Experiment-1: Course Name: Angular JS
1-a) 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.

Aim: 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.

Description:
To develop an application using Angular on a local system, we need to set up a development environment that includes the installation of:
- Node.js (^12.20.2 || ^14.15.5 || ^16.10.0) and npm (min version required 6.13.4)
- Angular CLI
- Visual Studio Code

First of all, we need to Install Node.js and Visual Studio Code from their respective official websites.

Installation of Node.js:
Step-1: Downloading the Node.js '.msi' installer
The first step to install Node.js on windows is to download the installer. Visit the official Node.js website i.e; https://nodejs.org/en/download/ and download the .msi file according to your system environment (32-bit & 64-bit). An MSI installer will be downloaded on your system.
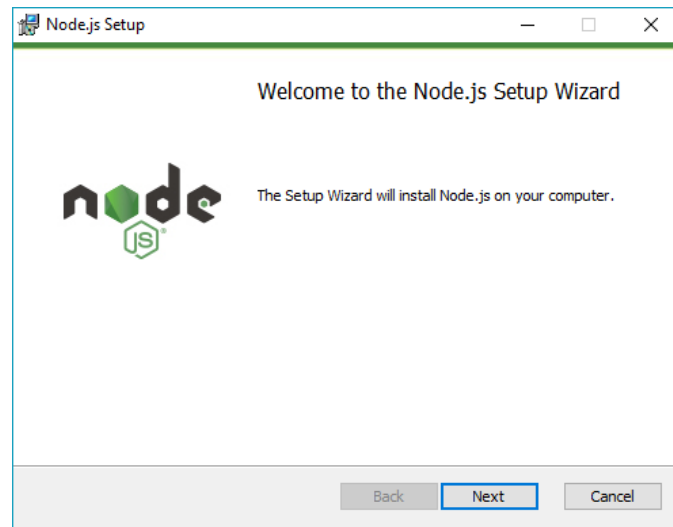

Step-2: Running the Node.js installer
Now you need to install the node.js installer on your PC. You need to follow the following steps for the Node.js to be installed:
- Double click on the .msi installer.
- The Node.js Setup wizard will open.
- Welcome To Node.js Setup Wizard. Select "Next"

- After clicking "Next", End-User License Agreement (EULA) will open.
- Check "I accept the terms in the License Agreement". Select "Next".
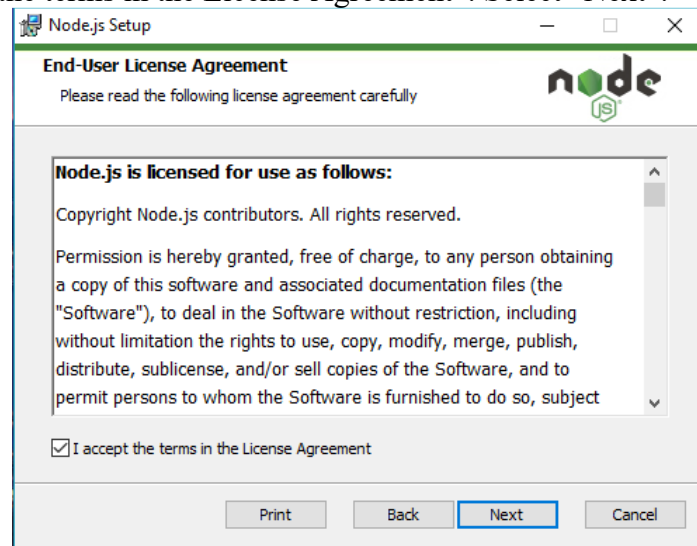
- Destination Folder. Set the Destination Folder where you want to install Node.js & Select "Next".
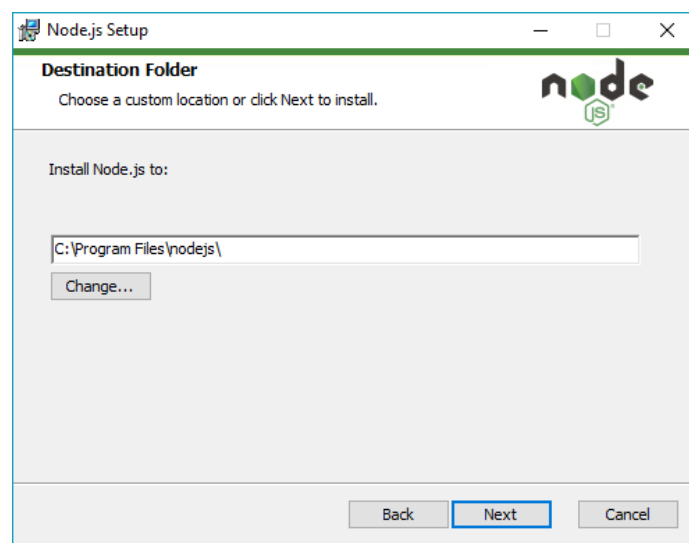
- Custom Setup. Select "Next"
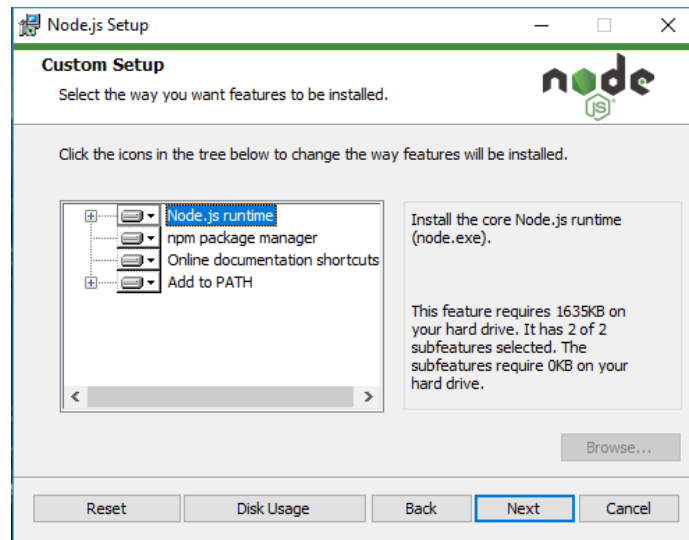
- Ready to Install Node.js. Select "Install".
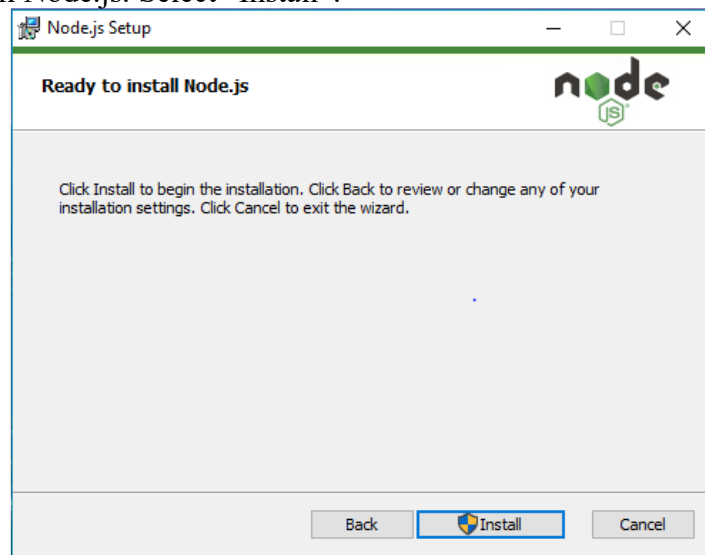
- Complete the Node.js Setup Wizard. Click "Finish".
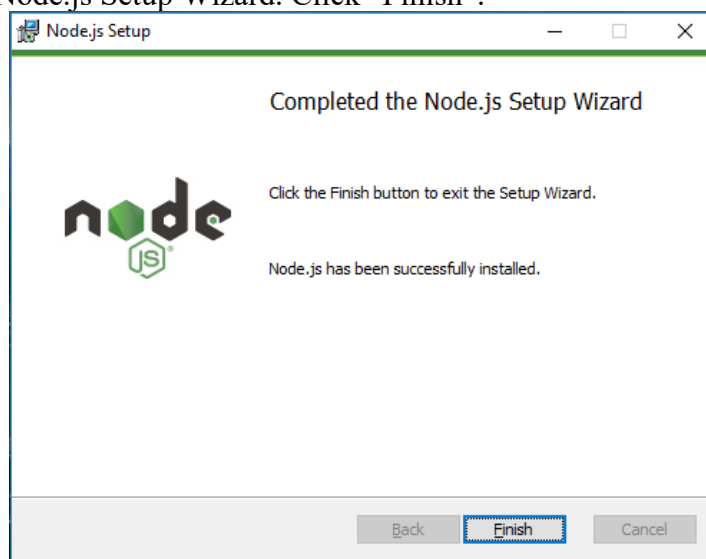
Step 3: Verify that Node.js was properly installed or not.

To check that node.js and npm were completely installed on your system or not, you can run the following commands in your command prompt or Windows Powershell and test it:-

C:\Users\Admin> node –v
C:\Users\Admin>npm -v

Installation of VS Code:
Step 1: Visit the official website of the Visual Studio Code using any web browser like Google Chrome, Microsoft Edge, etc.
Step 2: Press the "Download for Windows" button on the website to start the download of the Visual Studio Code Application.
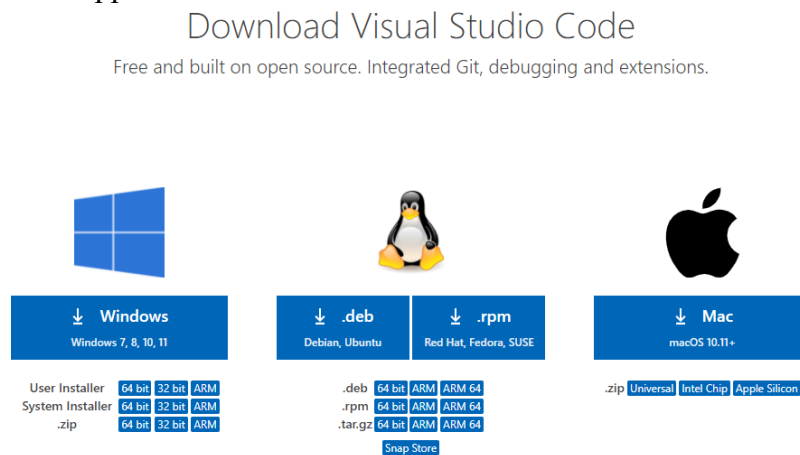
Step 3: When the download finishes, then the Visual Studio Code icon appears in the downloads folder.
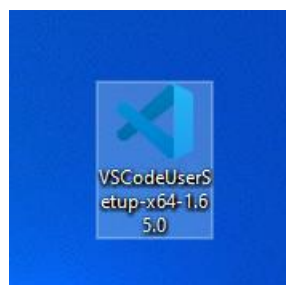
Step 4: Click on the installer icon to start the installation process of the Visual Studio Code.
Step 5: After the Installer opens, it will ask you for accepting the terms and conditions of the Visual Studio Code. Click on **I accept the agreement** and then click the **Next** button.
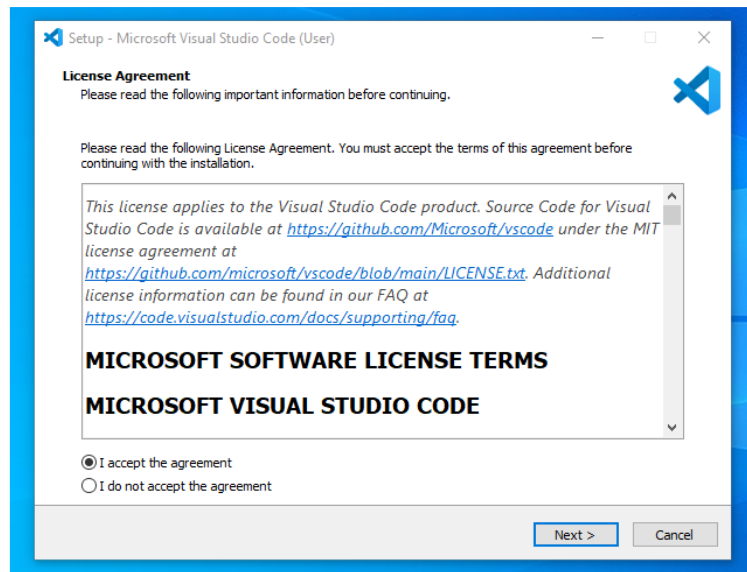
Step 6: Choose the location data for running the Visual Studio Code. It will then ask you for browsing the location. Then click on **Next** button.
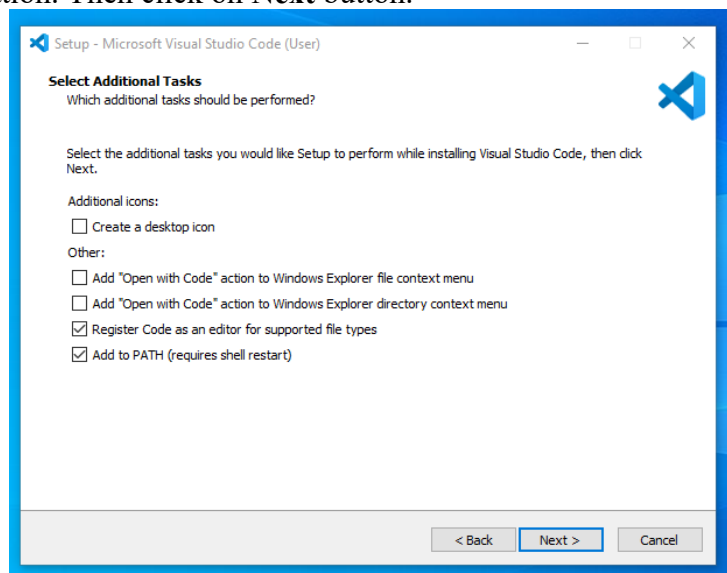
Step 7: Then it will ask for beginning the installing setup. Click on the **Install** button.
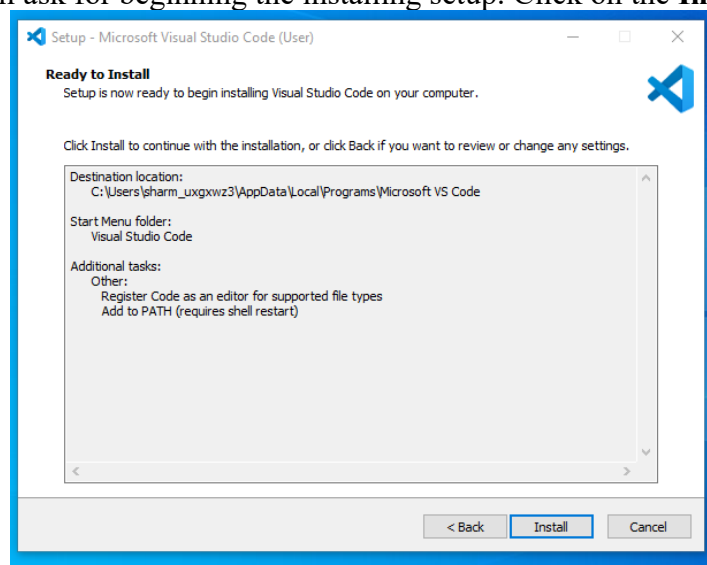
Step 8: After clicking on Install, it will take about 1 minute to install the Visual Studio Code on your device.

Step 9: After the Installation setup for Visual Studio Code is finished, it will show a window like this below. Tick the "**Launch Visual Studio Code**" checkbox and then click **Next**.

Step 10: After the previous step, the **Visual Studio Code window** opens successfully. Now you can create a new file in the Visual Studio Code window and choose a language of yours to begin your programming journey!

Steps to install Angular CLI:

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

<div align="center">npm install -g @angular/cli</div>

Test successful installation of Angular CLI using the following command **Note:** Sometimes additional dependencies might throw an error during CLI installation but still check whether CLI is installed or not using the following command. If the version gets displayed, you can ignore the errors.

```
C:\Windows\system32\cmd.exe

Microsoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.

C:\Users\harsh>ng v


                    _                      ____ _     ___
     /\   _ __   __ _ _   _| | __ _ _ __    / ___| |   |_ _|
    /  \ | '_ \ / _` | | | | |/ _` | '__|  | |   | |    | |
   / /\ \| | | | (_| | |_| | | (_| | |     | |___| |___ | |
  /_/  \_\_| |_|\__, |\__,_|_|\__,_|_|      \____|_____|___|
               |___/


Angular CLI: 17.1.1
Node: 20.11.0
Package Manager: npm 10.4.0
OS: win32 x64

Angular:
...

Package                      Version
------------------------------------------------------
@angular-devkit/architect    0.1701.1 (cli-only)
@angular-devkit/core         17.1.1 (cli-only)
@angular-devkit/schematics   17.1.1 (cli-only)
@schematics/angular          17.1.1 (cli-only)


C:\Users\harsh>
```

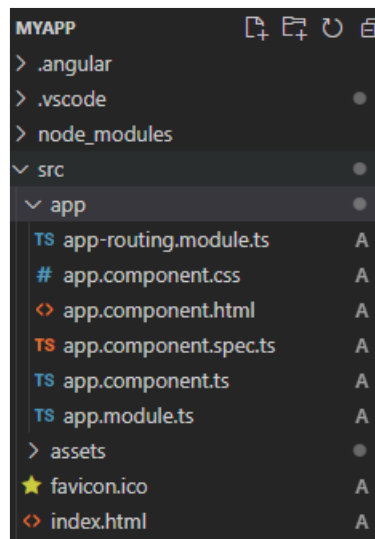**Creation of first Angular Js application:** Create an application with the name 'MyApp' using the following CLI command **ng new filename** The above command will display two questions. The first question is as shown below screen short. Typing 'y' will create a routing module file (app-routing.module.ts). The next question is to select the stylesheet to use in the application. Select CSS and press Enter as shown below:

```
C:\Users\admin>ng new MyApp
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE MyApp/angular.json (2696 bytes)
CREATE MyApp/package.json (1037 bytes)
CREATE MyApp/README.md (1059 bytes)
CREATE MyApp/tsconfig.json (901 bytes)
CREATE MyApp/.editorconfig (274 bytes)
CREATE MyApp/.gitignore (548 bytes)
CREATE MyApp/tsconfig.app.json (263 bytes)
CREATE MyApp/tsconfig.spec.json (273 bytes)
CREATE MyApp/.vscode/extensions.json (130 bytes)
CREATE MyApp/.vscode/launch.json (474 bytes)
CREATE MyApp/.vscode/tasks.json (938 bytes)
CREATE MyApp/src/favicon.ico (948 bytes)
CREATE MyApp/src/index.html (291 bytes)
CREATE MyApp/src/main.ts (214 bytes)
CREATE MyApp/src/styles.css (80 bytes)
CREATE MyApp/src/assets/.gitkeep (0 bytes)
CREATE MyApp/src/app/app-routing.module.ts (245 bytes)
CREATE MyApp/src/app/app.module.ts (393 bytes)
CREATE MyApp/src/app/app.component.html (23115 bytes)
CREATE MyApp/src/app/app.component.spec.ts (1070 bytes)
CREATE MyApp/src/app/app.component.ts (209 bytes)
CREATE MyApp/src/app/app.component.css (0 bytes)
✓ Packages installed successfully.
```

This will create the following folder structure with the dependencies installed inside the node_modules folder.

```
MYAPP
> .angular
> .vscode
> node_modules
∨ src
  ∨ app
    TS app-routing.module.ts        A
    #  app.component.css            A
    <> app.component.html           A
    TS app.component.spec.ts        A
    TS app.component.ts             A
    TS app.module.ts                A
  > assets
  ★ favicon.ico                     A
  <> index.html                     A
```

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

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS

○ PS F:\SOC-II Lab\AngularApp\myapp> ng serve --o

Initial Chunk Files | Names      | Raw Size
polyfills.js        | polyfills  | 83.46 kB |
main.js             | main       |  2.22 kB |
styles.css          | styles     | 95 bytes |

                    | Initial Total | 85.78 kB

Application bundle generation complete. [5.773 seconds]
Watch mode enabled. Watching for file changes...
Re-optimizing dependencies because vite config has changed
  → Local:   http://localhost:4200/
  → press h + enter to show help
```

- ng serve will build and run the application
- --open option will show the output by opening a browser automatically with the default port.

Use the following command to change the port number if another application is running on the default port(4200)
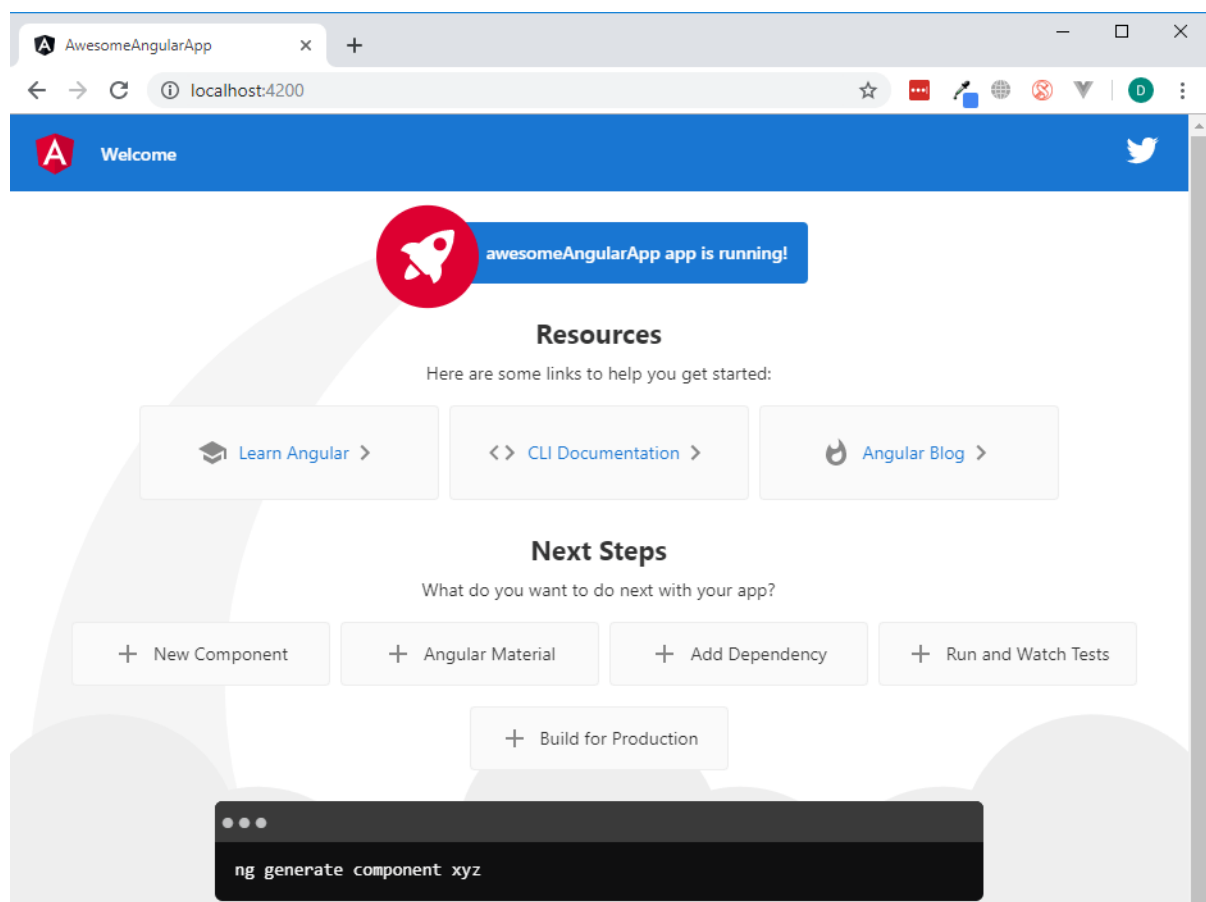
1-b) Module Name: Components and Modules
Create a new component called hello and render Hello Angular on the page.

Aim: Components and Modules. Create a new component called hello and render Hello Angular on the page.
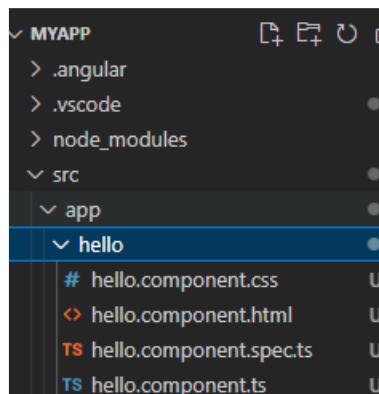
Description:
Component: A component is the basic building block of an Angular application. It emphasizes the separation of concerns and each part of the Angular application can be written independently of one another.
Module: Modules in Angular are used to organize the application. It sets the execution context of an Angular application. A module in Angular is a class with the @NgModule decorator added to it. @NgModule metadata will contain the declarations of components, pipes, directives, services that are to be used across the application.

Program:
In the same **MyApp** application created earlier, create a new component called hello using the following CLI command.

```
C:\Users\admin\MyApp>ng generate component hello
CREATE src/app/hello/hello.component.html (20 bytes)
CREATE src/app/hello/hello.component.spec.ts (592 bytes)
CREATE src/app/hello/hello.component.ts (198 bytes)
CREATE src/app/hello/hello.component.css (0 bytes)
UPDATE src/app/app.module.ts (471 bytes)
```

```
∨ MYAPP                    ⌐₊ ⌐₊ ⟳ ᴱ
  > .angular
  > .vscode                          ●
  > node_modules
  ∨ src                              ●
    ∨ app                            ●
      ∨ hello                        ●
        # hello.component.css        U
        <> hello.component.html      U
        TS hello.component.spec.ts   U
        TS hello.component.ts        U
```

hello.component.ts:
```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './hello.component.html',
  styleUrls: ['./hello.component.css']
})
export class HelloComponent {
  courseName: string="Angular!";
}
```
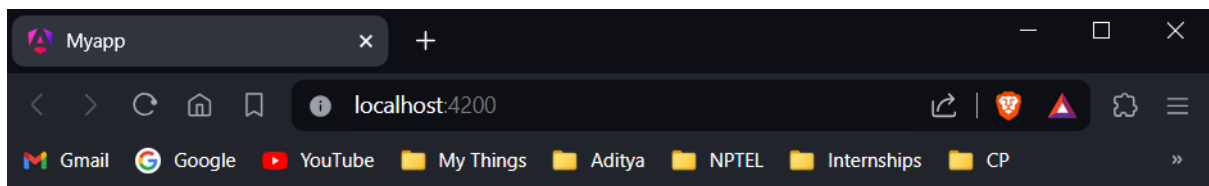**hello.component.html:**
```
<p>Hello, {{courseName}}</p>
```

hello.component.css:
```css
p{
   color:red;
   font-size:20px;
   text-align:center;
}
```

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

Output:

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

Aim: Add an event to the hello component template and when it is clicked, it should change the courseName.

Description: An Events in AngularJS can be used to perform particular tasks, based on the action taken. Both Angular Event & the HTML Event will be executed & will not overwrite with an HTML Event.
To bind to an event, you use the Angular event binding syntax. This syntax consists of a target event name within parentheses to the left of an equal sign, and a quoted template statement to the right.

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

target event name   template statement
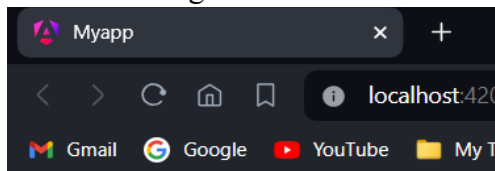
Program:
hello.component.ts:
```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './hello.component.html',
  styleUrls: ['./hello.component.css']
})
export class HelloComponent {
  Message: string="Good Morning!";
  changeName(){
    this.Message="Have a nice day!";
  }
}
```
hello.component.html:
```
<p>{{Message}}</p>
<h2 (click)="changeName()">Click here to change</h2>
```

Output:
Before clicking:

Good Morning!

Click here to change

After Clicking:
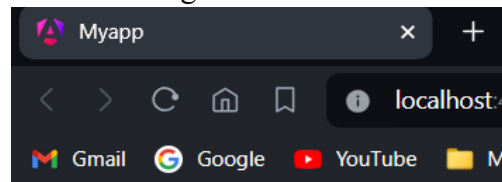
Have a nice day!

Click here to change

1-d) Module Name: Change Detection
Progressively building the PoolCarz Application.

Aim: To progressively build the PoolCarz application.

Description: Change detection is a crucial aspect of maintaining the integrity and performance of AngularJS applications. It ensures that the user interface reflects the latest data and that updates are efficiently propagated throughout the application. AngularJS provides a built-in mechanism for change detection through the "$watch" function. This function allows you to monitor changes in a specific model or expression and execute custom logic when a change is detected.

Program:
hello.component.ts:
```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './hello.component.html',
  styleUrls: ['./hello.component.css']
})
export class HelloComponent {
  Message: string="Good Morning!";
  changeName(){
    this.Message="Have a nice day!";
  }
}
```
hello.component.html:
```
<p>{{Message}}</p>
<h2 (click)="changeName()">Click here to change</h2>
```
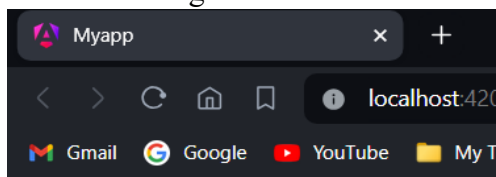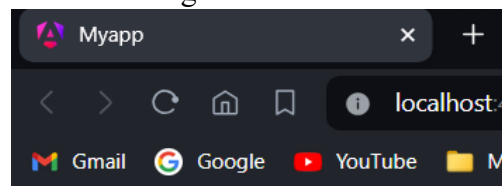
Output:
Before clicking:

After Clicking:

Good Morning!

Click here to change

Have a nice day!

Click here to change

Experiment-2: Course Name: Angular JS
2-a) 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 <<username>>" message otherwise it should render "Invalid Login!!! Please try again..." message.

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

Description: Structural directives are directives which change the DOM layout by adding and removing DOM elements. The ng-if directive removes the HTML element if the expression evaluates to false. If the if statement evaluates to true, a copy of the Element is added in the DOM. The ng-if directive is different from the ng-hide, which hides the display of the element, where the ng-if directive completely removes the element from the DOM.

Program:
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 action="">
    <label for="text">User Name: </label>
    <input type="text" #username /><br /><br />
    <label for="password">Password: </label>
    <input type="password" #password /><br /><br />
  </form>
  <button (click)="onSubmit(username.value,password.value)">Login</button>
</div>
<div *ngIf="submitted">
```

```html
<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:

```typescript
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [
    AppComponent
  ]
})
export class AppModule { }
```
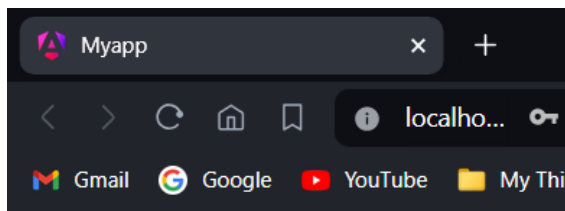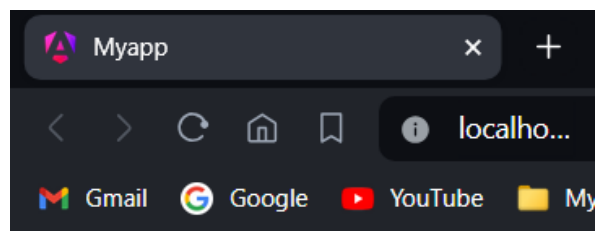
<u>Output:</u>

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

Aim: To create a courses array and rendering it in the template using ngFor directive in a list format.

Description: NgFor is used as a structural directive that renders each element for the given collection each element can be displayed on the page.
Syntax: <li *ngFor='condition'>
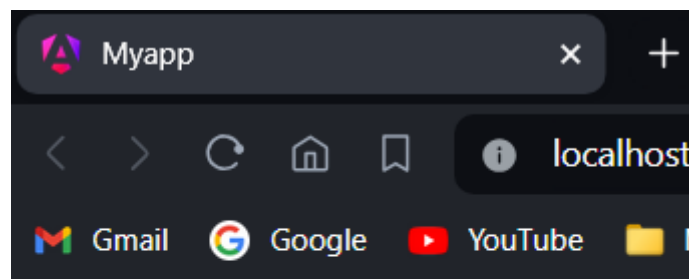
Program:
app.component.ts:
```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  courses: any[] = [
    {id:1,name:"TypeScript"},
    {id:2,name:"Angular"},
    {id:3,name:"Node.js"},
    {id:4,name:"MongoDB"}
  ];
}
```
app.component.html:
```
<ul>
  <li *ngFor="let course of courses; let i=index">
    {{i}}-{{course.name}}
  </li>
</ul>
```

Output:

- 0-TypeScript
- 1-Angular
- 2-Node.js
- 3-MongoDB

2-c) Module Name: ngSwitch
Display the correct option based on the value passed to ngSwitch directive.

Aim: Display the correct option based on the value passed to ngSwitch directive.

Description: The NgSwitch directive specifies an expression to match against. The NgSwitchCase directive defines the expressions to match.
- It renders every view that matches.
- If there are no matches, the view with the NgSwitchDefault directive is rendered.
- Elements outside of any NgSwitchCase or NgSwitchDefault directive but within the NgSwitch statement but are preserved at the location.
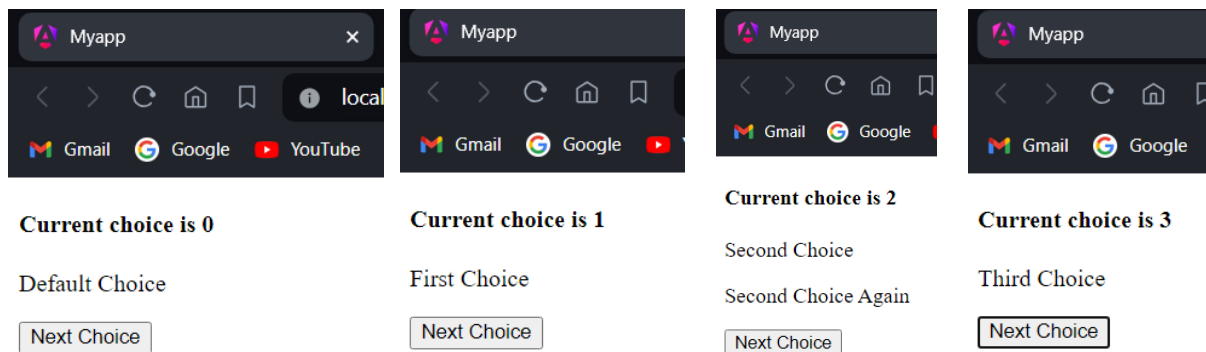
Program:
app.component.ts:
```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  choice=0;
  nextChoice(){
    this.choice++;
  }
}
```
app.component.html:
```
<h4>Current 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>
```

Output:

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

Aim: Create a custom structural directive called 'repeat' which should repeat the element given a number of times.

Description: In AngularJS, custom structural directives can be created to manipulate the DOM (Document Object Model) based on certain conditions. These directives are typically used to repeat or conditionally render HTML elements.
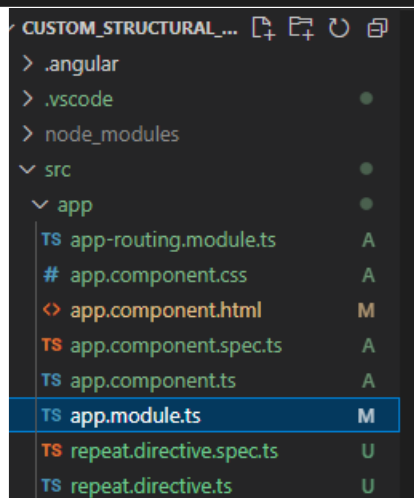Generate a directive called 'repeat' using the following command:

ng generate directive repeat

Program:
app.module.ts:
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { RepeatDirective } from './repeat.directive';
@NgModule({
 declarations: [
   AppComponent,
   RepeatDirective
 ],

```
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  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);
    }
  }
}
```
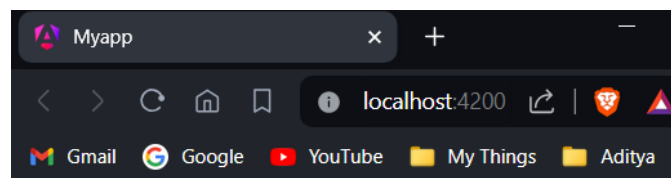
app.component.html:

```
<h1>Custom Structural Directives</h1>
<h2 *appRepeat="10">Angular JS</h2>
```

Output:

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

Aim: To apply multiple CSS properties to a paragraph in a component using ngStyle.

Description: The NgStyle directive lets you set a given DOM elements style properties. ngStyle becomes much more useful when the value is dynamic. The values in the object literal that we assign to ngStyle can be JavaScript expressions which are evaluated and the result of that expression is used as the value of the CSS property.
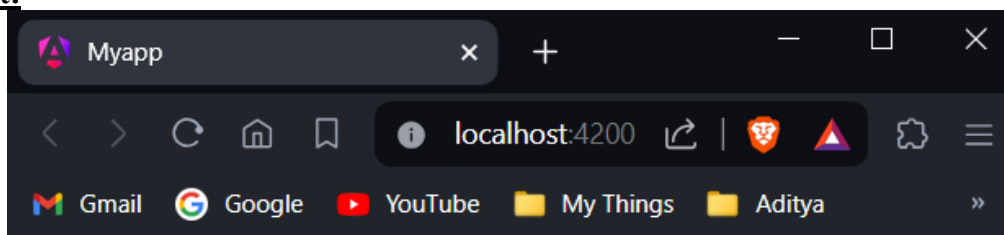
Program:
app.component.ts:
```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title="attribute_directives";
  textColor="green";
  fontWeight="solid";
  borderStyle="solid 2px #ffaa00";
  textAlignment="center";
  fontSize="20px";
}
```
app.component.html:
```
<p [ngStyle]="{color:textColor,borderBottom:borderStyle,'font-weight':fontWeight,'text-align':textAlignment,'font-size':fontSize}">Attribute Directives</p>
```

Output:

3-b) Module Name: ngClass
Apply multiple CSS classes to the text using ngClass directive.

Aim: To apply multiple CSS classes to the text using ngClass directive.

Description: The ngClass directive allows you to dynamically set CSS classes on an HTML element by databinding an expression that represents all classes to be added. The directive won't add duplicate classes if a particular class was already set. When the expression changes, the previously added classes are removed and only then are the new classes added.

Program:
app.component.ts:
```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  isBordered=true;
}
```
app.component.html:
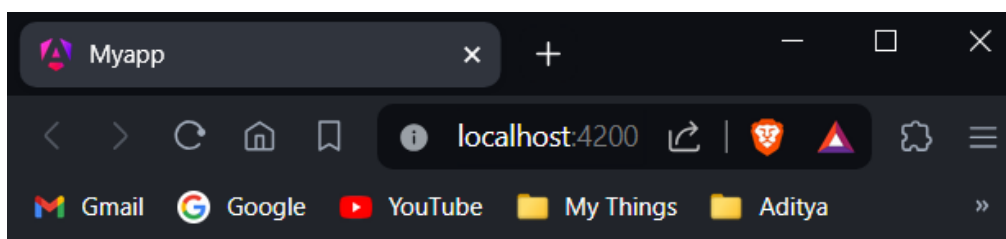```
<h1>Attribute Directives: ngClass</h1>
<h2 [ngClass]="{bordered:isBordered}">Border {{isBordered ? "ON" : "OFF"}}</h2>
```
**app.component.css:**
```
.bordered{
    border: 1px dashed purple;
    background-color: yellow;
}
```

**Output:**

**3-c) 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.**

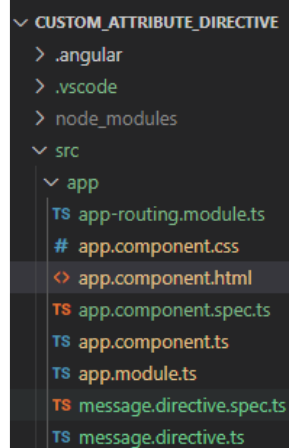**Aim:** To create an attribute directive called 'showMessage' which should display the given message in a paragraph when a user clicks on it and should change the text color to red.

**Description:** Attribute directives helps to change the appearance or behavior of DOM elements and Angular components. Generate a directive called 'message' using the following command:
ng generate directive message





**Program:**
**app.module.ts:**
```
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { MessageDirective } from './message.directive';
@NgModule({
  declarations: [
    AppComponent,
    MessageDirective
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [
    AppComponent
  ]
})
export class AppModule { }
```
**message.directive.ts:**
```
import { Directive, ElementRef, Renderer2, HostListener, Input } from '@angular/core';
```

```
@Directive({
  selector: '[appMessage]'
})
export class MessageDirective {
  @Input('appMessage') message!: string;
  constructor(private el:ElementRef, private renderer:Renderer2){
    renderer.setStyle(el.nativeElement,'cursor','pointer');
  }
  @HostListener('click') onClick(){
    this.el.nativeElement.innerHTML=this.message;
    this.renderer.setStyle(this.el.nativeElement,'color','red');
  }
}
```

app.component.html:

```
<h3>Attribute Directive</h3>
<p [appMessage]="myMessage">Click Here</p>
```

app.component.ts:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  myMessage="Hello, It is my first custom attribute directive named message!";
}
```

app.component.css:

```
h3{
    color: #369;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 250%;
}
p{
    color: #ff0080;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 150%;
}
```

Output:

Experiment-4: Course Name: Angular JS
4-a) Module Name: Property Binding
Binding image with class property using property binding.

Aim: To bind an image with class property using property binding.

Description: Property binding is a one-way mechanism that lets you set the property of a view element. It involves updating the value of a property in the component and binding it to an element in the view template. Property binding uses the [] syntax for data binding.

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="Property Binding";
  imgUrl: string="https://i.ytimg.com/vi/NNS5Piu-EII/hq720.jpg?sqp=-oaymwEhCK4FEIIDSFryq4qpAxMIARUAAAAAGAElAADIQj0AgKJD&rs=AOn4CLCFqzGKbnbnk8Z3Aa9t7jyUZ7jT_w";
}
```
app.component.html:
```
<h2>Property Binding</h2>
<img [src]="imgUrl" width="650px" height="350px" alt="wild">
```

Output:

4-b) Module Name: Attribute Binding
Binding colspan attribute of a table element to the class property.

Aim: Binding colspan attribute of a table element to the class property.

Description: In AngularJS, you can dynamically control the colspan attribute of a table cell using AngularJS directives and expressions. AngularJS allows you to bind data to HTML attributes, including colspan, enabling you to manipulate the table structure based on dynamic data.
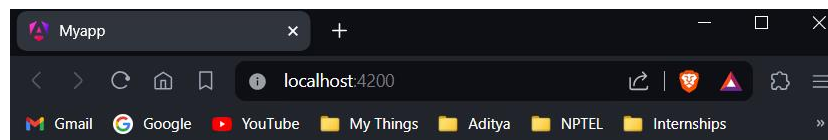
Program:
app.component.ts:
```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title="Property Binding";
  colspanValue="2";
}
```
app.component.html:
```
<h2>Attribute Binding</h2>
<table border="1">
  <tr>
    <td [attr.colspan]="colspanValue">First</td>
    <td>Second</td>
  </tr>
  <tr><td>Third</td><td>Fourth</td><td>Fifth</td></tr>
  <tr><td>Sixth</td><td>Seventh</td><td>Eighth</td></tr>
</table>
```
Output:

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

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

Description: Event Binding is the data binding type is when information flows from the view to the component when an event is triggered. The view sends the data from an event like the click of a button to be used to update the component. It is the exact opposite of property binding, where the data goes from the component to the view.

Style binding is used to set a style of a view element. We can set the inline styles of an HTML element using the style binding in angular. You can also add styles conditionally to an element, hence creating a dynamically styled element.
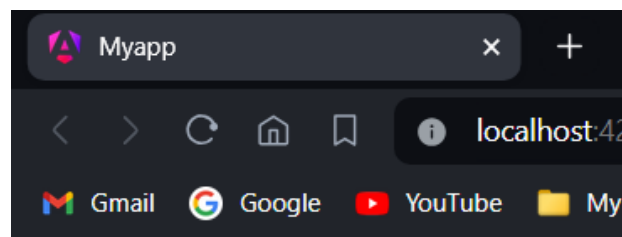
Program:

Style Binding:
app.component.ts:
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  isValid="blue";
  isValid1="13";
}
app.component.html:
<h2>Style Binding</h2>
<button [style.color]="isValid?'blue':'red'">Hello!</button>

<button [style.font-size.px]="isValid1?11:26">Hehe</button>

Output:

If we does not assign values to the isValid and isValid1 properties it will be set to red color and 26px by default because expression will become false. Then the output becomes:
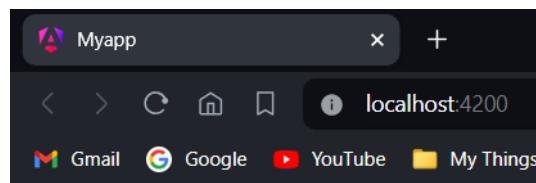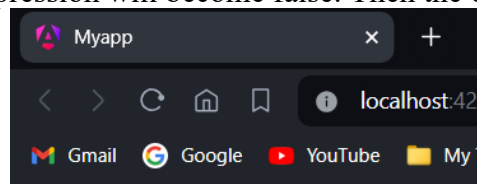

Event Binding:
app.component.ts:
```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  onSubmit(){
    var un=(document.getElementById("uname") as HTMLInputElement).value;
    (document.getElementById("id1") as HTMLInputElement).innerHTML="Your Name is: "+un;
    var p=(document.getElementById("pwd") as HTMLInputElement).value;
    (document.getElementById("id2") as HTMLInputElement).innerHTML="Your Password is: "+p;
  }
}
```
app.component.html:
```
<h2>Event Binding</h2>
<table>
  <tr>
    <td>
      <label for="uname">Enter a User Name: </label>
      <input type="text" size="20px" id="uname" required autocomplete="off" placeholder="Enter your username/id"><br /><br />
    </td>
  </tr>
  <tr>
    <td>
      <label for="pwd">Enter your Password: </label>
      <input type="password" id="pwd" required placeholder="Enter your password">
      <br /><br />
    </td>
  </tr>
  <tr>
    <td>
```
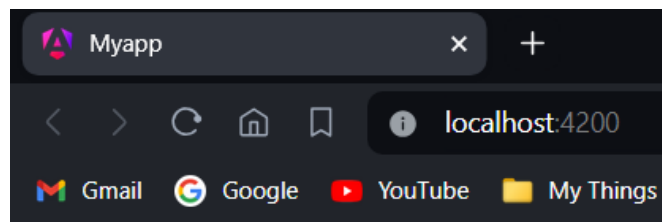
```
    <button (click)="onSubmit()">Login</button>
  </td>
 </tr>
</table>
<div id="id1"></div>
<div id="id2"></div>
```

Output:

Experiment-5: Course Name: Angular JS
5-a) Module Name: Built in Pipes
Display the product code in lowercase and product name in uppercase using built-in pipes.

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

Description: Angular Pipes transform the output. Pipes are simple functions designed to accept an input value, process, and return a transformed value as the output.
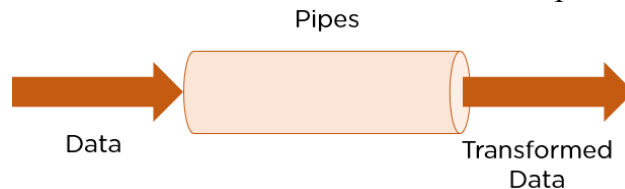
Angular supports many built-in pipes. However, you can also create custom pipes that suit your requirements. Some salient features include:
- Pipes are defined using the pipe "|" symbol.
- Pipes can be chained with other pipes.
- Pipes can be provided with arguments by using the colon (:) sign.

Program:
Generate a new component "pipesexp"

pipesexp.component.ts:
```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './pipesexp.component.html',
  styleUrl: './pipesexp.component.css'
})
export class PipesexpComponent {
  // Experiment-5-a
  productCode="ABC12DEF";
  productName="Vivo X Series";
}
```

pipesexp.component.html:
```
<p>Product Code: {{productCode|lowercase}}</p>
<p>Product Name: {{productName|uppercase}}</p>
```

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

Output:
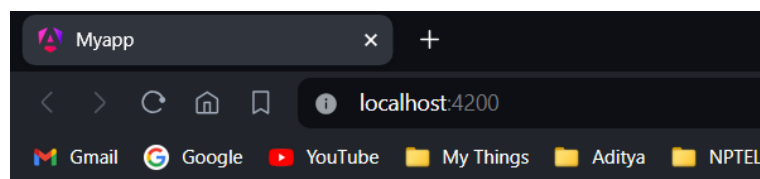
Product Code: abc12def

Product Name: VIVO X SERIES

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

Aim: Apply built-in pipes with parameters to display product details.

Description:
SlicePipe: This returns a slice of an array. The first argument is the start index of the slice and the second argument is the end index.
Syntax: {{ value_expression | slice : start [: end] }}

DatePipe: Formats a date value according to locale rules.
Syntax: {{ value_expression | date [: format [: timezone [: locale]]] }}

Program:
pipesexp.component.ts:
```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './pipesexp.component.html',
  styleUrl: './pipesexp.component.css'
})
export class PipesexpComponent {
  birthday=new Date(2024,3,1);
  productCode="ABC12DEF";
  productName="Vivo X Series";
  newArray=[1,2,3,4,5,6,7,8,9,10];
}
```
pipesexp.component.html:
```
<p>Date is: {{birthday | date:"fullDate" | uppercase}}</p>
<p>Product Code: {{productCode | lowercase}}</p>
<p>Product Name: {{productName | uppercase}}</p>
<p>{{newArray | slice:0:4}}</p>
```

Output:
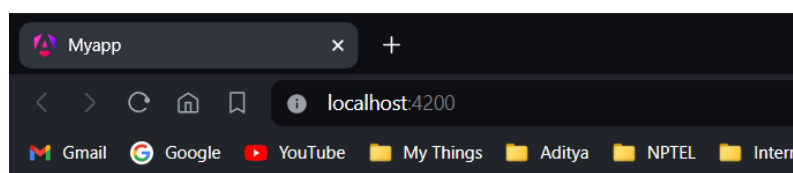
Date is: MONDAY, APRIL 1, 2024

Product Code: abc12def

Product Name: VIVO X SERIES

1,2,3,4

5-c) Module Name: Nested Components Basics
Load CourseslistComponent in the root component when a user clicks on the View courses list button.
Aim: Load CourseslistComponent in the root component when a user clicks on the View courses list button.

Description: In Angular, you can nest components within other components to create complex user interfaces. This hierarchical structure helps break down the UI into smaller, manageable parts, making your code more organized and maintainable. The Angular framework allows us to use a component within another component and when we do so then it is called Angular Nested Components. The outside component is called the parent component and the inner component is called the child component.

Program:
Generate a new component "course-list"

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

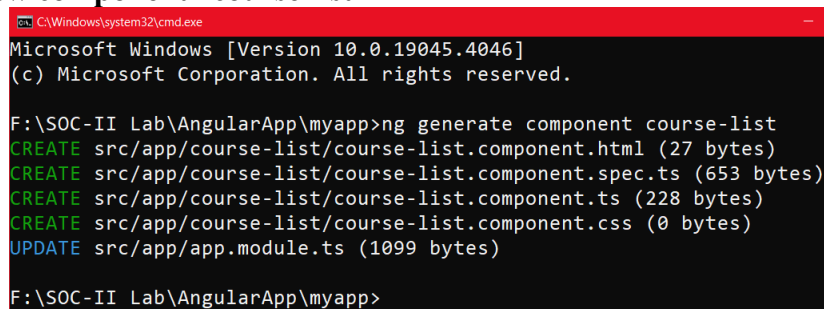**app.component.ts:**
```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  visible:boolean=false;
  showView(){
    this.visible=!this.visible;
  }
}
```

course-list.component.html:
```
<table border="1">
   <thead><tr><th>CourseName</th><th>CourseId</th><th>Course
Price</th></tr></thead>
   <tbody>
      <tr><td>HTML</td><td>1</td><td>1000</td></tr>
      <tr><td>CSS</td><td>2</td><td>2000</td></tr>
      <tr><td>JS</td><td>3</td><td>3000</td></tr>
      <tr><td>Node JS</td><td>4</td><td>4000</td></tr>
      <tr><td>React</td><td>5</td><td>5000</td></tr>
      <tr><td>Angular</td><td>6</td><td>6000</td></tr>
   </tbody>
</table>
```

app.component.html:
```
<button (click)="showView()">Show/Hide</button>
<app-course-list *ngIf="visible"></app-course-list>
```
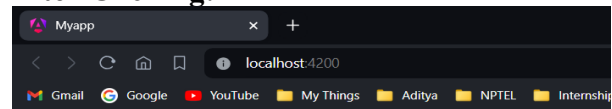
Output:

Before Clicking:
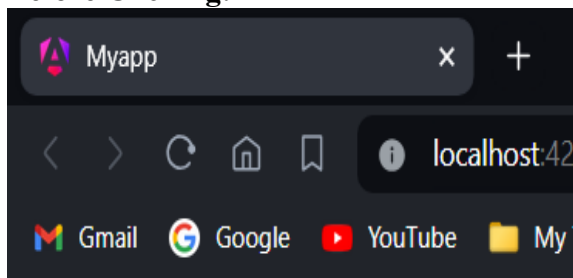
After Clicking:

CourseName	CourseId	Course Price
HTML	1	1000
CSS	2	2000
JS	3	3000
Node JS	4	4000
React	5	5000
Angular	6	6000

Experiment-6: Course Name: Angular JS
6-a) 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.

Aim: 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.
Description: Component communication is needed if data needs to be shared between the
components. In order to pass data from container/parent component to child component,
@Input decorator can be used. A child component can use @Input decorator on any property
type like arrays, objects, etc. making it a data-bound input property. The parent component can
pass value to the data-bound input property when rendering the child within it.

Program:
app.module.ts:
```
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { CourseListComponent } from './course-list/course-list.component';
@NgModule({
  declarations: [
    AppComponent,
    CourseListComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [
    AppComponent
  ]
})
export class AppModule { }
```

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

course-list.component.ts:

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

course-list.component.html:

```
<table border="1" *ngIf="course.length">
   <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>
```

app.component.html:

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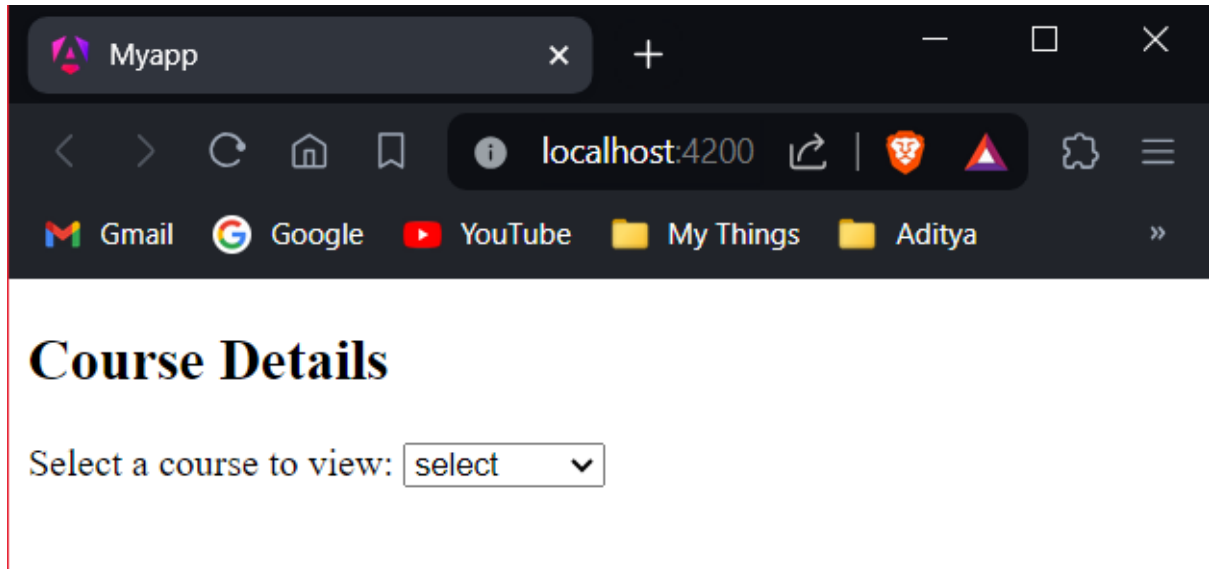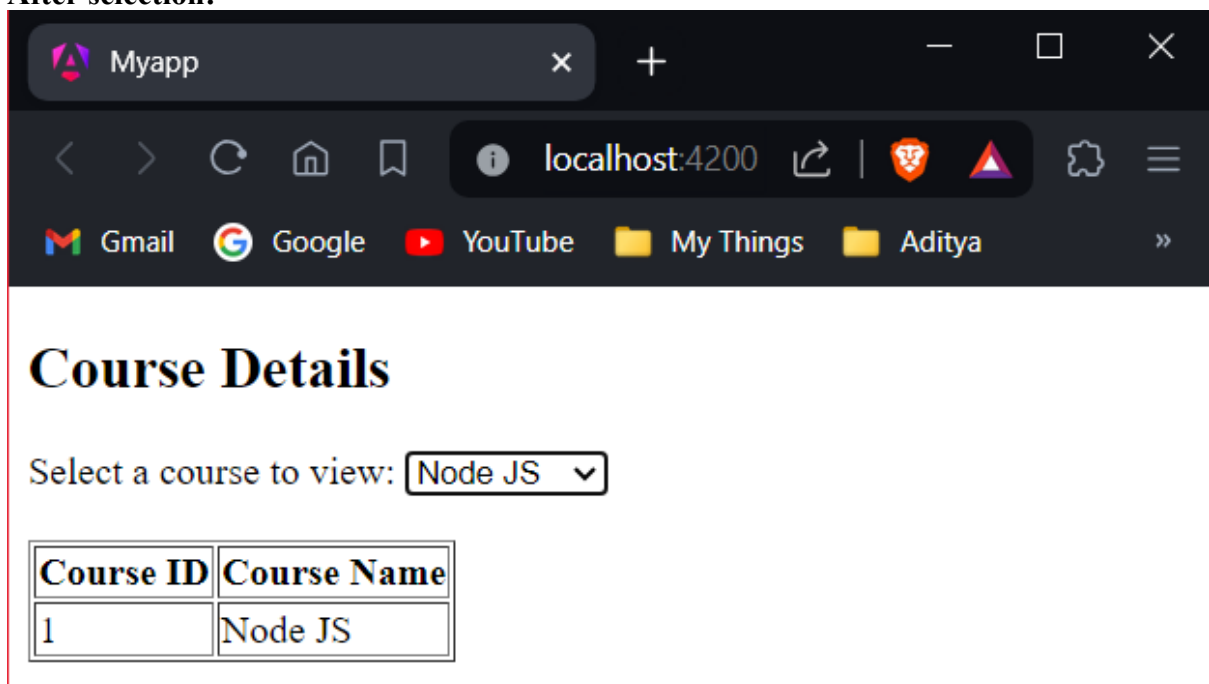
```
</select>
<br /><br />
<app-course-list [cName]="name"></app-course-list>
```

Output:

Before selection:

After selection:

6-b) 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.

Aim: 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.

Description: If a child component wants to send data to its parent component, then it must create a property with @Output decorator. The only method for the child component to pass data to its parent component is through events. The property must be of type EventEmitter. Create a property called onRegister of type EventEmitter and attach @Output decorator which makes the property to send the data from child to parent. emit() emits the courseName value i.e, send the courseName value back to parent component.

Program:
app.module.ts:
```
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { CourseListComponent } from './course-list/course-list.component';
@NgModule({
  declarations: [
    AppComponent,
    CourseListComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [
    AppComponent
  ]
})
export class AppModule { }
```

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`;
  }
}
```

course-list.component.ts:
```
import { Component, Input, Output, EventEmitter } from '@angular/core';
@Component({
  selector: 'app-course-list',
  templateUrl: './course-list.component.html',
  styleUrls: ['./course-list.component.css'],
})
export class CourseListComponent {
  @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-course-list (registerEvent)="courseReg($event)"></app-course-list>
<br /><br />
<div *ngIf="message">{{ message }}</div>
```

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>
```
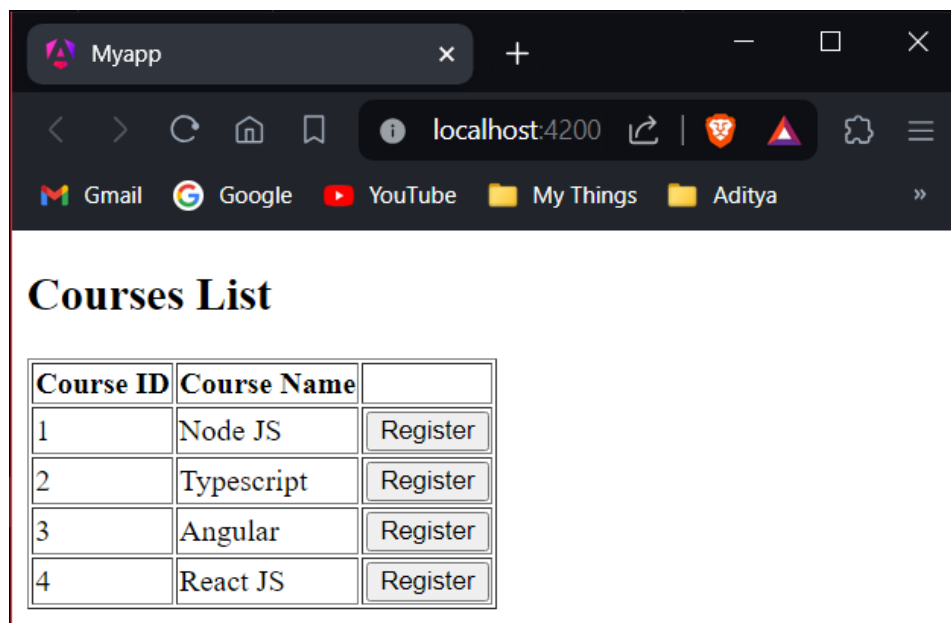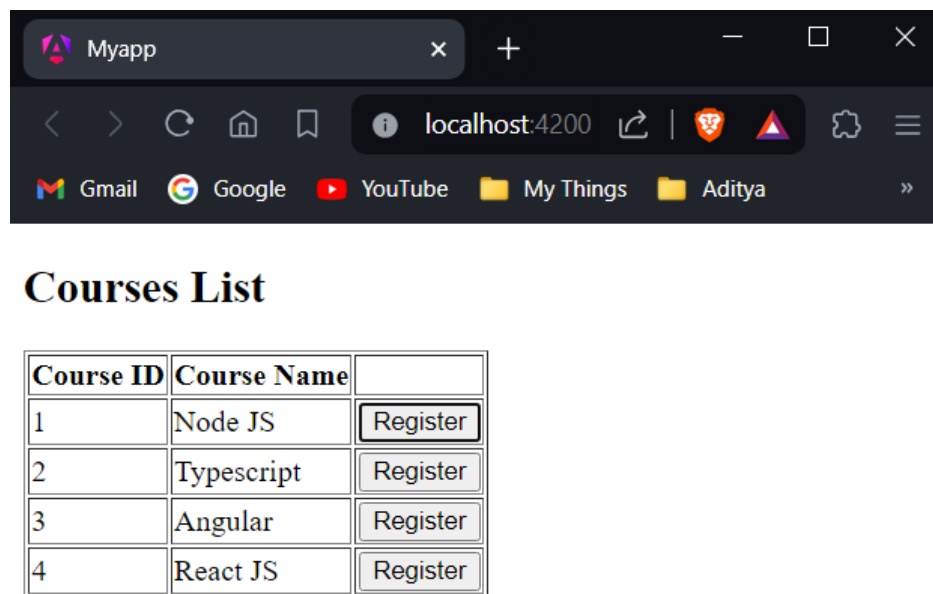
</table>

Output:

After Registration:

6-c) Module Name: Shadow DOM
Apply ShadowDOM and None encapsulation modes to components.

Aim: Apply ShadowDOM and None encapsulation modes to components.

Description: Shadow DOM is a web components standard by W3C. It enables encapsulation for DOM tree and styles. Shadow DOM hides DOM logic behind other elements and confines styles only for that component. Angular has built-in view encapsulation which enables you to use Shadow DOM. View Encapsulation defines how to encapsulate CSS styles into a component without flowing them to the rest of the page. The following three modes of encapsulation provided by Angular helps in controlling how the encapsulation has to be applied:

- ViewEncapsulation.Emulated (default)
- ViewEncapsulation.ShadowDOM
- ViewEncapsulation.None

Firstly, generate a component called "first" using the following command:

```
ng generate component first
```

Generate another component called "second" using the following command:

```
ng generate component second
```

app.module.ts:
```
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } 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 { }
```
ViewEncapsulation.ShadowDOM:
first.component.css:
```
.cmp {
  padding: 6px;
  margin: 6px;
  border: blue 2px solid;
}
```

first.component.html:
```
<div class="cmp">First Component</div>
```
second.component.css:
```
.cmp {
    border: green 2px solid;
    padding: 6px;
    margin: 6px;
}
```
second.component.html:
```
<div class="cmp">Second Component</div>
```
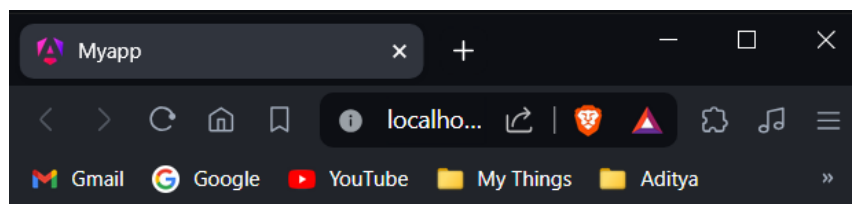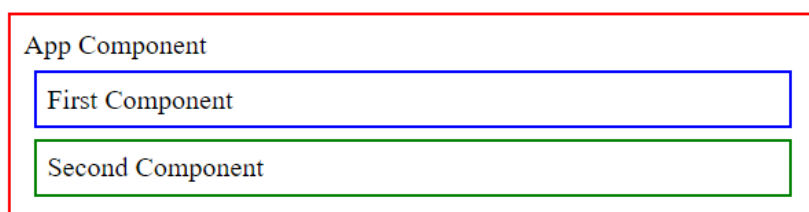**second.component.ts:**
```
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  selector: 'app-second',
  templateUrl: './second.component.html',
  styleUrls: ['./second.component.css'],
  // Experiment-6-c
  encapsulation: ViewEncapsulation.ShadowDom
})
export class SecondComponent {}
```
app.component.css:
```
.cmp {
    padding: 8px;
    margin: 6px;
    border: 2px solid red;
}
```
app.component.html:
```
<h3>CSS Encapsulation with Angular</h3>
<div class="cmp">
    App Component
    <app-first></app-first>
    <app-second></app-second>
</div>
```
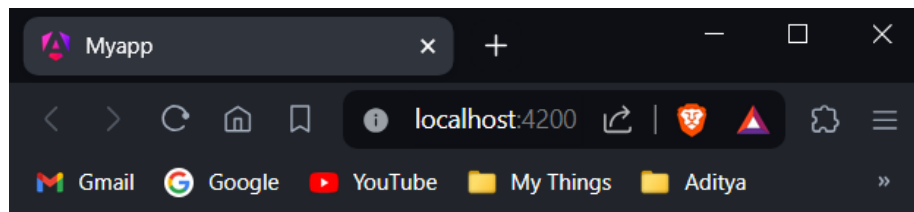
Output:

ViewEncapsulation.None:
app.component.ts:
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {}

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 {}

Output:

6-d) Module Name: Component Life Cycle
Override component life-cycle hooks and logging the corresponding messages to understand the flow.

Aim: Override component life-cycle hooks and logging the corresponding messages to understand the flow.

Description: A component has a life cycle that is managed by Angular. It includes creating a component, rendering it, creating and rendering its child components, checks when its data-bound properties change, and destroy it before removing it from the DOM. Angular has some methods/hooks which provide visibility into these key life moments of a component and the ability to act when they occur.
Lifecycle Hooks:
ngOnChanges – It gets invoked when Angular sets data-bound input property i.e., the property attached with @Input(). This will be invoked whenever input property changes its value.
ngOnInit – It gets invoked when Angular initializes the directive or component
ngDoCheck - It will be invoked for every change detection in the application
ngAfterContentInit – It gets invoked after Angular projects content into its view
ngAfterContentChecked – It gets invoked after Angular checks the bindings of the content it projected into its view.
ngAfterViewInit – It gets invoked after Angular creates component's views
ngAfterViewChecked – Invokes after Angular checks the bindings of the component's views
ngOnDestroy – It gets invoked before Angular destroys directive or component

To start this program, generate a component called child with the following command:

```
ng generate component child
```

Program:
app.module.ts:
```
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';
import { ChildComponent } from './child/child.component';
@NgModule({
  declarations: [
    AppComponent,
    ChildComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [
    AppComponent,
  ]
})
```

```
export class AppModule { }
```

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

child.component.ts:
```typescript
import { Component, OnChanges, Input } from '@angular/core';
@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent implements OnChanges {
  @Input() title!: string;
  ngOnChanges(changes: any): void {
    console.log('changes in child:' + JSON.stringify(changes));
  }
}
```
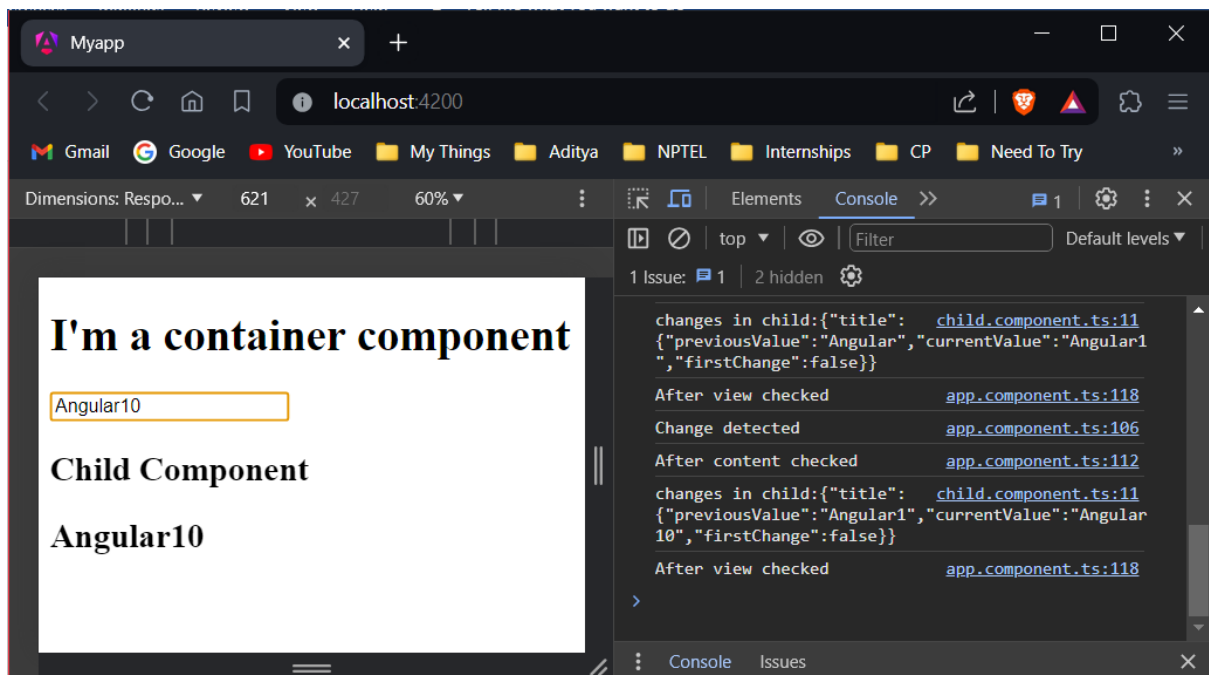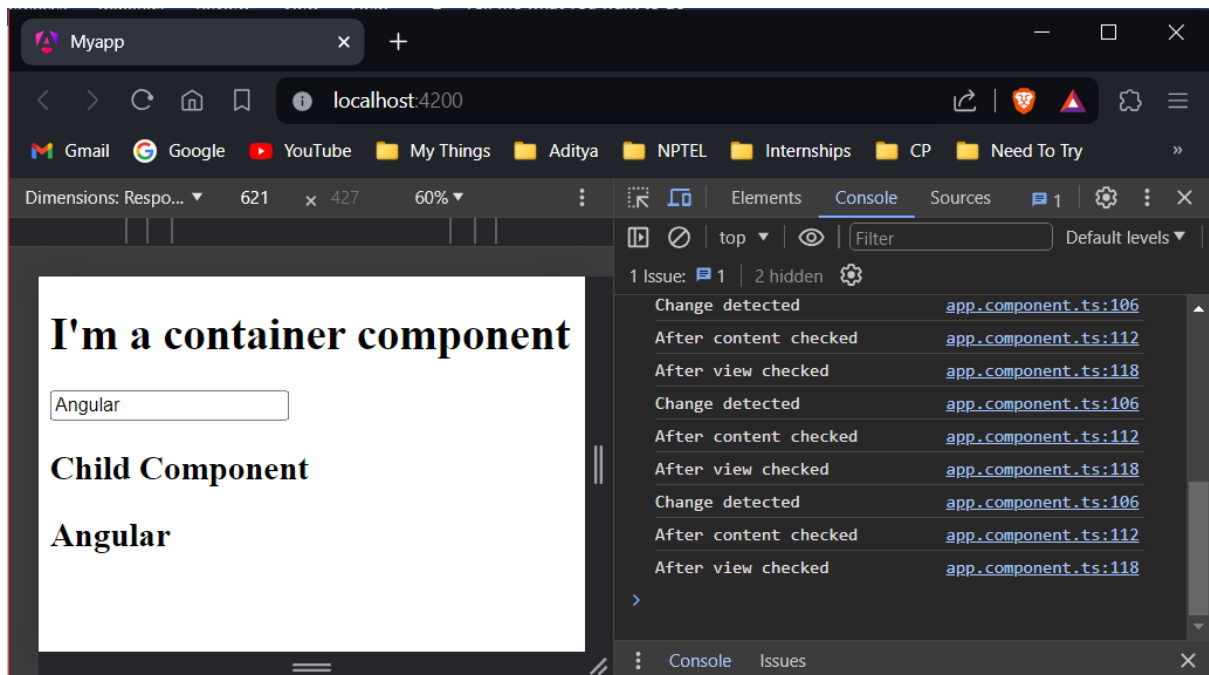
app.component.html:
```html
<div>
  <h1>I'm a container component</h1>
  <input type="text" [(ngModel)]="data" />
  <app-child [title]="data"></app-child>
</div>
```

child.component.html:
```html
<h2>Child Component</h2>
<h2>{{title}}</h2>
```

Output:

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

Aim: To create a course registration form as a template-driven form.
Description: A template-driven form is the simplest way to build a form in Angular. It uses Angular's two-way data-binding directive (ngModel) to create and manage the underlying form instance. Additionally, as the name suggests, a template form is mainly driven by the view component. So, it uses directives placed in HTML rather than TypeScript or JavaScript to manage the form. A template-driven form is asynchronous due to the use of "directives" because the creation of form controls is delegated to the declared directives (IoC). Template-driven forms are the forms that are created using Angular template syntax. In template-driven form, data binding, validation, etc., will be written in the template itself.

To start this program, generate a component called child with the following command:
```
ng generate component course-form
```
Also, install bootstrap module using the following command:
```
npm i bootstrap@3.3.7 --save
```

Program:
angular.json:
```
"styles": [
        "src/styles.css",
        "./node_modules/bootstrap/dist/css/bootstrap.min.css"
],
```
app.module.ts:
```
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { CourseFormComponent } from './course-form/course-form.component';
import { FormsModule } from '@angular/forms';
@NgModule({
  declarations: [
    AppComponent,
    CourseFormComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [
    AppComponent,
  ]
})
export class AppModule { }
```

Create "course.ts" inside course-form component folder:

```
export class Course {
   constructor(
      public courseId: number,
      public courseName: string
   ) { }
}
```

course-form.component.ts:

```
import { Component } from '@angular/core';
import { Course } from './course';
@Component({
  selector: 'app-course-form',
  templateUrl: './course-form.component.html',
  styleUrl: './course-form.component.css'
})
export class CourseFormComponent {
  course=new Course(1,"Angular");
  submitted=false;
  onSubmit(){
    this.submitted=true;
  }
}
```

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>
            <label for="name">Course Name</label>
            <input type="text" class="form-control" required [(ngModel)]="course.courseName" name="name" #name="ngModel">
         </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>
```
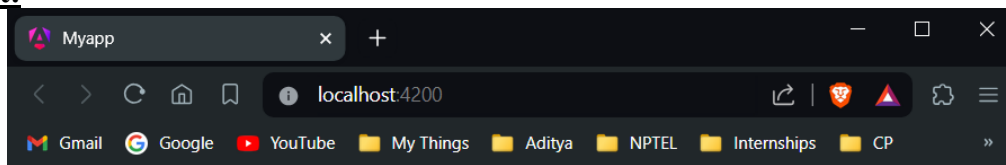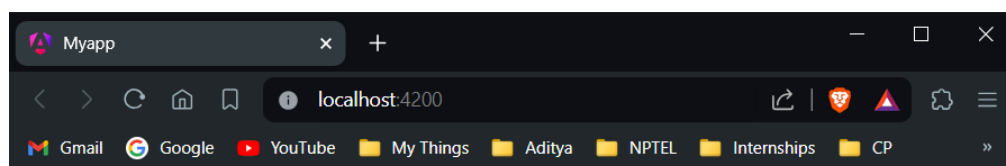
```
    <div class="row">
        <div class="col-xs-3">Course Name</div>
        <div class="col-xs-9 pull-left">{{ course.courseName }}</div>
        <br><button class="btn btn-default" (click)="submitted=false">Edit</button>
    </div>
  </div>
</div>
```

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;
}
```
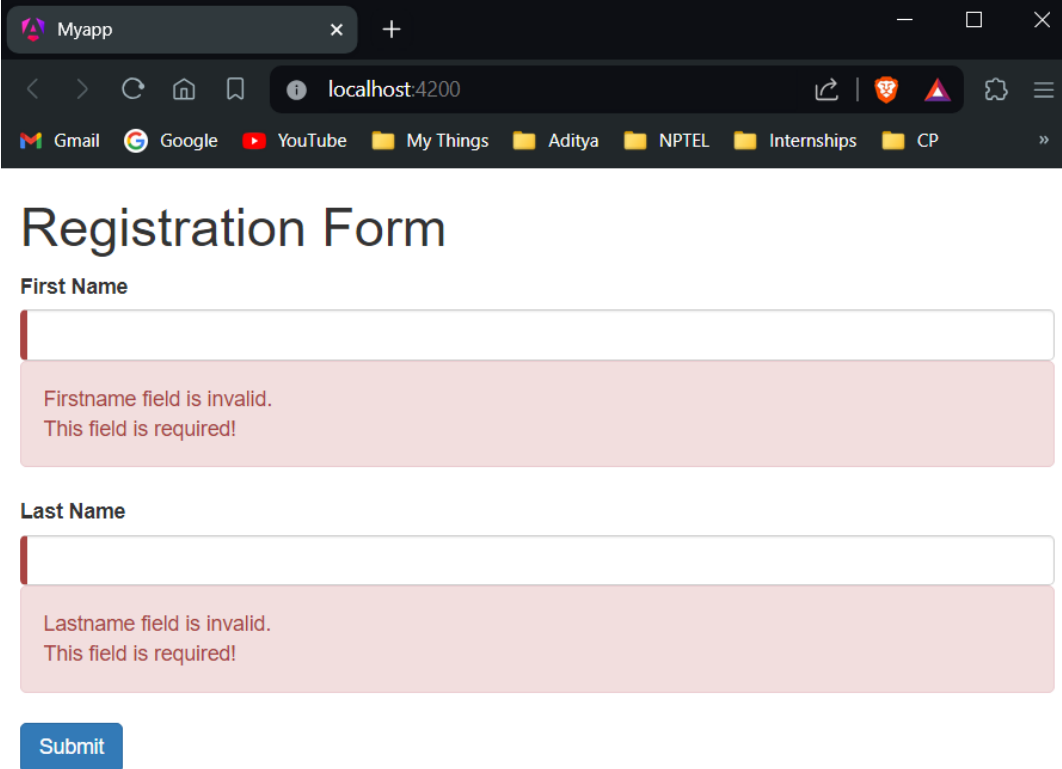
app.component.html:
```
<app-course-form></app-course-form>
```

Output:

7-b) Module Name: Model Driven Forms or Reactive Forms
Create an employee registration form as a reactive form.
Aim: To create an employee registration form as a reactive form.
Description: Angular reactive forms facilitate a reactive style of programming that favors explicit management of the data flowing between a non-UI data model (typically retrieved from a server) and a UI-oriented form model that retains the states and values of the HTML controls on screen. Reactive forms offer the ease of using reactive patterns, testing, and validation.
With reactive forms, you create a tree of Angular form control objects in the component class and bind them to native form control elements in the component template, using techniques described in this guide.
You create and manipulate form control objects directly in the component class. As the component class has immediate access to both the data model and the form control structure, you can push data model values into the form controls and pull user-changed values back out. The component can observe changes in form control state and react to those changes.
To begin, generate a component called RegistrationForm with the following command:

```
ng generate component RegistrationForm
```

Program:
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',
  styleUrl: './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],
    });
  }
}
```
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>
```

```html
      </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>
      <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>
    </div>
</div>
```

registration-form.component.css:
```css
.ng-valid[required] {
    border-left: 5px solid #42A948;
}
.ng-invalid:not(form) {
    border-left: 5px solid #a94442;
}
```

app.component.ts:
```html
<app-registration-form></app-registration-form>
```

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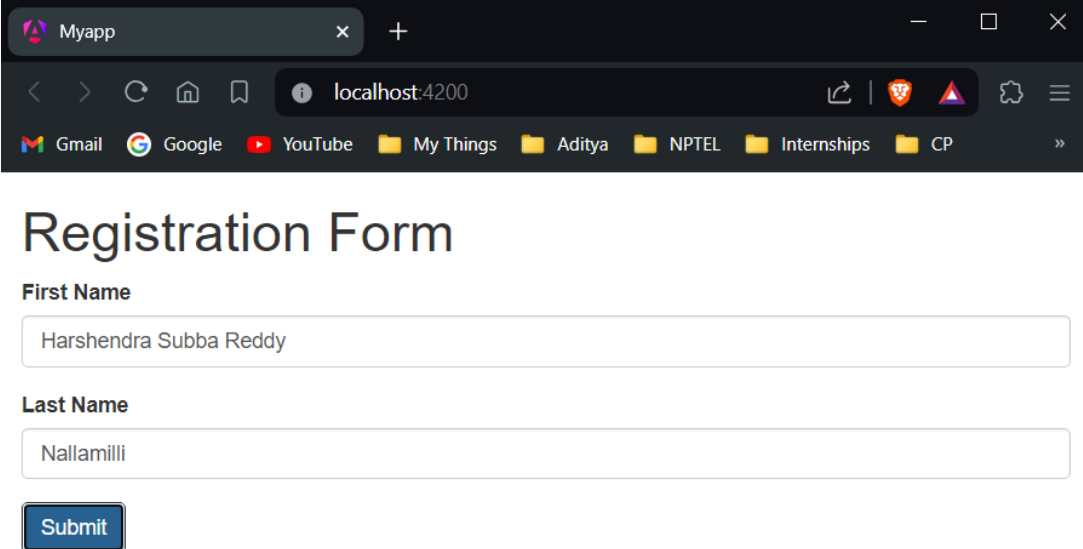
```
    AppComponent,
  ]
})
export class AppModule { }
```

Output:

After filling in details and submitting:

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

Aim: To create a custom validator for an email field in the employee registration reactive form.

Description: While creating forms, there can be situations 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, custom validators can be created to implement the required validation functionality.
Custom validation can be applied to form controls of a Reactive Form in Angular.
- Custom validators are implemented as separate functions inside the component.ts file.
- These functions can be added to the list of other validators configured for a form control.

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

registration-form.component.html:
```
<div class="container">
    <h1>Registration Form</h1>
    <form [formGroup]="registerForm">
        <div class="form-group">
            <label>First Name</label>
```

```html
      <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">
      <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'].message }}
        </p>
      </div>
    </div>
    <button type="submit" class="btn btn-primary" (click)="submitted=true">Submit</button>
  </form>
  <br />
  <div [hidden]="!submitted">
    <h3> Employee Details </h3>
    <p>First Name: {{ registerForm.get('firstName')?.value }} </p>
    <p> Last Name: {{ registerForm.get('lastName')?.value }} </p>
    <p>Email: {{ registerForm.get('email')?.value }}</p>
  </div>
</div>
```

registration-form.component.css:
```css
.ng-valid[required] {
  border-left: 5px solid #42A948;
}
.ng-invalid:not(form) {
  border-left: 5px solid #a94442;
}
```
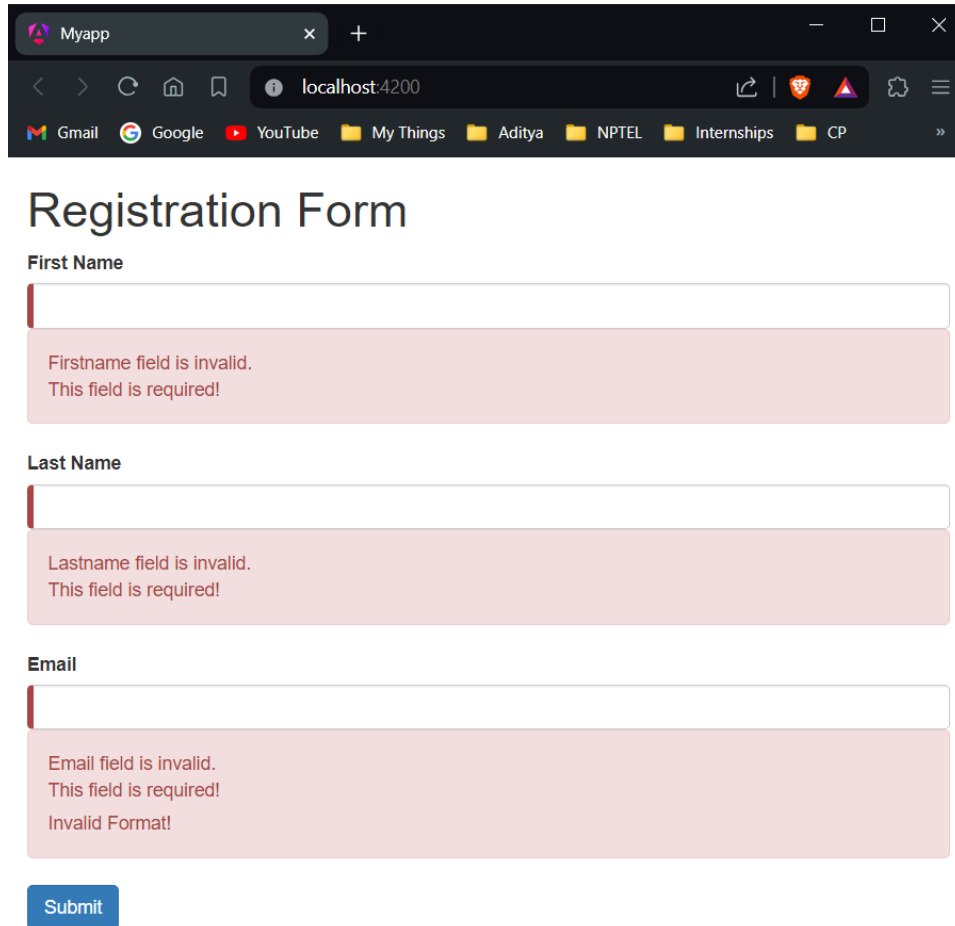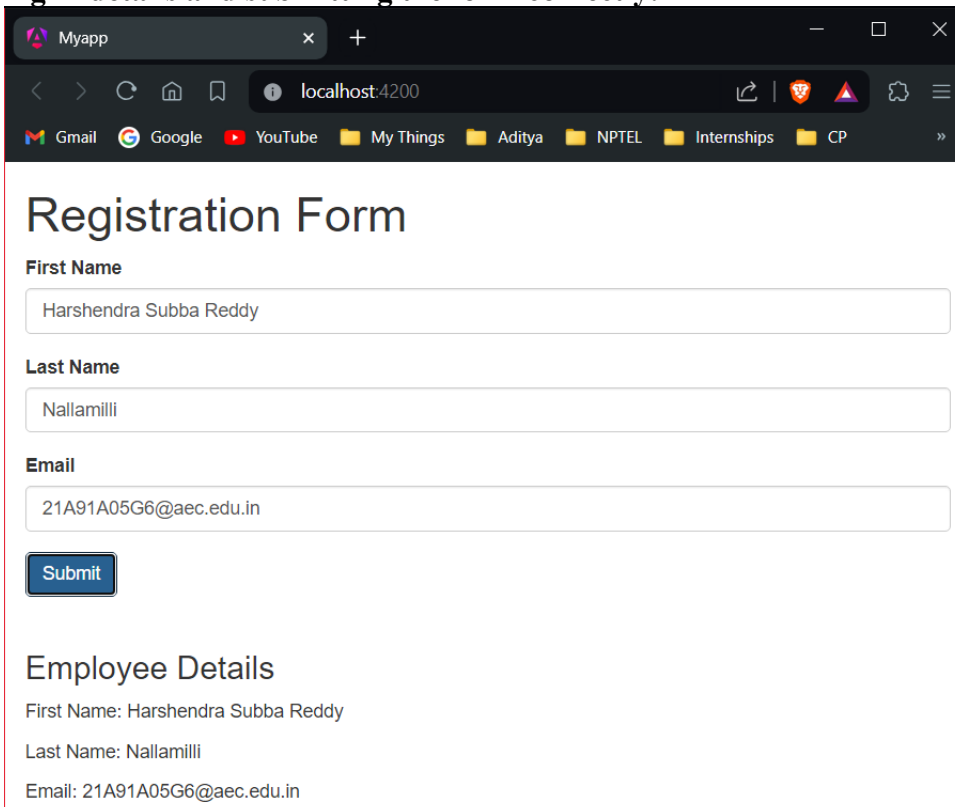
app.component.ts:

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

Output:

After filling in details and submitting the form correctly:

Experiment-8: Course Name: Angular JS
8-a) 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: A template-driven form is the simplest way to build a form in Angular. It uses Angular's two-way data-binding directive (ngModel) to create and manage the underlying form instance. Additionally, as the name suggests, a template form is mainly driven by the view component. So, it uses directives placed in HTML rather than TypeScript or JavaScript to manage the form. 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:
Create "email.validator.ts" inside course-form component folder:

```
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;
   }
}
```

course.ts:

```
export class Course {
   constructor(
      public courseId: number,
      public courseName: string,
      public email: string
   ) {}
}
```

course-form.component.ts:

```
import { Component } from '@angular/core';
import { Course } from './course';
@Component({
  selector: 'app-course-form',
  templateUrl: './course-form.component.html',
  styleUrl: './course-form.component.css'
})
export class CourseFormComponent {
```

```
  course=new Course(1,"Angular","sample@gmail.com");
  submitted=false;
  onSubmit(){
    this.submitted=true;
  }
}
```

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>
        <label for="name">Course Name</label>
        <input          type="text"          class="form-control"          required
[(ngModel)]="course.courseName" name="name" #name="ngModel">
        <label for="email">Email</label>
        <input type="email" class="form-control" required [(ngModel)]="course.email"
name="email" #email="ngModel" validateEmail>
        <div *ngIf="email.errors && (email.dirty || email.touched)">
          <div    *ngIf="email.errors['emailInvalid']"    class="alert    alert-danger">{{
email.errors['emailInvalid'] }}</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">Email</div>
      <div class="col-xs-9 pull-left">{{ course.email }}</div>
      <br><button class="btn btn-default" (click)="submitted=false">Edit</button>
    </div>
```
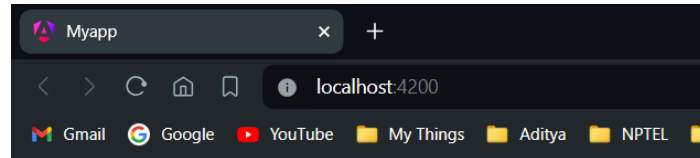
```
    </div>
  </div>
```

course-form.component.css:
```css
input.ng-valid[required]  {
    border-left: 5px solid #42A948;
}
input.ng-dirty.ng-invalid:not(form)  {
    border-left: 5px solid #a94442;
}
```

app.module.ts:
```typescript
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';
import { CourseFormComponent } from './course-form/course-form.component';
import { EmailValidator } from './course-form/email.validator';
@NgModule({
  declarations: [
    AppComponent,
    CourseFormComponent,
    EmailValidator
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [
    AppComponent,
  ]
})
export class AppModule { }
```
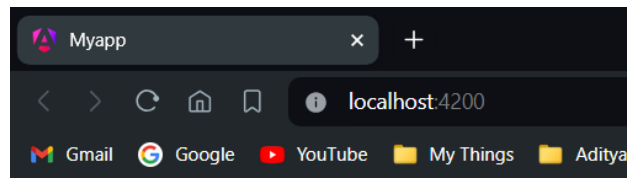
app.component.html:
```html
<app-course-form></app-course-form>
```

Output:

Course Form

Course Id

1

Course Name

Angular

Email

sample@gmail.com

Submit | New Course

Course Form

Course Id

1

Course Name

Angular

Email

sample123@

Email is invalid

Submit | New Course

You submitted the following:

Course ID	1
Course Name	Angular
Email	sample123@gmail.com

Edit

8-b) 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.
<u>Aim:</u> To create a Book Component which fetches book details like id, name and displays them in a list format. Store the book details in an array and fetch the data using a custom service.
<u>Description:</u> 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 share the code across components of an application and to make HTTP requests.
To begin, generate a component called book with the following command:

```
ng generate component book
```

<u>**Program:**</u>
Create "book.ts" inside book component folder:
```
export class Book {
   id!: number;
   name!: string;
}
```
Create "books-data.ts" inside book component folder:
```
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: 'Angular JS' }
];
```
Move inside book folder: cd .\src\app\book\
Generate a service called "book": ng generate service book
Come back to myapp folder: cd ../../..
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: #607d8b;
    line-height: 0.5em;
    position: relative;
    left: -1px;
```

```
    top: -4px;
    height: 1.8em;
    margin-right: 0.8em;
    border-radius: 4px 0 0 4px;
}
```
app.component.html:
```
<app-book></app-book>
```
app.module.ts:
```
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
@NgModule({
  declarations: [
    AppComponent,
    BookComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
  ],
  providers: [],
  bootstrap: [
    AppComponent,
  ]
})
export class AppModule { }
```

<u>Output:</u>
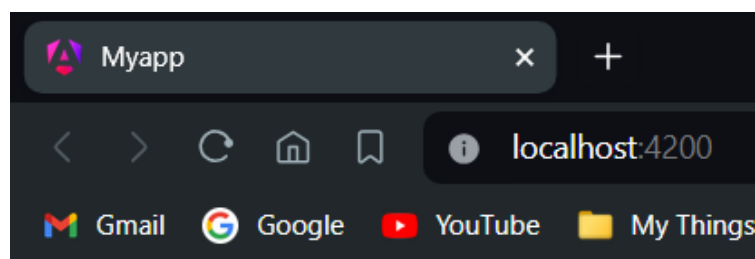
8-c) Module Name: RxJS Observables
Create and use an observable in Angular.

Aim: To create and use an observable in Angular.

Description: 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. Angular team has recommended Observables for asynchronous calls because of the following reasons:

- Promises emit a single value whereas observables (streams) emit many values
- Observables can be cancellable where Promises are not cancellable. If an HTTP response is not required, observables allow us to cancel the subscription whereas promises execute either success or failure callback even if the results are not required.
- Observables support functional operators such as map, filter, reduce, etc.,

Program:
app.component.ts:
```
import { Component } from '@angular/core';
import { Observable } from 'rxjs';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
})
export class AppComponent {
  data!: Observable<number>;
  myArray: number[] = [];
  errors!: boolean;
  finished!: boolean;
  fetchData(): void {
    this.data = new Observable(observer => {
      setTimeout(() => { observer.next(11); }, 1000),
      setTimeout(() => { observer.next(22); }, 2000),
      setTimeout(() => { observer.complete(); }, 3000);
    });
    this.data.subscribe((value) => this.myArray.push(value),
      error => this.errors = true,
      () => this.finished = true);
  }
}
```
app.component.html:
```
<b> Using Observables!</b>
<h6 style="margin-bottom: 0">VALUES:</h6>
<div *ngFor="let value of myArray">{{ value }}</div>
<div style="margin-bottom: 0">ERRORS: {{ errors }}</div>
<div style="margin-bottom: 0">FINISHED: {{ finished }}</div>
<button style="margin-top: 2rem" (click)="fetchData()">Fetch Data</button>
```
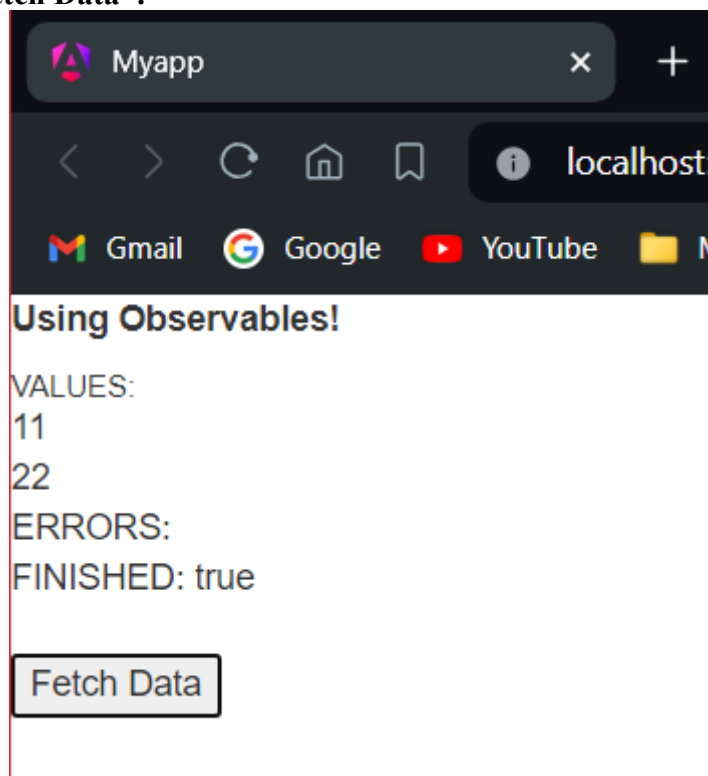
Output:
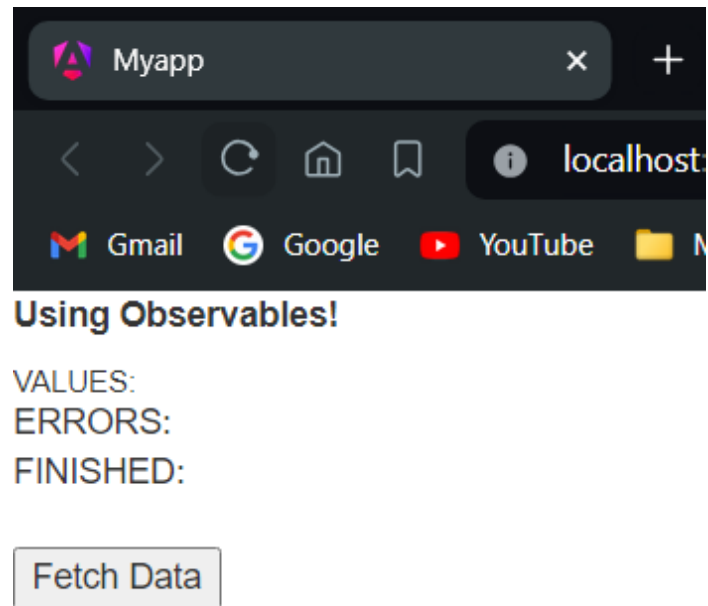
After clicking "Fetch Data":

Experiment-9: Course Name: Angular JS
9-a) Module Name: Server Communication using HttpClient
Create an application for Server Communication using HttpClient
Aim: To 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.
Program:
app.module.ts:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import {HttpClientModule} from '@angular/common/http';
import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
@NgModule({
  imports: [BrowserModule, HttpClientModule],
  declarations: [AppComponent, BookComponent],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

book.service.ts:

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse, HttpHeaders } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError, tap } from 'rxjs/operators';
import { Book } from './book';
@Injectable({
  providedIn:'root'
})
export class BookService {
  booksUrl = 'http://localhost:3020/bookList';
  constructor(private http: HttpClient) { }
  getBooks(): Observable<Book[]> {
    return this.http.get<Book[]>('http://localhost:3020/bookList').pipe(
      tap((data: any) => console.log('Data Fetched:' + JSON.stringify(data))),
      catchError(this.handleError));
  }
  addBook(book: Book): Observable<any> {
    const options = new HttpHeaders({ 'Content-Type': 'application/json' });
    return this.http.post('http://localhost:3020/addBook', book, { headers: options }).pipe(
      catchError(this.handleError));
  }
```

```typescript
  updateBook(book: Book): Observable<any> {
    const options = new HttpHeaders({ 'Content-Type': 'application/json' });
    return this.http.put<any>('http://localhost:3020/update', book, { headers: options }).pipe(
      tap((_: any) => console.log(`updated hero id=${book.id}`)),
      catchError(this.handleError)
    );
  }
  deleteBook(bookId: number): Observable<any> {
    const url = `${this.booksUrl}/${bookId}`;
    return this.http.delete(url).pipe(
      catchError(this.handleError));
  }
  private handleError(err: HttpErrorResponse): Observable<any> {
    let errMsg = '';
    if (err.error instanceof Error) {
      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.component.ts:

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

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

book.component.html:

```
<h2>{{ title }}</h2>
<h2>My Books</h2>
<ul class="books"><li *ngFor="let book of books"> <span class="badge">{{ book.id
}}</span> {{ book.name }} </li></ul>
<button class="btn btn-primary" (click)="ADD_BOOK = true">Add Book</button> 
<button    class="btn    btn-primary"    (click)="UPDATE_BOOK    =    true">Update
Book</button> 
<button class="btn btn-primary" (click)="DELETE_BOOK = true">Delete Book</button>
<br />
<div *ngIf="ADD_BOOK">
  <table>
    <tr><td>Enter Id of the book:</td><td><input type="number" #id /></td></tr>
    <br />
    <tr><td>Enter Name of the Book:</td><td><input type="text" #name /><br /></td></tr>
    <br />
    <tr><td><button    class="btn    btn-primary"    (click)="addBook(id.value,    name.value);
ADD_BOOK = false">Add Record</button></td></tr>
  </table>
  <br />
</div>
<div *ngIf="UPDATE_BOOK">
  <table>
    <tr><td>Enter Id of the book:</td><td><input type="number" #id /></td></tr><br />
    <tr><td>Enter Name of the Book:</td><td><input type="text" #name /><br /></td></tr>
    <br />
    <tr><td><button    class="btn    btn-primary"    (click)="updateBook(id.value,    name.value);
UPDATE_BOOK = false">Update Record</button></td></tr>
  </table>
</div>
<br />
<div *ngIf="DELETE_BOOK">
  <table>
    <tr><td>Enter Id of the book:</td><td><input type="number" #id /></td></tr><br />
```

```
    <tr><td><button class="btn btn-primary" (click)="deleteBook(id.value); DELETE_BOOK
= false">Delete Record</button></td></tr>
  </table>
</div>
<div class="error" *ngIf="errorMessage">{{ errorMessage }}</div>
```

Output:

Experiment-10: Course Name: Angular JS
10-a) Module Name: Routing Basics, Router Links
Create multiple components and add routing to provide navigation between them.
Aim: To 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 and Create modular applications.
Program:
dashboard.component.ts:

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

dashboard.component.html:

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

book.service.ts:

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse, HttpHeaders, HttpResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError, tap, map} from 'rxjs/operators';
import { Book } from './book';
@Injectable({
  providedIn:'root'
```

```typescript
})
export class BookService {
  booksUrl = 'http://localhost:3020/bookList';
  private txtUrl = './assets/sample.txt';
  constructor(private http: HttpClient) { }
  getBooks(): Observable<Book[]> {
    return this.http.get<any>(this.booksUrl, {observe:'response'}).pipe(
      tap((data: any) => console.log('Data Fetched:' + JSON.stringify(data))),
      catchError(this.handleError));
  }
  getBook(id: any) {
    return this.getBooks().pipe(
      map((books) => books.find((book) => book.id == id))
    );
  }
  addBook(book: Book): Observable<any> {
    const options = new HttpHeaders({ 'Content-Type': 'application/json' });
    return this.http.post('http://localhost:3020/addBook', book, { headers: options }).pipe(
      catchError(this.handleError));
  }
  updateBook(book: Book): Observable<any> {
    const options = new HttpHeaders({ 'Content-Type': 'application/json' });
    return this.http.put<any>('http://localhost:3020/update', book, { headers: options }).pipe(
      tap((_: any) => console.log(`updated hero id=${book.id}`)),
      catchError(this.handleError)
    );
  }
  deleteBook(bookId: number): Observable<any> {
    const url = `${this.booksUrl}/${bookId}`;
    return this.http.delete(url).pipe(
      catchError(this.handleError));
  }
  private handleError(err: HttpErrorResponse): Observable<any> {
    let errMsg = '';
    if (err.error instanceof Error) {
      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-detail.component.ts:

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

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

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

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

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

Output:

Tour of Books1

Top Books

| CSS 3 | Java Script | Ajax Programming | jQuery |

Tour of Books1

Dashboard   Books

My Books

1. HTML 5
2. CSS 3
3. Java Script
4. Ajax Programming

Tour of Books1

Dashboard   Books

CSS 3 details!

id: 2

name: CSS 3

Back

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

Aim: To Install MongoDB and configure ATLAS.

Description: MongoDB is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use. MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.

Installation of MongoDB:

Step-1: Search "MongoDB Community Download" in browser and open first link.

Step-2: Select all the required options as below and click download.
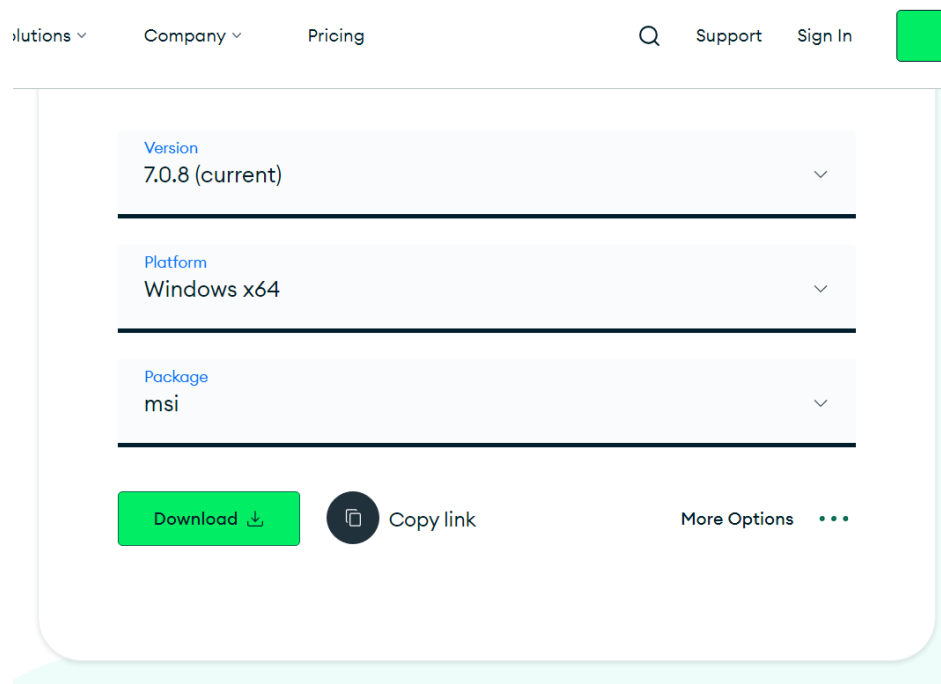
Step-3: Double click on the downloaded file and click next.

Step-4: Accept the license agreement and click next.
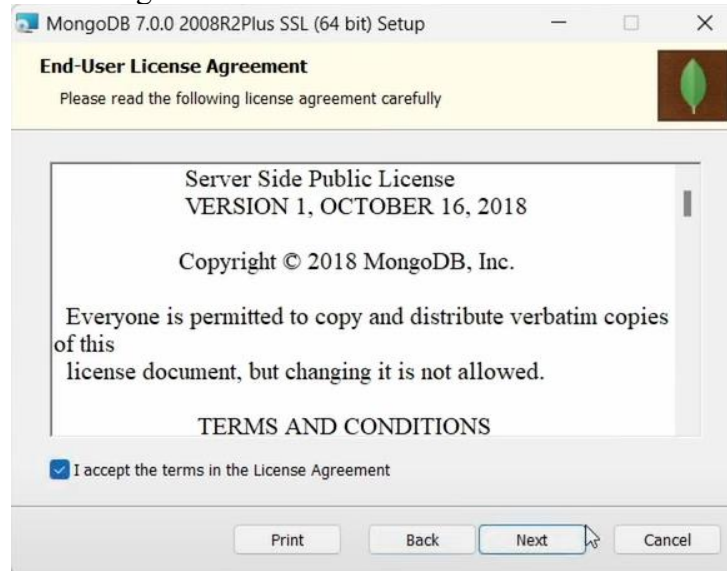
Step-5: Choose the Setup Type as "Complete" and click on next.
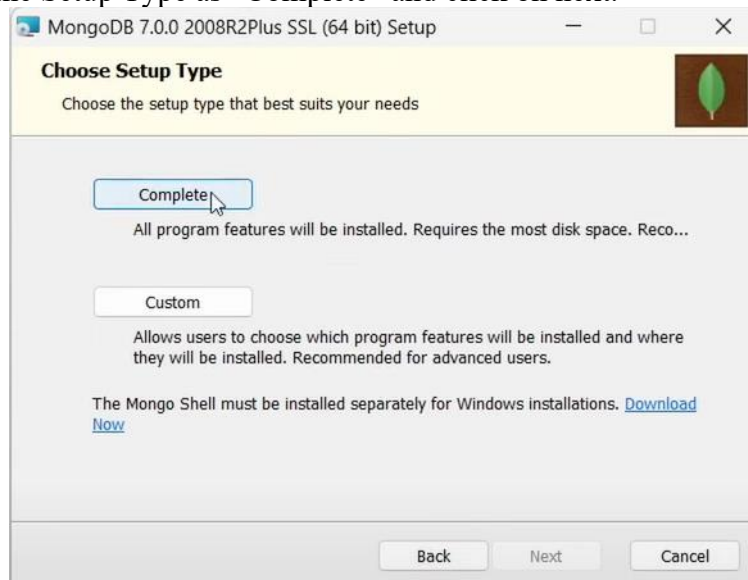
Step-6: Keep the service configuration unchanged and click next.
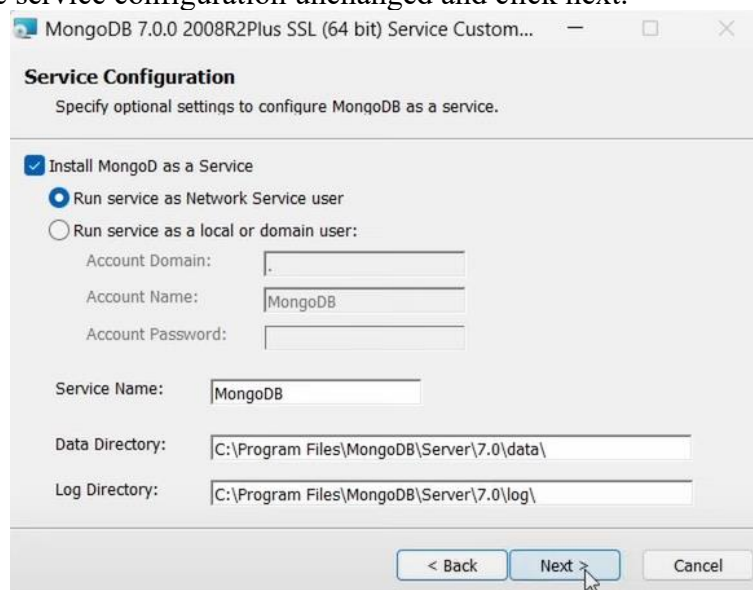
Step-7: Hit Install and wait for some time.
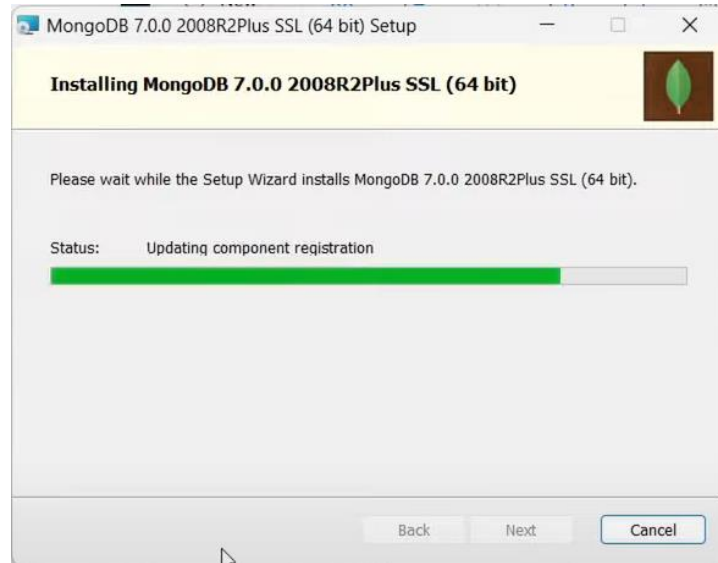
Finally, our MongoDB Compass Application has been installed.
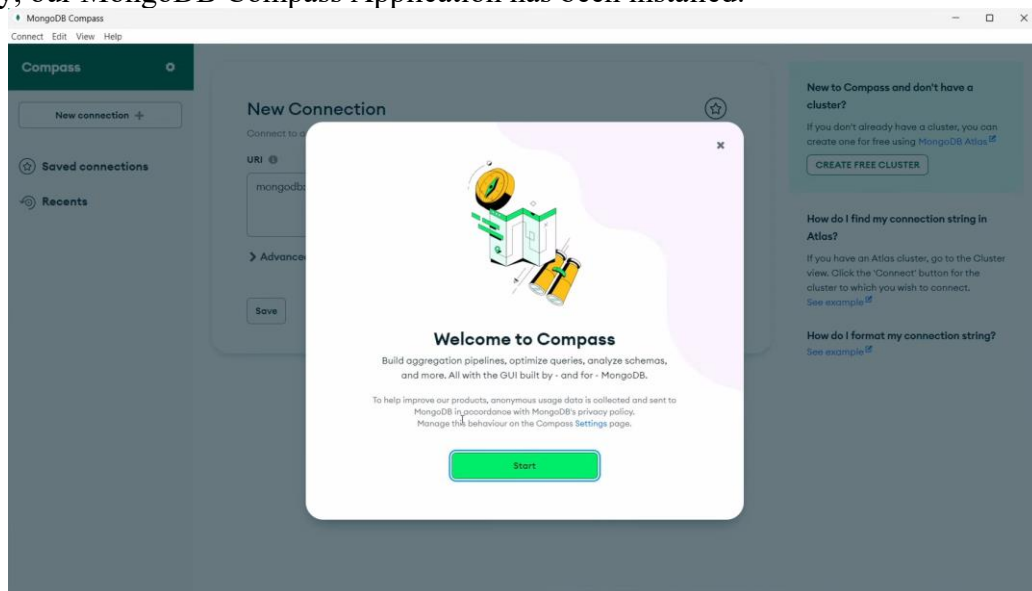
<u>**Configuring MongoDB Shell:**</u>

Step-1: Search "MongoDB Shell Download" in browser and open first link.

Step-2: Select all the required options as below and click download.
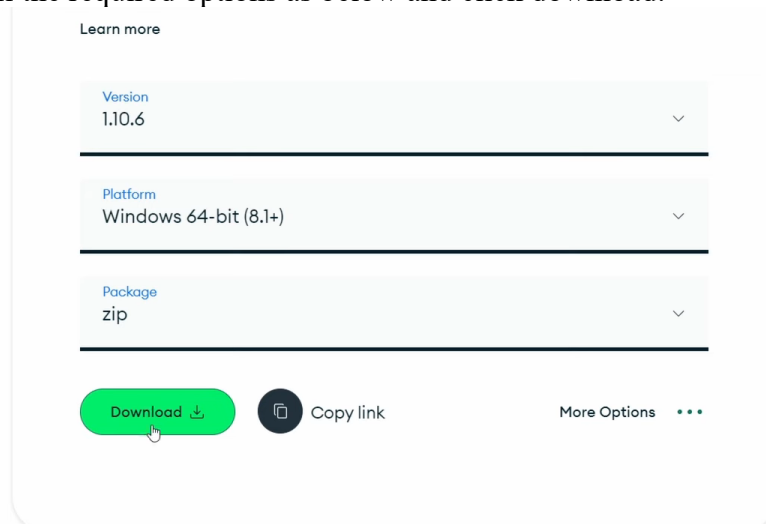
Step-3: After downloading the zip file, extract the contents inside "C:\Program Files\MongoDB". Later, go inside the extracted folder and bin folder inside. Copy the path.
Step-4: Now go to "Environment Variables" and add the copied path into "System Variables Path" as shown below.
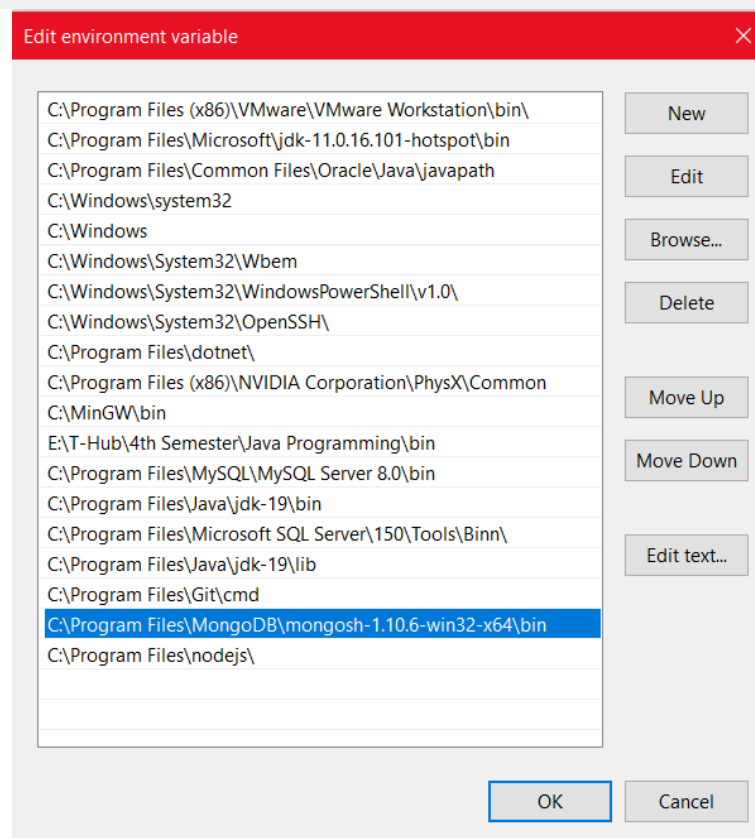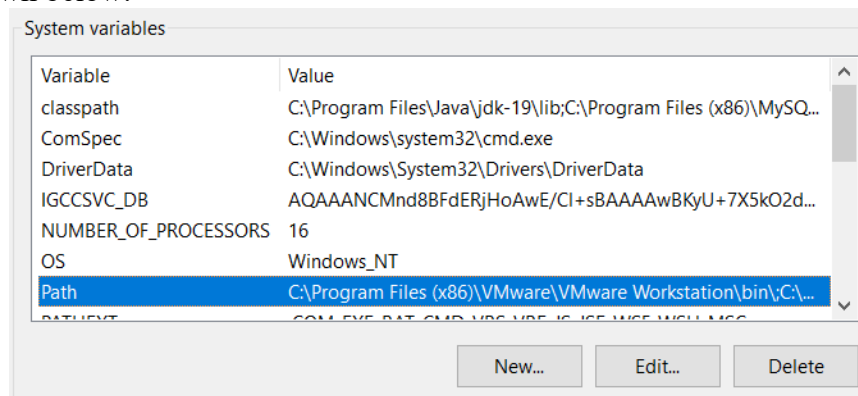
Step-5: Click on "Ok" 3 times. After this step, open command prompt and type in the command "mongosh" to access MongoDB Shell from anywhere within your system.

Creation of MongoDB ATLAS Cluster: It is a cloud-hosted database-as-a-service.
Step-1: Create an ATLAS Account on "cloud.mongodb.com". Then create a new project.
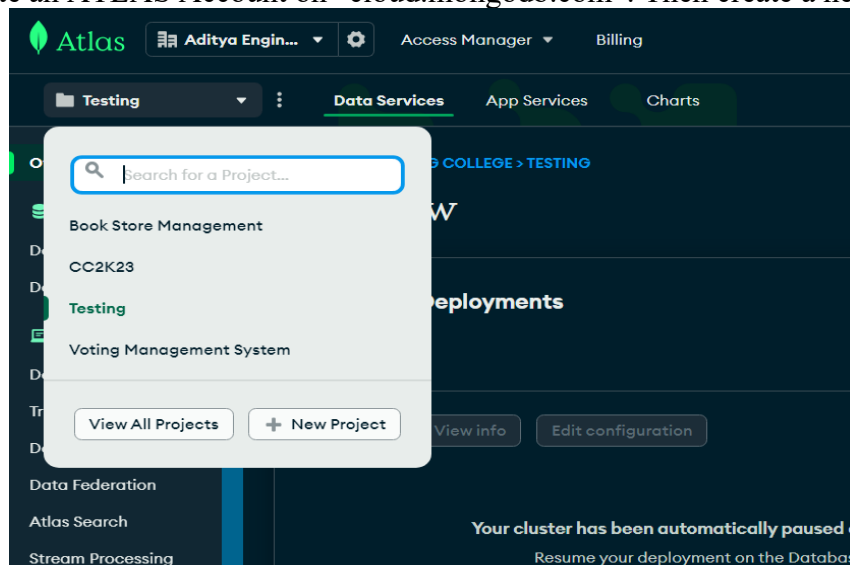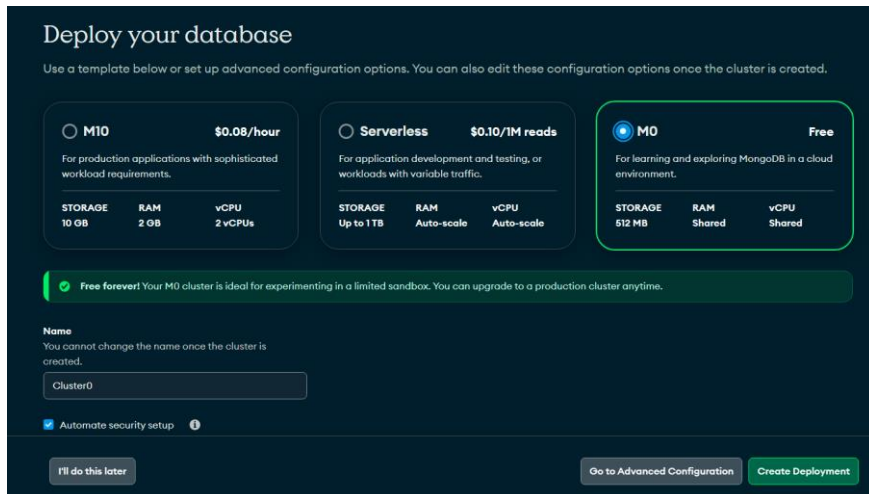
Step-2: Name your project as "MST" and click on "Create Project". After creation of project, create a new Deployment.
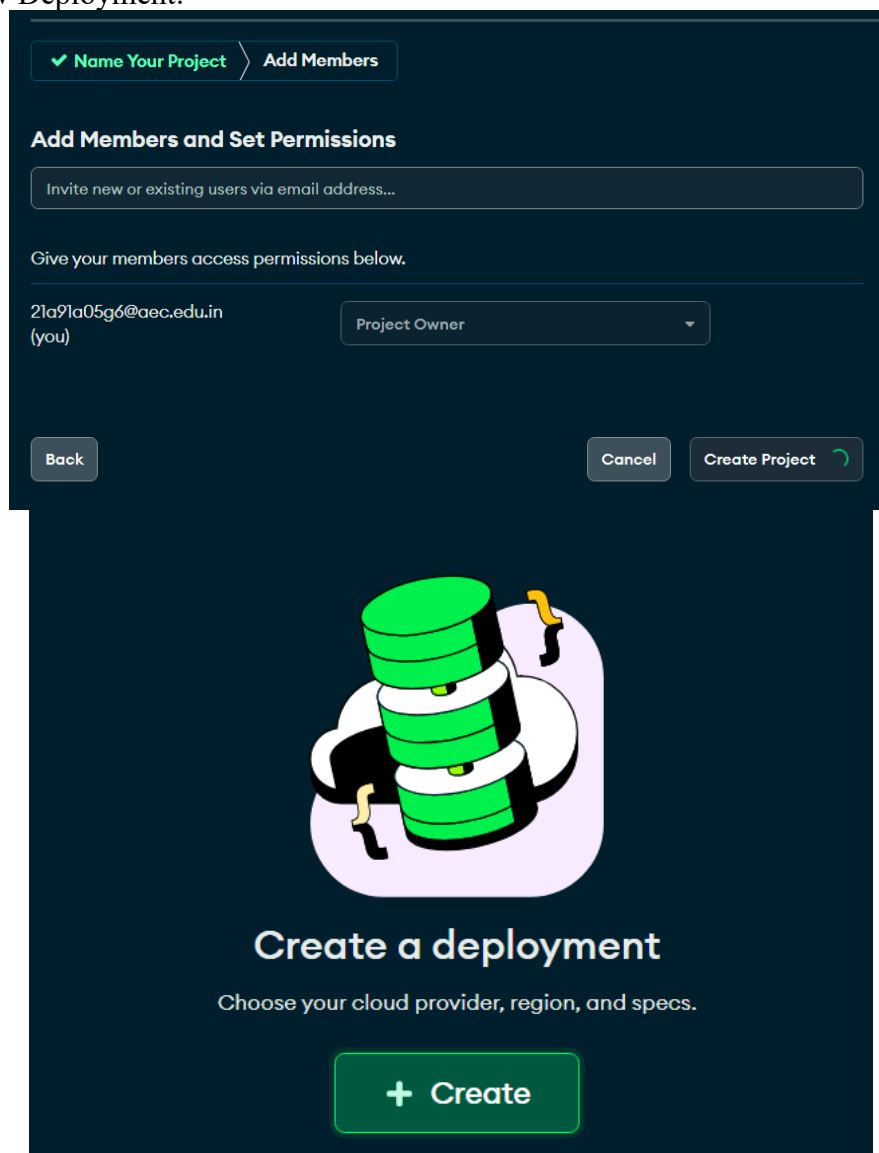
Step-3: Click on "M0" which will make your cluster free of cost and click create. It'll take 2 mins of time to create.

Step-4: After this step, you'll see this page. Copy the password and click on "Create Database User" and click "Choose a Connection Method".
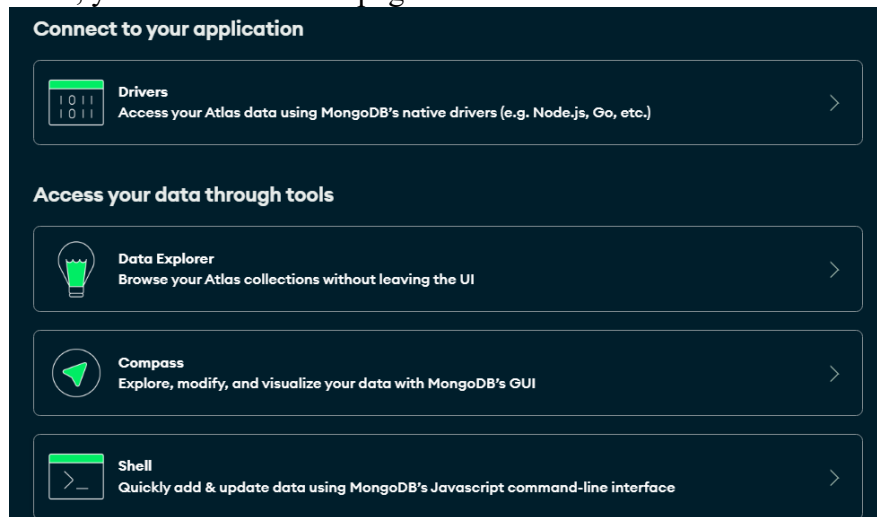
Step-5: After this, you'll see the below page. Click on "Shell" and click "I have Shell".

Step-6: Copy the connection string present there and paste the same in your command prompt.

Let's try to see what databases we have in this cluster with command "**show dbs**".

You can see we have sample_mflix which was automatically created by MongoDB, admin and local databases.

To create a new database, enter "**use new_database_name**".

To see the collections, enter "**show collections**".

As we just created a new database, we cannot find any created collections. **Remember that the changes we make in here will reflect the same in MongoDB Atlas Cloud.**

11-b) Module Name: Introduction to the CRUD Operations
Write MongoDB queries to perform CRUD operations on document using insert(), find(), update(), remove()
Aim: Write MongoDB queries to perform CRUD operations on document using insert(), find(), update(), remove()
Description:
Before diving in, understand the basic terminology in MongoDB.
MongoDB stores data records as **documents** (specifically BSON documents) which are gathered together in **collections**. A **database** stores one or more collections of documents.
Database: Databases hold one or more collections of documents.
Collection: MongoDB stores documents in collections. Collections are analogous to tables in relational databases.
Document: MongoDB documents are composed of field-and-value pairs and stores data records as BSON format. BSON is a binary representation of JSON documents, though it contains more data types than JSON.
Queries:
Using MongoDB, one can make CRUD operations. CRUD means Create, Read, Update, Delete. We can perform these operations on our data using MongoDB Shell Queries. Let us see them one by one.
To make these operations, we must need a collection to work with. So, let's create a collection using the command **db.createCollection("collection_name")**.

```
Atlas atlas-9jld67-shard-0 [primary] test> db.createCollection("soc")
{ ok: 1 }
Atlas atlas-9jld67-shard-0 [primary] test> show collections
soc
Atlas atlas-9jld67-shard-0 [primary] test>
```

Create(C):
Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection. MongoDB provides the following methods to insert documents into a collection:
    db.collectionname.insertOne({FieldName:"FieldValue"}) → to insert one document
      db.collectionname.insertMany([{},{},{}]) → to insert multiple documents at once

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

```
mongosh mongodb+srv://<credentials>@cluster0.zg0lgre.mongodb.net/
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.insertOne({name:"Harsha",roll:"21A91A05G6",section:"B"})
{
  acknowledged: true,
  insertedId: ObjectId("660f973116264a2d1d9e35b6")
}
Atlas atlas-9jld67-shard-0 [primary] test>
```

Read(R):
Read operations retrieve documents from a collection; i.e. query a collection for documents. MongoDB provides the following methods to read documents from a collection:

db.collectionname.find() → to see all the documents in a particular collection

db.Students.find({FieldName:"FieldValue"}) → To see the records/documents with FieldVal

db.Students.find().count() → To see the count of records/documents present in our collection

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

```
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.find()
[
  {
    _id: ObjectId("660f973116264a2d1d9e35b6"),
    name: 'Harsha',
    roll: '21A91A05G6',
    section: 'B'
  }
]
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.find({name:"Harsha"})
[
  {
    _id: ObjectId("660f973116264a2d1d9e35b6"),
    name: 'Harsha',
    roll: '21A91A05G6',
    section: 'B'
  }
]
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.find().count()
1
Atlas atlas-9jld67-shard-0 [primary] test>
```

Update(U):
Update operations modify existing documents in a collection. MongoDB provides the following methods to update documents of a collection:

db.collection.updateOne({FieldName:"FieldValue"},{$set:{ FieldName:"FieldValue"}}) → to update single document by finding with a unique value

db.collection.updateMany({}) → to update multiple documents

db.collection.replaceOne() → to replace a single document

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

```
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.updateOne({roll:"21A91A05G6"},{$set:{section:"A"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.find({name:"Harsha"})
[
  {
    _id: ObjectId("660f973116264a2d1d9e35b6"),
    name: 'Harsha',
    roll: '21A91A05G6',
    section: 'A'
  }
]
Atlas atlas-9jld67-shard-0 [primary] test>
```

Delete(D):

Delete operations remove documents from a collection. Delete operations target a single collection. MongoDB provides the following methods to delete documents of a collection:

db.collection.deleteOne({}) → delete a document where the field value matches with the name provided

db.collection.deleteMany({}) → deletes many documents where the field matches


```
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.deleteOne({roll:"21A91A05G6"})
{ acknowledged: true, deletedCount: 1 }
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.find({roll:"21A91A05G6"})

Atlas atlas-9jld67-shard-0 [primary] test>
```

You can also do the above operation using remove() method. The remove() method removes documents from the database. It can remove one or all documents from the collection that matches the given query expression. If you pass an empty document({}) in this method, then it will remove all documents from the specified collection.

Experiment-12: Course Name: MongoDB Essentials - A Complete MongoDB Guide
12-a) Module Name: Create and Delete Databases and Collections
Write MongoDB queries to Create and drop databases and collections.
Aim: To write MongoDB queries to Create and drop databases and collections.
Description:
Database:
A database is an organized collection of structured or unstructured information stored electronically on a machine locally or in the cloud. Databases are managed using a Database Management System (DBMS). The DBMS acts as an interface between the end user (or an application) and the database. Databases use a query language for storing or retrieving data.
Collection:
A collection is a grouping of MongoDB documents. Documents within a collection can have different fields. A collection is the equivalent of a table in a relational database system. A collection exists within a single database.
Queries:
Create Database:
Once you have access to a cluster via the shell, you can see all the databases in the cluster that you have access using "**show dbs**" command.

Note that admin and local are databases that are part of every MongoDB Cluster. There's no "create" command in shell. In order to create, "**use databasename**".

After creating and view databases, you still cannot see it. Why?? MongoDB only creates the database when you first store data in that database.

To find out in which database we're currently in, enter the command "**db**"

```
Atlas atlas-9jld67-shard-0 [primary] newdb> db
newdb
Atlas atlas-9jld67-shard-0 [primary] newdb> _
```

Create Collection:

There are two ways to create a collection.

You can create a collection using the **createCollection()** database method.

```
Atlas atlas-9jld67-shard-0 [primary] newdb> db.createCollection("students")
{ ok: 1 }
Atlas atlas-9jld67-shard-0 [primary] newdb> show collections
students
user
Atlas atlas-9jld67-shard-0 [primary] newdb> S
```

Or you can also create a collection during the insert process.

```
Atlas atlas-9jld67-shard-0 [primary] newdb> db.faculty.insertOne({name:"U.V.Ramesh",subject:"MST"})
{
  acknowledged: true,
  insertedId: ObjectId("66109cc8f9d7c1105912b051")
}
Atlas atlas-9jld67-shard-0 [primary] newdb> show collections
faculty
students
user
Atlas atlas-9jld67-shard-0 [primary] newdb> _
```

Drop Collection:

Removes a collection or view from the database. The method also removes any indexes associated with the dropped collection using **db.collection.drop()**.

```
Atlas atlas-9jld67-shard-0 [primary] newdb> db.faculty.drop()
true
Atlas atlas-9jld67-shard-0 [primary] newdb> db.students.drop()
true
Atlas atlas-9jld67-shard-0 [primary] newdb> show collections
user
Atlas atlas-9jld67-shard-0 [primary] newdb>
```

Drop Database:

The **dropDatabase** command drops the current database, deleting the associated data files. You have to be using the database which you want to delete. If you want to delete "**newdb**", then first enter "**use new db**".

```
Atlas atlas-9jld67-shard-0 [primary] test> use newdb
switched to db newdb
Atlas atlas-9jld67-shard-0 [primary] newdb> db.dropDatabase()
{ ok: 1, dropped: 'newdb' }
Atlas atlas-9jld67-shard-0 [primary] newdb> use test
switched to db test
Atlas atlas-9jld67-shard-0 [primary] test> show dbs
sample_mflix   113.42 MiB
test            64.00 KiB
admin          288.00 KiB
local           80.50 GiB
Atlas atlas-9jld67-shard-0 [primary] test> _
```

12-b) Module Name: Introduction to MongoDB Queries
Write MongoDB queries to work with records using find(), limit(), sort(), createIndex(), aggregate().
Aim: To write MongoDB queries to work with records using find(), limit(), sort(), createIndex(), aggregate().
Description and Queries:
find(): To select data from a collection in MongoDB, we can use the find() method. This method accepts a query object. If left empty, all documents will be returned.

```
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.insertMany([{name:"Angular"},{name:"JS"},{name:"HTML"},{name:"CSS"}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("66109ebff9d7c1105912b052"),
    '1': ObjectId("66109ebff9d7c1105912b053"),
    '2': ObjectId("66109ebff9d7c1105912b054"),
    '3': ObjectId("66109ebff9d7c1105912b055")
  }
}
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.find({name:"Angular"})
[ { _id: ObjectId("66109ebff9d7c1105912b052"), name: 'Angular' } ]
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.find()
[
  { _id: ObjectId("66109ebff9d7c1105912b052"), name: 'Angular' },
  { _id: ObjectId("66109ebff9d7c1105912b053"), name: 'JS' },
  { _id: ObjectId("66109ebff9d7c1105912b054"), name: 'HTML' },
  { _id: ObjectId("66109ebff9d7c1105912b055"), name: 'CSS' }
]
Atlas atlas-9jld67-shard-0 [primary] test> _
```

limit(): Use the limit() method on a cursor to specify the maximum number of documents the cursor will return. limit() is analogous to the LIMIT statement in a SQL database.

```
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.find().limit(0)
[
  { _id: ObjectId("66109ebff9d7c1105912b052"), name: 'Angular' },
  { _id: ObjectId("66109ebff9d7c1105912b053"), name: 'JS' },
  { _id: ObjectId("66109ebff9d7c1105912b054"), name: 'HTML' },
  { _id: ObjectId("66109ebff9d7c1105912b055"), name: 'CSS' }
]
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.find().limit(1)
[ { _id: ObjectId("66109ebff9d7c1105912b052"), name: 'Angular' } ]
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.find().limit(2)
[
  { _id: ObjectId("66109ebff9d7c1105912b052"), name: 'Angular' },
  { _id: ObjectId("66109ebff9d7c1105912b053"), name: 'JS' }
]
Atlas atlas-9jld67-shard-0 [primary] test>
```

sort(): Specifies the order in which the query returns matching documents. You must apply sort() to the cursor before retrieving any documents from the database.

Sorting data in ascending order:

```
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.find().sort({name:1})
[
  { _id: ObjectId("66109ebff9d7c1105912b052"), name: 'Angular' },
  { _id: ObjectId("66109ebff9d7c1105912b055"), name: 'CSS' },
  { _id: ObjectId("66109ebff9d7c1105912b054"), name: 'HTML' },
  { _id: ObjectId("66109ebff9d7c1105912b053"), name: 'JS' }
]
```

Sorting data in descending order:

```
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.find().sort({name:-1})
[
  { _id: ObjectId("66109ebff9d7c1105912b053"), name: 'JS' },
  { _id: ObjectId("66109ebff9d7c1105912b054"), name: 'HTML' },
  { _id: ObjectId("66109ebff9d7c1105912b055"), name: 'CSS' },
  { _id: ObjectId("66109ebff9d7c1105912b052"), name: 'Angular' }
]
```

createIndex(): If you had a collection with thousands of documents with no indexes, and then you query to find certain documents, then in such case MongoDB would need to scan the entire collection to find the documents. But if you had indexes, MongoDB would use these indexes to limit the number of documents that had to be searched in the collection.

In the example below, the Employeeid "1" and EmployeeCode "AA" are used to index the documents in the collection. So when a query search is made, these indexes will be used to quickly and efficiently find the required documents in the collection.

So even if the search query is based on the EmployeeCode "AA", that document would be returned.

```
{
    Employeeid    : 1
    EmployeeCode  : AA          ←── An example of an index
    EmployeeName  : "Joe"            for a collection
    Awards        : 1
    Country       : India

}
```

Creating an Index in MongoDB is done by using the "createIndex" method.

The following example shows how add index to collection. Let's assume that we have our same Employee collection which has the Field names of "Employeeid" and "EmployeeName".

```
C:\Windows\system32\cmd.exe - mongo.exe
> db.Employee.createIndex({Employeeid:1})
```

Using the createIndex method on Employeeid

This indicates to sort the field values in ascending order

The createIndex method now takes into account multiple Field values which will now cause the index to be created based on the "Employeeid" and "EmployeeName". The Employeeid:1 and EmployeeName:1 indicates that the index should be created on these 2 field values with the :1 indicating that it should be in ascending order.

aggregate(): It collects values from various documents and groups them together and then performs different types of operations on that grouped data like sum, average, minimum, maximum, etc to return a computed result. It is similar to the aggregate function of SQL.