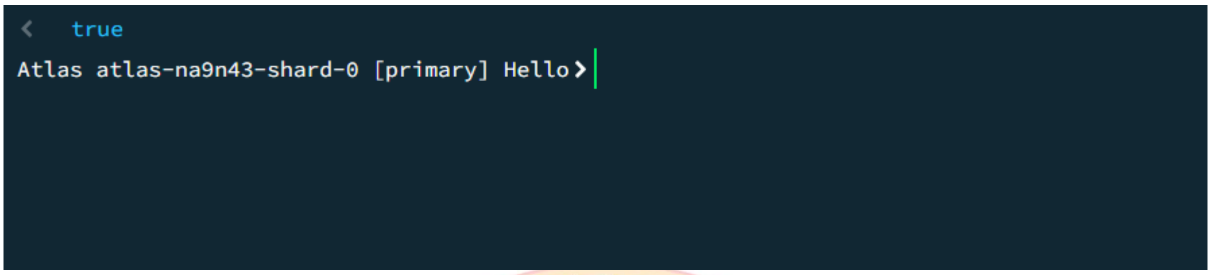
Command:db.createCollection(“students”);

Output:



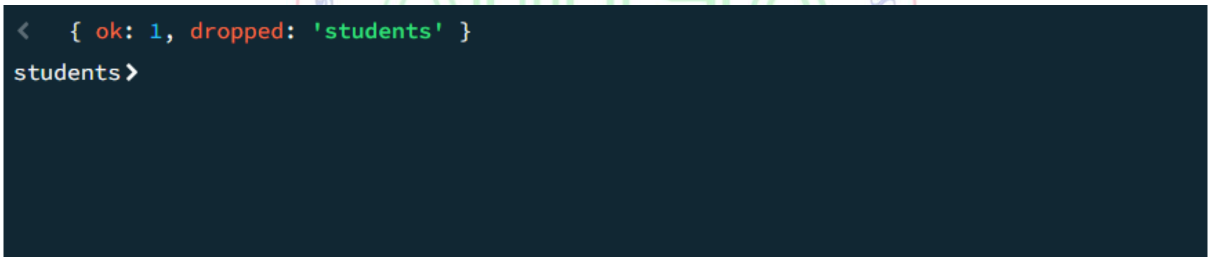
```
<   { ok: 1 }
Atlas atlas-na9n43-shard-0 [primary] Hello>
```

3.Delete Collection:
Command:db.students.drop( )
Output:



```
<   true
Atlas atlas-na9n43-shard-0 [primary] Hello>
```

4.Delete Database:
Command:db.dropDatabase( );
Output:



```
<   { ok: 1, dropped: 'students' }
students>
```

12.b Course Name: MongoDB Essentials - A Complete MongoDB Guide Module Name: Introduction to MongoDB Queries Write MongoDB queries to work with records using find(), limit(), sort(), createIndex(), aggregate().

Description:

find(): The find() method is used to retrieve documents from a collection that match a specified query. The query can include various query operators to specify the criteria. For example, to find all documents in the "students" collection where the age is greater than 20, use the following command: db.students.find({age: {$gt: 20}}) .

limit(): The limit() method is used to limit the number of documents returned by a query. For example, to find the first 5 documents in the "students" collection, use the following command: db.students.find().limit(5) .

sort(): The sort() method is used to sort the documents returned by a query. For example, to find all documents in the "students" collection sorted by age in ascending order, use the following command: db.students.find().sort({age: 1}) .

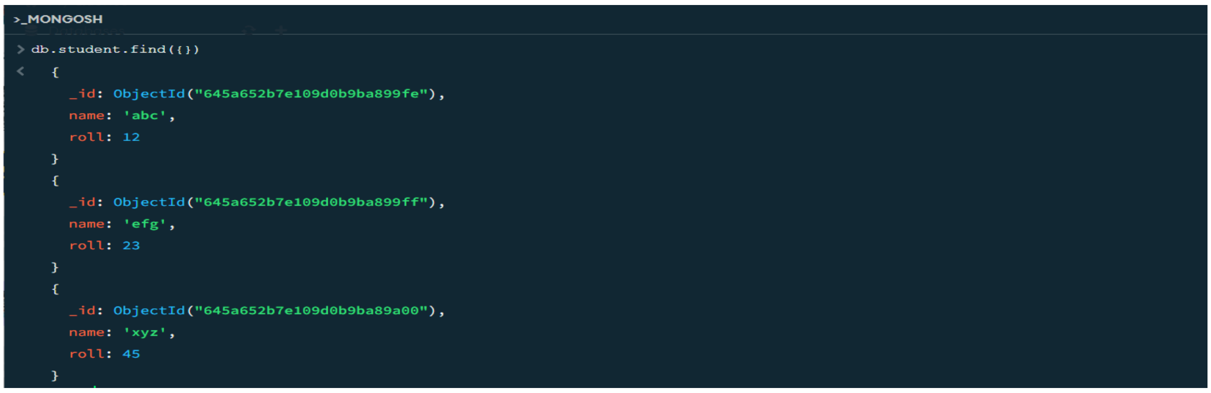
createIndex(): The createIndex() method is used to create an index on a collection to improve query performance. For example, to create an index on the "age" field in the "students" collection, use the following command: db.students.createIndex({age: 1}) **.**

aggregate(): The aggregate() method is used to perform aggregation operations on a collection, such as grouping, counting, and averaging. For example, to group the documents in the "students" collection by major and count the number of documents in each group, use the following command: db.students.aggregate([{$group: {_id: "$major", count: {$sum: 1}}}]).

FIND()

Command:db.student.find({})

Output:

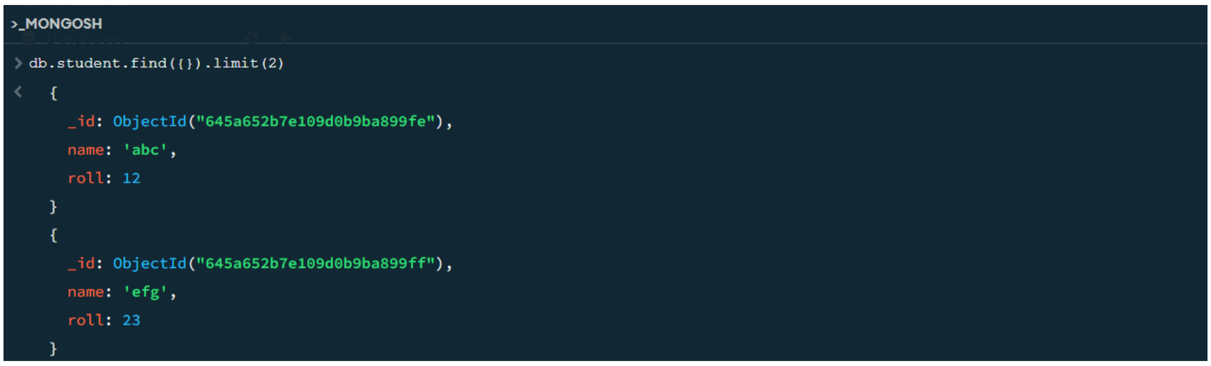


```
>_MONGOSH
> db.student.find({})
<   {
      _id: ObjectId("645a652b7e109d0b9ba899fe"),
      name: 'abc',
      roll: 12
    }
    {
      _id: ObjectId("645a652b7e109d0b9ba899ff"),
      name: 'efg',
      roll: 23
    }
    {
      _id: ObjectId("645a652b7e109d0b9ba89a00"),
      name: 'xyz',
      roll: 45
    }
```

LIMIT()

Command:db.student.find().limit(2)

Output:

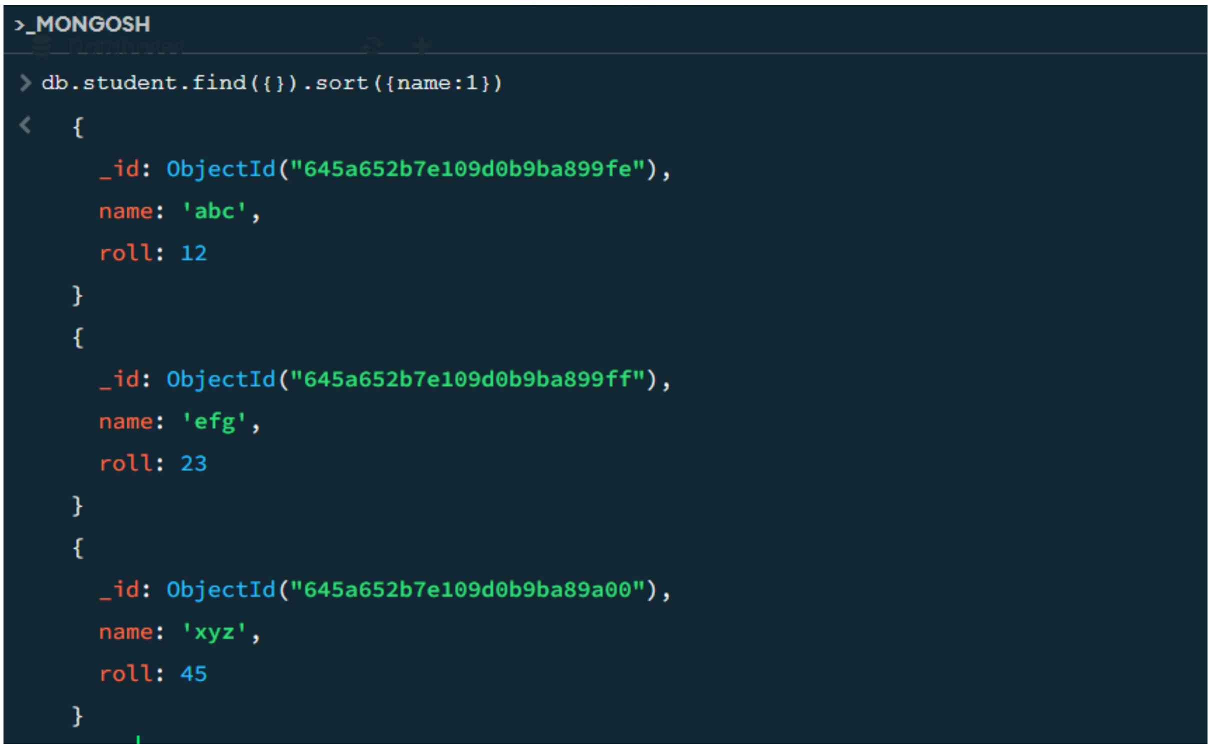


```
>_MONGOSH
> db.student.find({}).limit(2)
<   {
      _id: ObjectId("645a652b7e109d0b9ba899fe"),
      name: 'abc',
      roll: 12
    }
    {
      _id: ObjectId("645a652b7e109d0b9ba899ff"),
      name: 'efg',
      roll: 23
    }
```

SORT()

Command:db.student.find(). Sort({name:1})

Output:

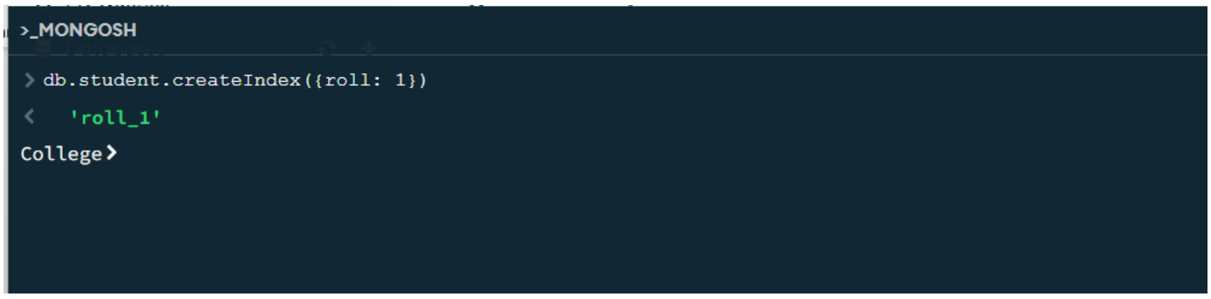


```
>_MONGOSH
> db.student.find({}).sort({name:1})
<   {
      _id: ObjectId("645a652b7e109d0b9ba899fe"),
      name: 'abc',
      roll: 12
    }
    {
      _id: ObjectId("645a652b7e109d0b9ba899ff"),
      name: 'efg',
      roll: 23
    }
    {
      _id: ObjectId("645a652b7e109d0b9ba89a00"),
      name: 'xyz',
      roll: 45
    }
```

CREATEINDEX()

Command:db.createIndex({roll:1});

Output:

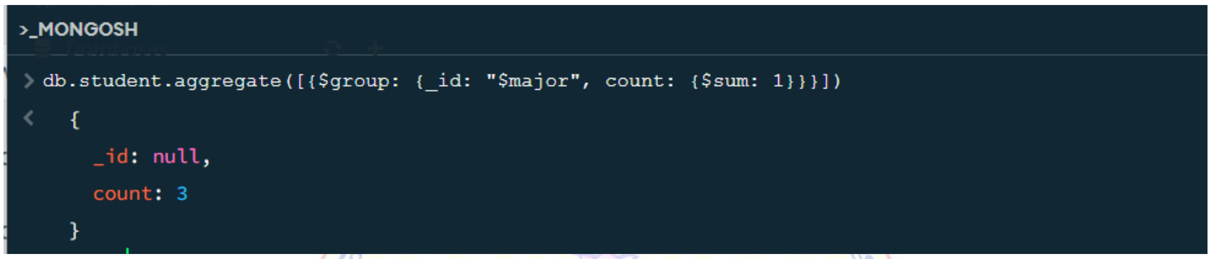


```
>_MONGOSH
> db.student.createIndex({roll: 1})
<   'roll_1'
College>
```

AGGREGATE

Command:db.student.aggregate([{$group:{_id:"$major",count:{$sum:1}}}]);

Output:



```
>_MONGOSH
> db.student.aggregate([{$group: {_id: "$major", count: {$sum: 1}}}])
<   {
      _id: null,
      count: 3
    }
```