Experiment-1

1a)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

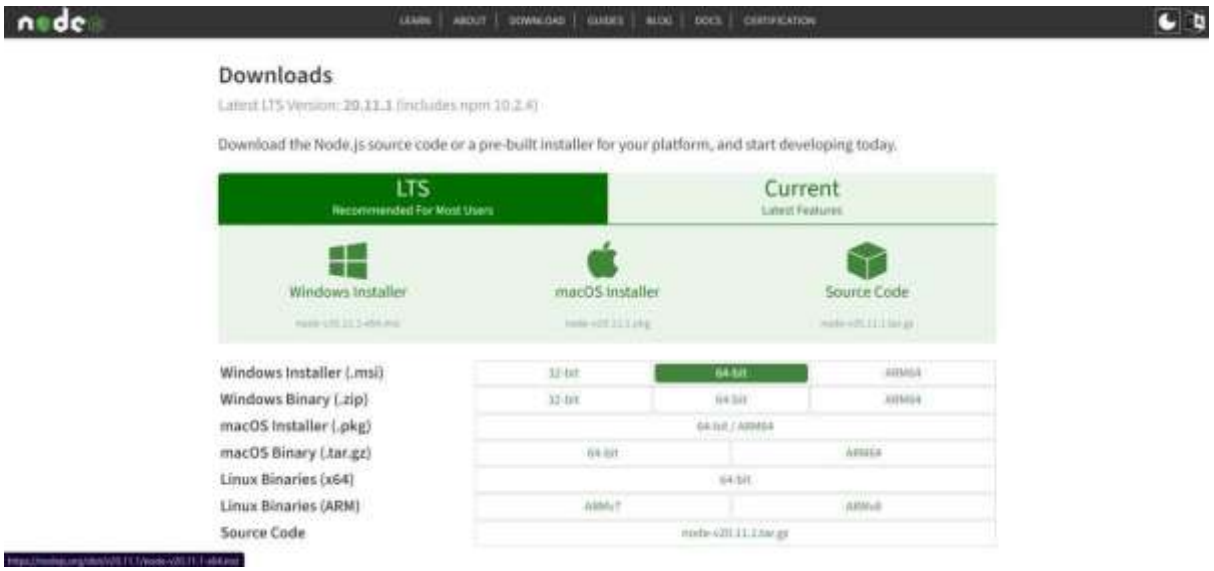
First of all, we need to Install Node.js and Visual Studio Code from their respective official websites.

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.



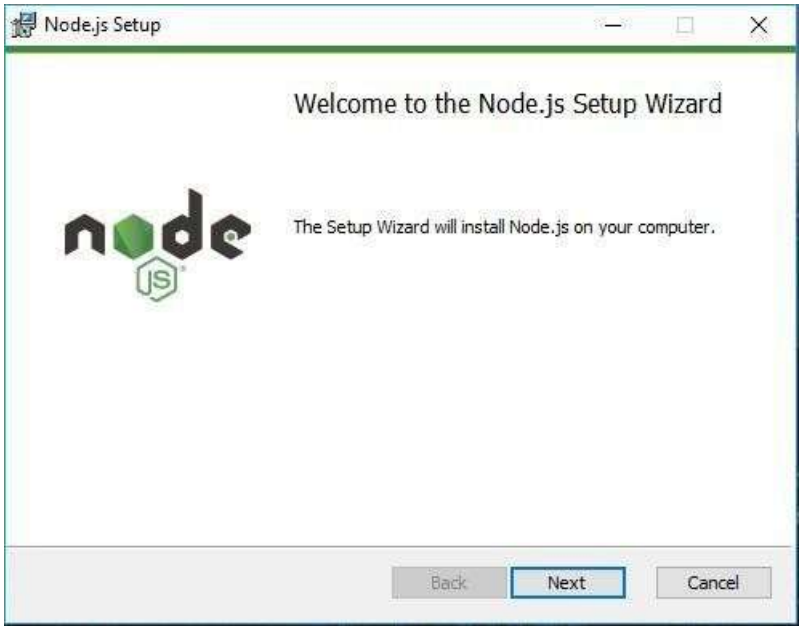
Step-2: Running the Node.js installer

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.

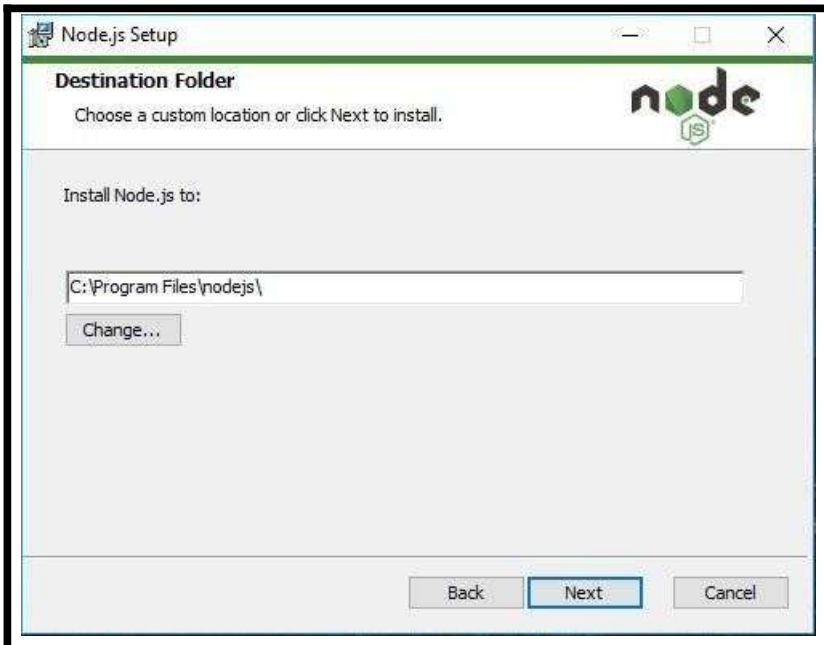
• Welcome To Node.js Setup Wizard. Select “Next”



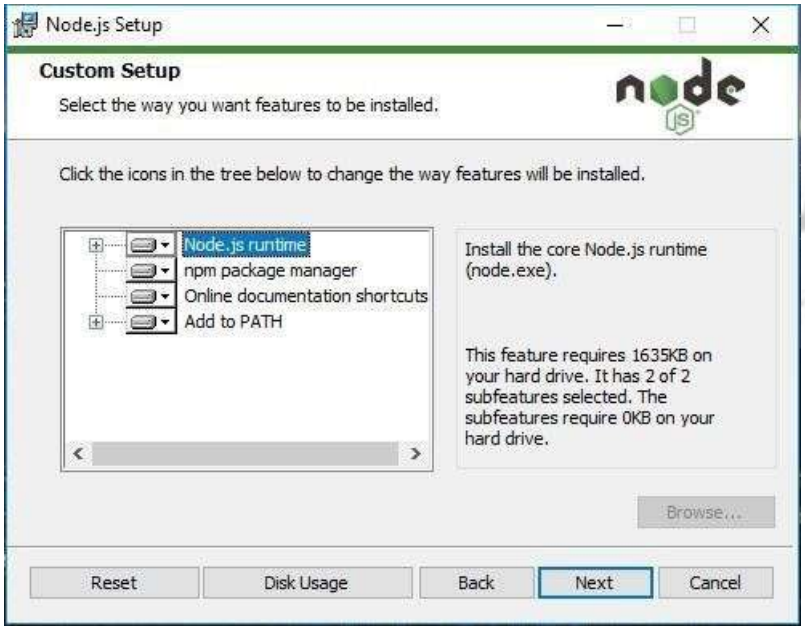
• After clicking “Next”, End-User License Agreement (EULA) will open.

• Check “I accept the terms in the License Agreement”. Select “Next”.

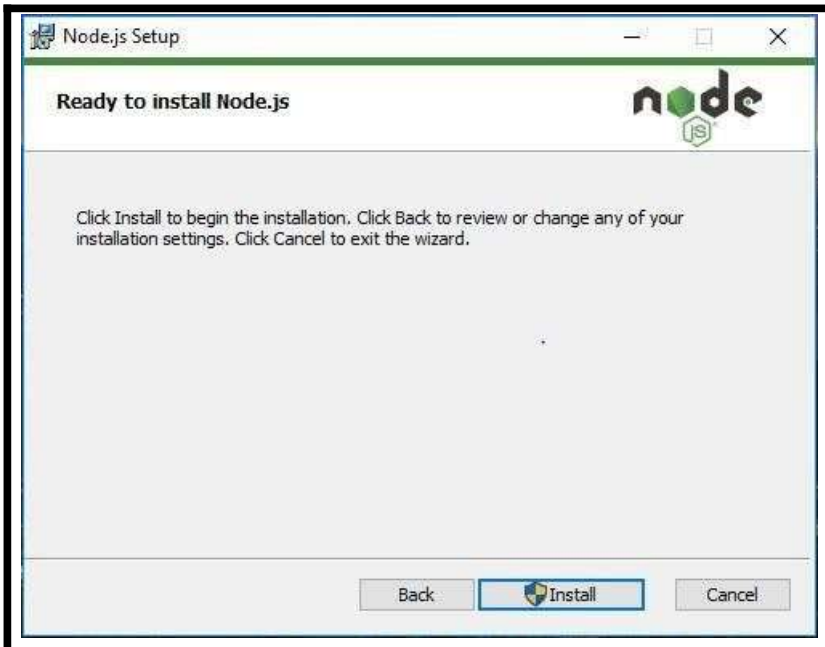
• Destination Folder. Set the Destination Folder where you want to install Node.js & Select “Next”.



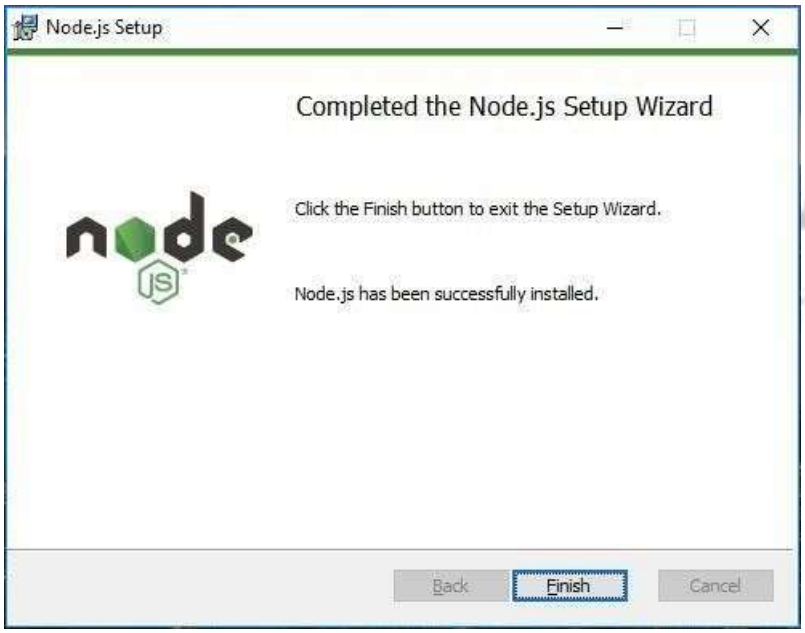
• Custom Setup. Select “Next”



• 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

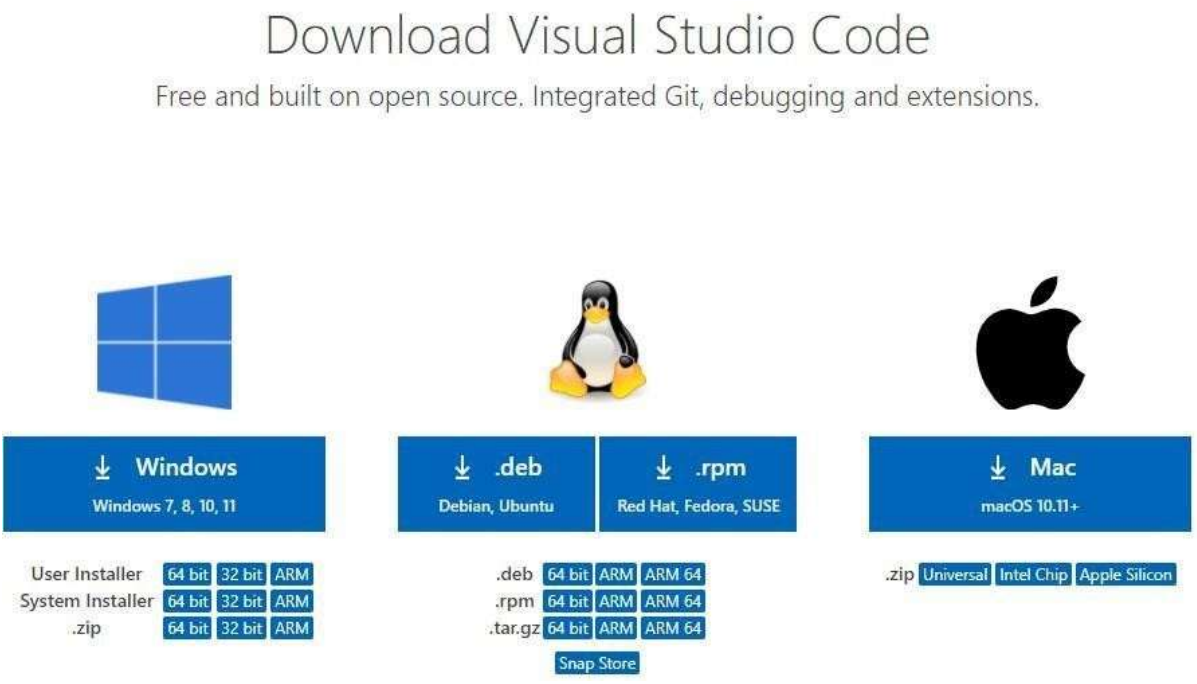
C:\Users\Admin>npm -v

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

C:\Users\harsh>node -v
v20.11.0

C:\Users\harsh>npm -v
10.2.4

C:\Users\harsh>
```

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.

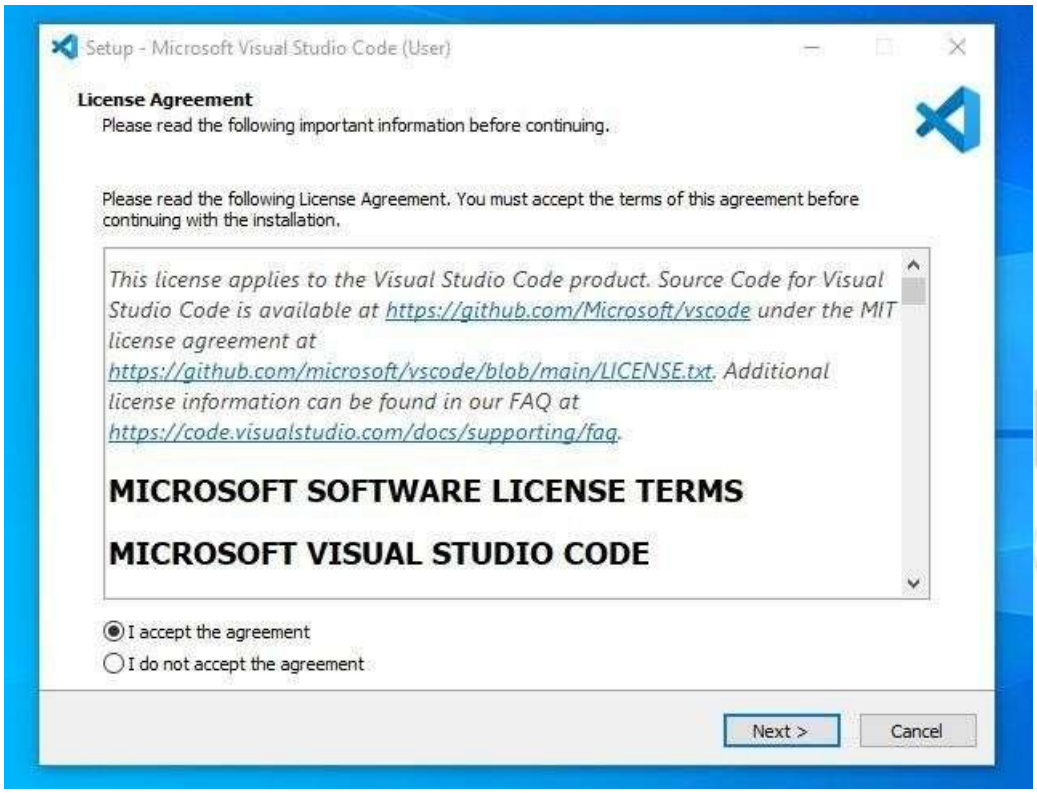
Step 2: Press the "Download for Windows" button on the website to start the download of the Visual Studio Code Application.



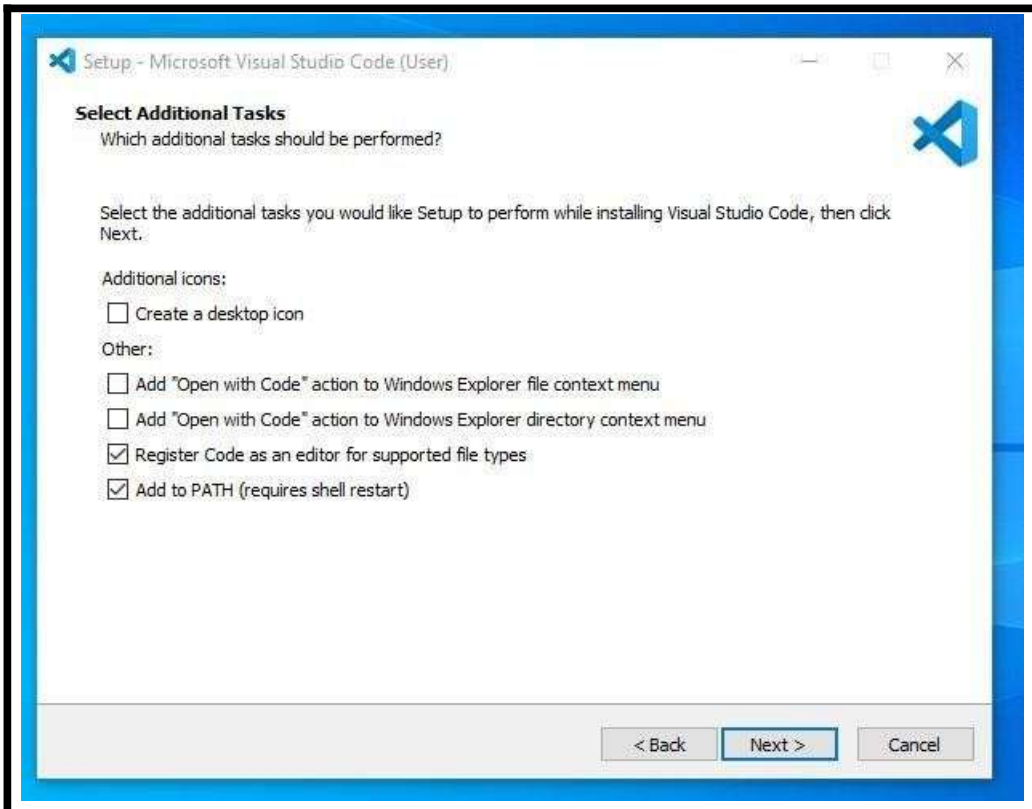
Step 3: When the download finishes, then the Visual Studio Code icon appears in the downloads folder.

Step 4: Click on the installer icon to start the installation process of the Visual Studio Code.

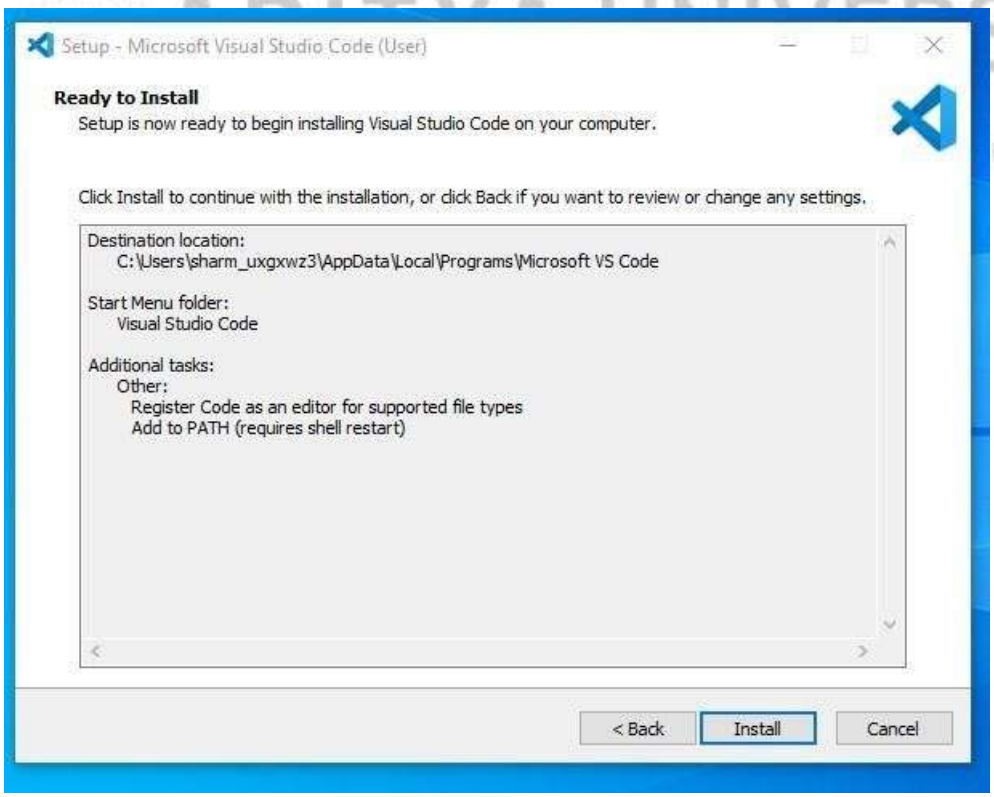
Step 5: After the Installer opens, it will ask you for accepting the terms and conditions of the Visual Studio Code. Click on I accept the agreement and then click the Next button.



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.

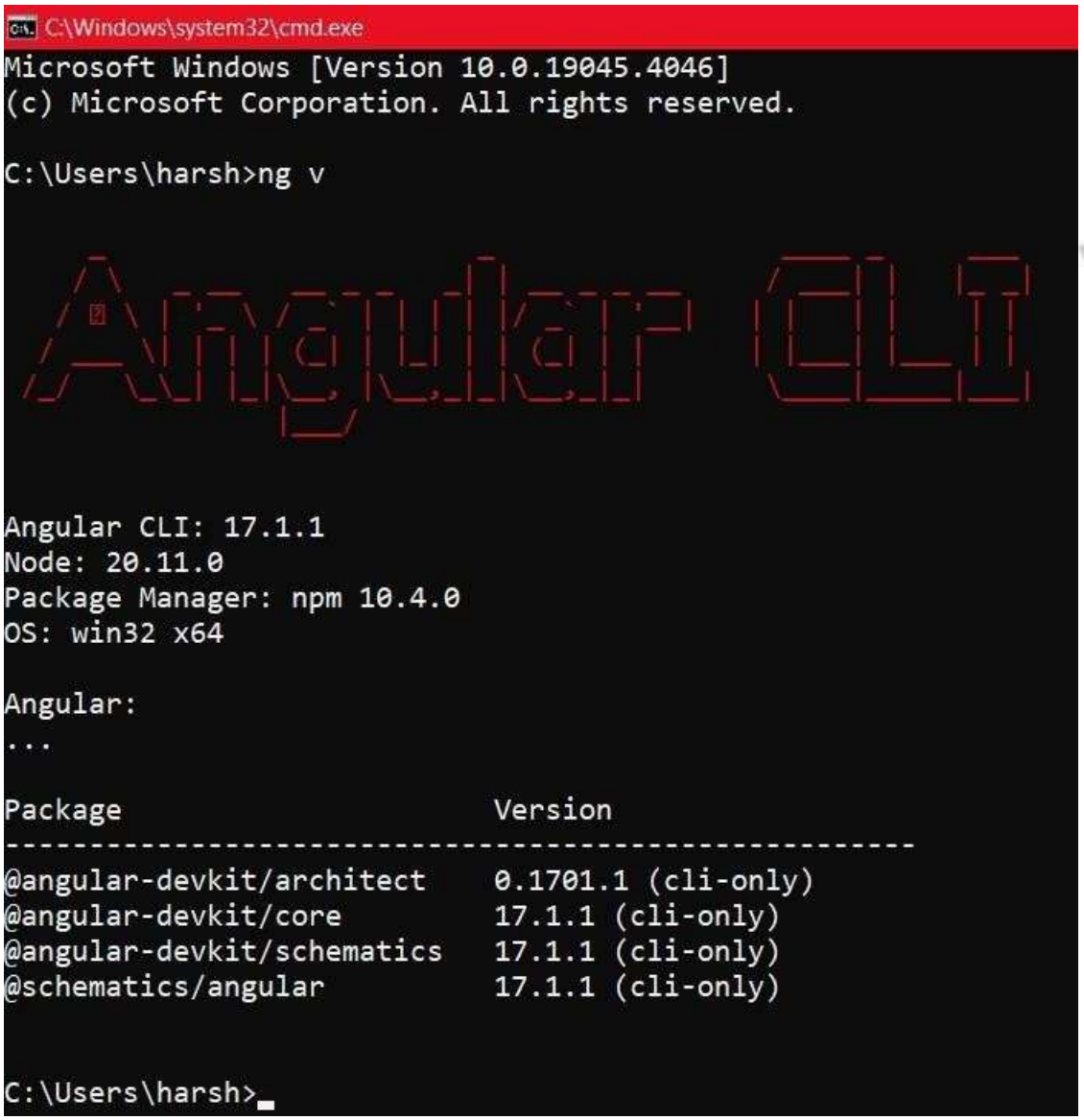
Step 10: After the previous step, the Visual Studio Code window opens successfully. Now you can create a new file in the Visual Studio Code window and choose a language of yours to begin your programming journey!

Steps to install Angular CLI:

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

npm install -g @angular/cli

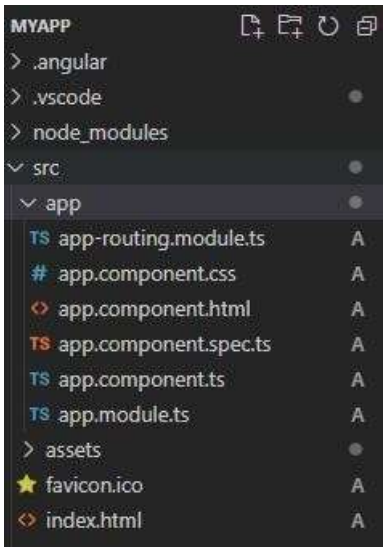
Test successful installation of Angular CLI using the following command Note: Sometimes additional dependencies might throw an error during CLI installation but still check whether CLI is installed or not using the following command. If the version gets displayed, you can ignore the errors.



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.



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

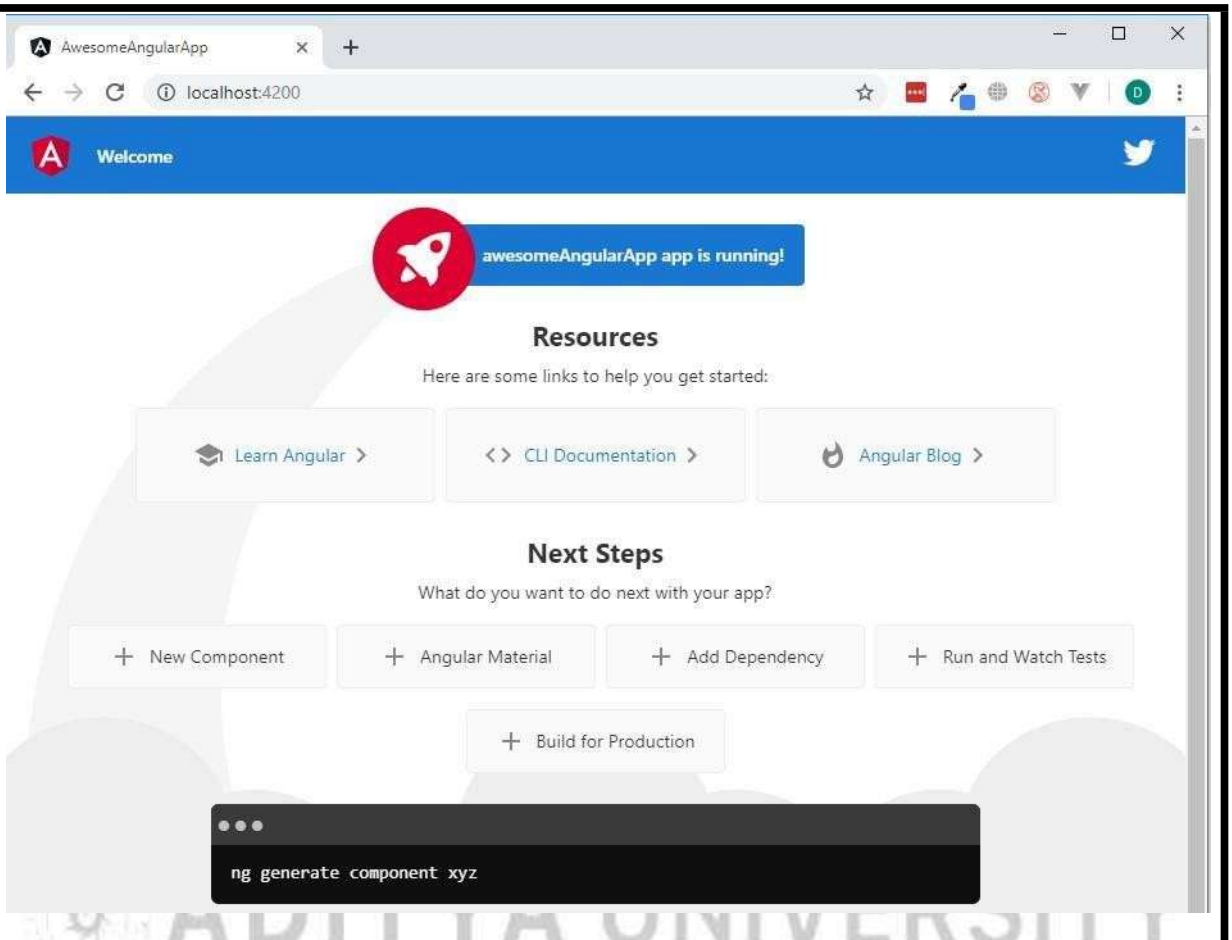
```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS

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)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS

PS F:\SOC-II Lab\AngularApp\myapp> ng serve --o --port 3000

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. [3.421 seconds]
Watch mode enabled. Watching for file changes...
  → Local:   http://localhost:3000/
  → press h + enter to show help
```



1b)Aim:

Components and Modules. Create a new component called hello and render Hello Angular on the page.

Program:

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:

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

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

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


Output:

1c)Aim:

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

Program:

hello.component.ts:

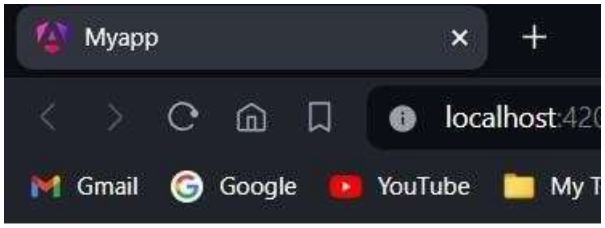
```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './hello.component.html',
styleUrls: ['./hello.component.css']
})
export class HelloComponent {
Message: string="Good Morning!";
changeName(){
this.Message="Have a nice day!";
}
}
```

hello.component.html:

```
<p>{{Message}}</p>
<h2 (click)="changeName()">Click here to change</h2>
```

Output:

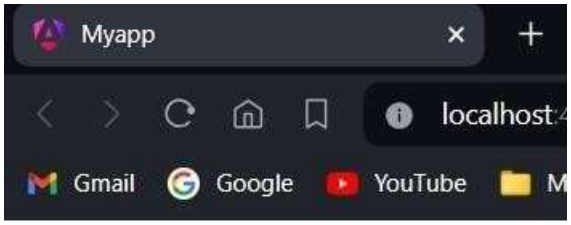
Before clicking:



Good Morning!

Click here to change

After Clicking:



Have a nice day!

Click here to change

1d)Aim:

To progressively build the PoolCarz application.

Program:

hello.component.ts:

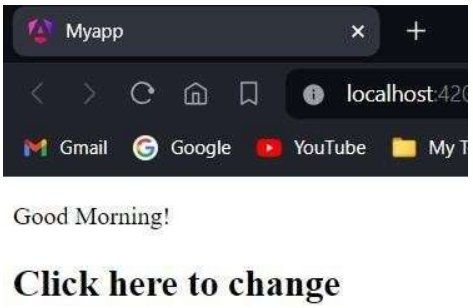
```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './hello.component.html',styleUrls:
['./hello.component.css']
})
export class HelloComponent { Message:
string="Good Morning!";changeName(){
this.Message="Have a nice day!";
}
}
```

hello.component.html:

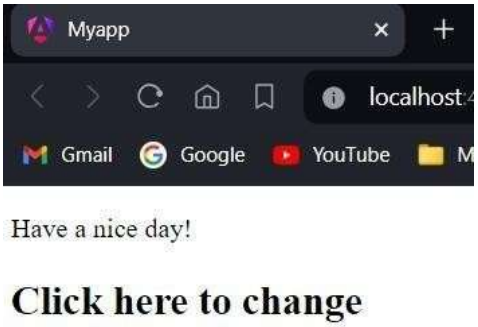
```
<p>{{Message}}</p>
<h2 (click)="changeName()">Click here to change</h2>
```

Output:

Before clicking:



After Clicking:



Experiment-2

2a)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.

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:

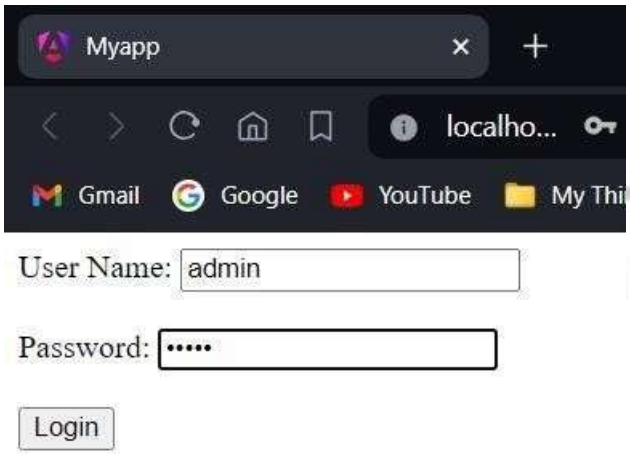
```
<div *ngIf="!submitted">
<form action="">
<label for="text">User Name: </label>
<input type="text" #username /><br /><br />
<label for="password">Password: </label>
<input type="password" #password /><br /><br />
</form>
<button (click)="onSubmit(username.value,password.value)">Login</button>
</div>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

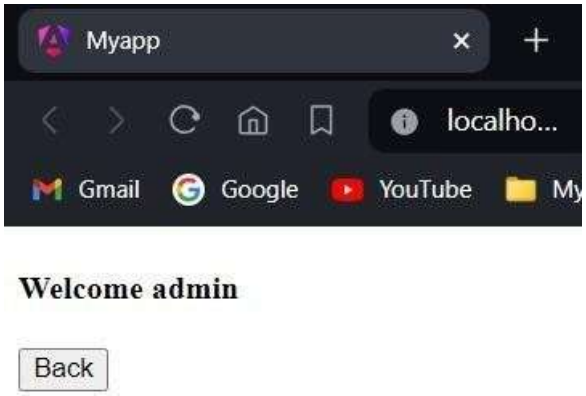
app.module.ts:

```
import { 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 { }
```

Output:





2b)Aim:

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

Program:

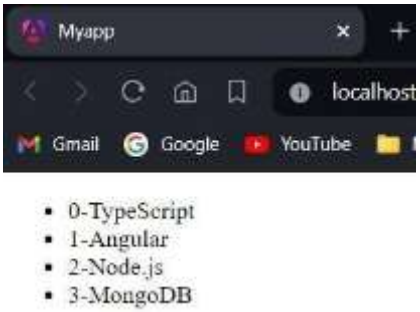
app.component.ts:

```
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:



2c) Aim:

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

Program:

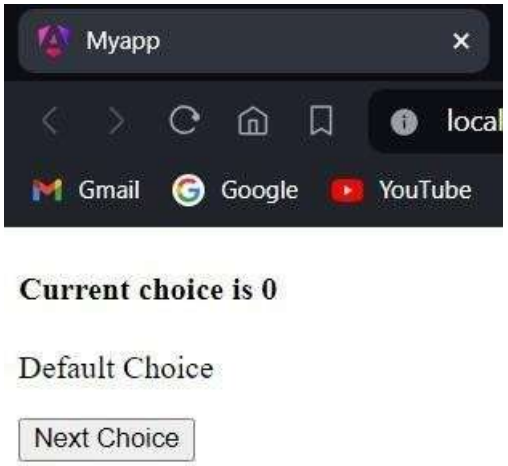
app.component.ts:

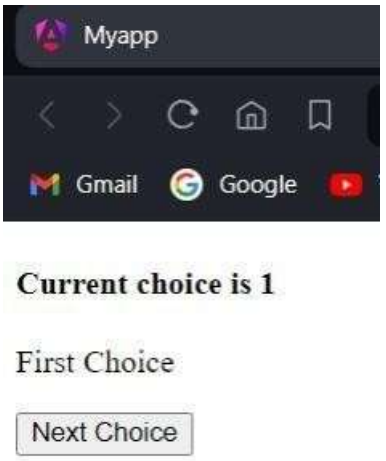
```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
choice=0;
nextChoice(){
this.choice++;
}
}
```

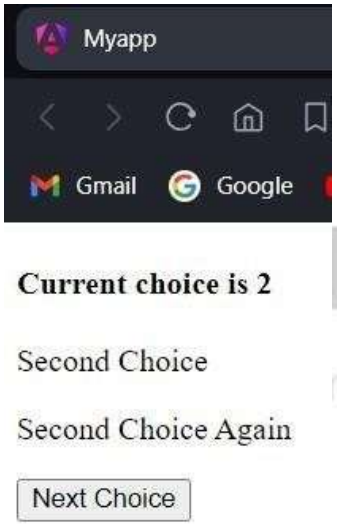
app.component.html:

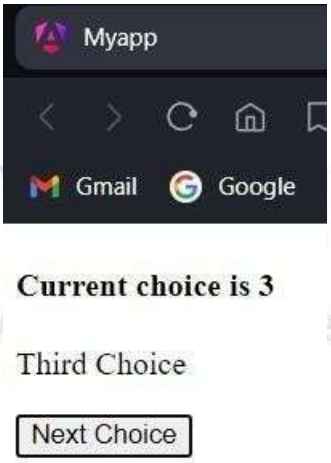
```
<h4>Current choice is {{choice}}</h4>
<div [ngSwitch]="choice">
<p *ngSwitchCase="1">First Choice</p>
<p *ngSwitchCase="2">Second Choice</p>
<p *ngSwitchCase="3">Third Choice</p>
<p *ngSwitchCase="2">Second Choice Again</p>
<p *ngSwitchDefault>Default Choice</p>
</div>
<div><button (click)="nextChoice()">Next Choice</button></div>
```

Output:


Myapp
local
Gmail
Google
YouTube
Current choice is 0
Default Choice
Next Choice


Myapp
Gmail
Google
Current choice is 1
First Choice
Next Choice


Myapp
Gmail
Google
Current choice is 2
Second Choice
Second Choice Again
Next Choice


Myapp
Gmail
Google
Current choice is 3
Third Choice
Next Choice

2d)Aim:

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

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

3a)Aim:

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

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

3b)Aim:

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

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:

3c)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.

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:

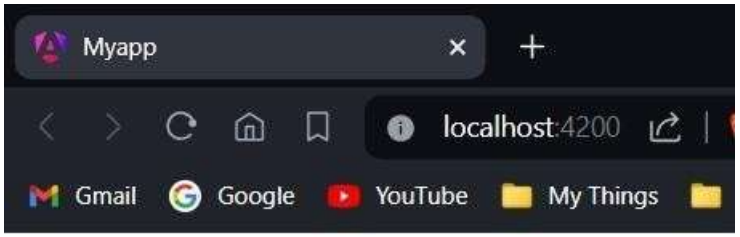
```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
myMessage="Hello, It is my first custom attribute directive named message!";
}
```

app.component.css:

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

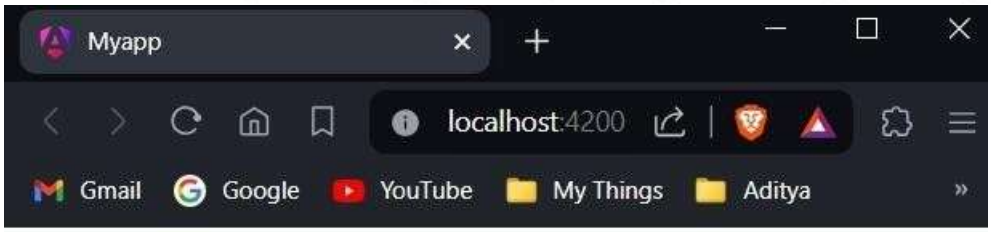
```
p{
color: #ff0080;
font-family: Arial, Helvetica, sans-serif;
font-size: 150%;
}
```

Output:



Attribute Directive

Click Here



Attribute Directive

Hello, It is my first custom attribute directive named message!

Experiment-4

4a)Aim:

To bind an image with class property using property 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:

4b)Aim:

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

Program:

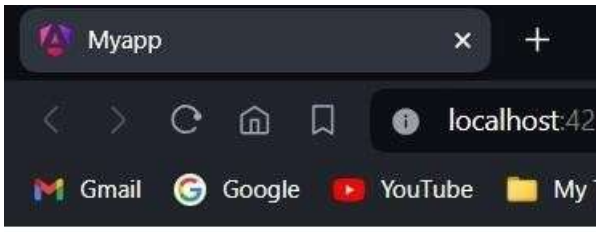
app.component.ts:

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



Attribute Binding

First		Second
Third	Fourth	Fifth
Sixth	Seventh	Eighth

4c)Aim:

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

Program:

Style Binding:

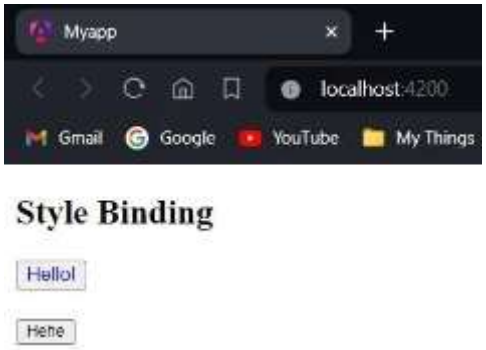
app.component.ts:

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

Output:



Event Binding:

app.component.ts:

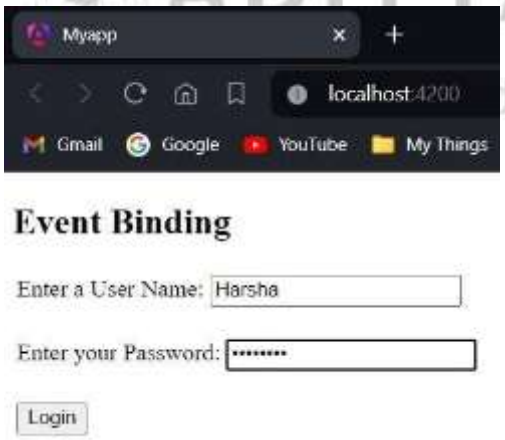
```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
onSubmit(){
var un=(document.getElementById("uname") as HTMLInputElement).value;
(document.getElementById("id1") as HTMLInputElement).innerHTML="Your Name is:
"+un;
var p=(document.getElementById("pwd") as HTMLInputElement).value;
(document.getElementById("id2") as HTMLInputElement).innerHTML="Your Password
is: "+p;
}
}
```

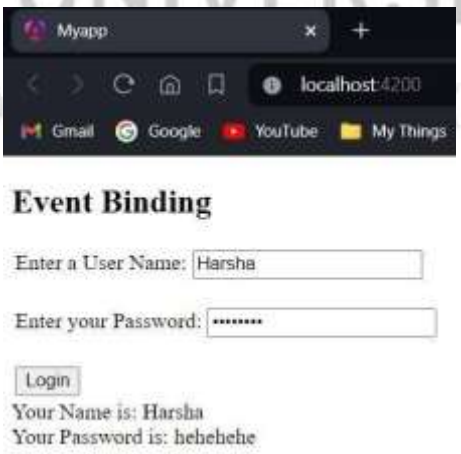
app.component.html:

```
<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

5a)Aim:

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

Program:

pipesexp.component.ts:

```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './pipesexp.component.html',
styleUrl: './pipesexp.component.css'
})
export class PipesexpComponent {
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

5b)Aim:

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

Program:

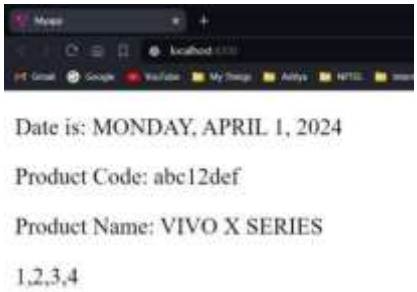
pipesexp.component.ts:

```
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:



5c)Aim:

Load CourseslistComponent in the root component when a user clicks on the Viewcourses list button.

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,
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.visi
ble;
}
}
```

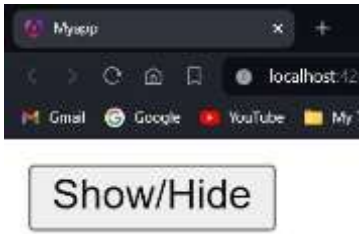
course-list.component.html:

```
<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:



Experiment-6

6a)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.

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]);
}
}
```

```
}
}
```

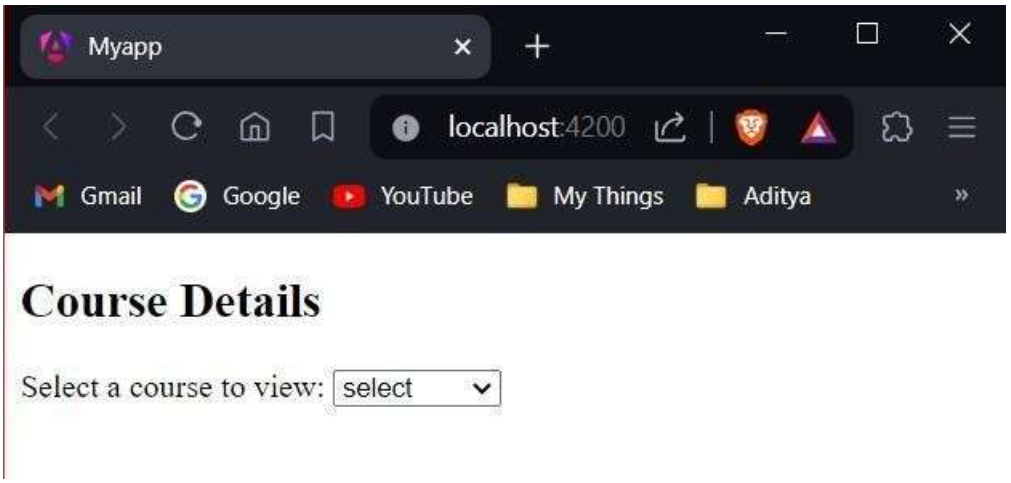
course-list.component.html:

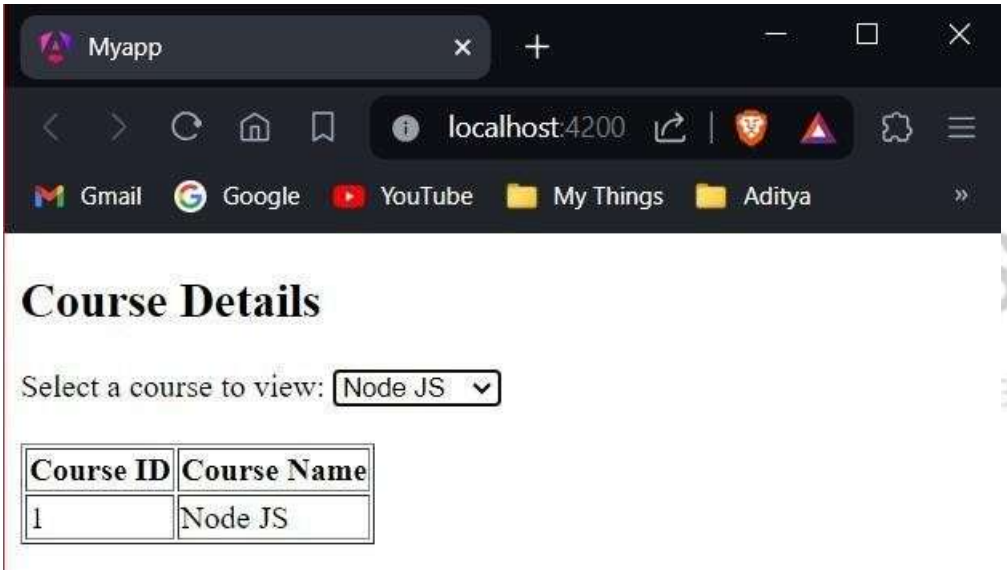
```
<table border="1" *ngIf="course.length">
<thead>
<tr>
<th>Course ID</th>
<th>Course Name</th>
</tr>
</thead>
<tbody>
<tr *ngFor="let c of course">
<td>{{ c.courseId }}</td>
<td>{{ c.courseName }}</td>
</tr>
</tbody>
</table>
```


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:


Myapp
localhost:4200
Gmail
Google
YouTube
My Things
Aditya
Course Details
Select a course to view:
select


Myapp
localhost:4200
Gmail
Google
YouTube
My Things
Aditya
Course Details
Select a course to view:
Node JS
Course ID
Course Name
1
Node JS

6b)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.

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);
}
}
```

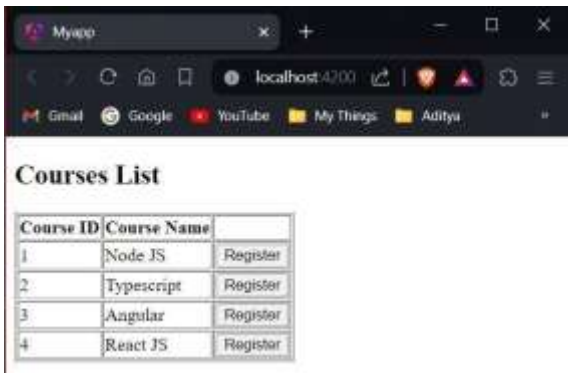
app.component.html:

```
<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:



6c)Aim:

Apply ShadowDOM and None encapsulation modes to components.

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 { 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'],
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:

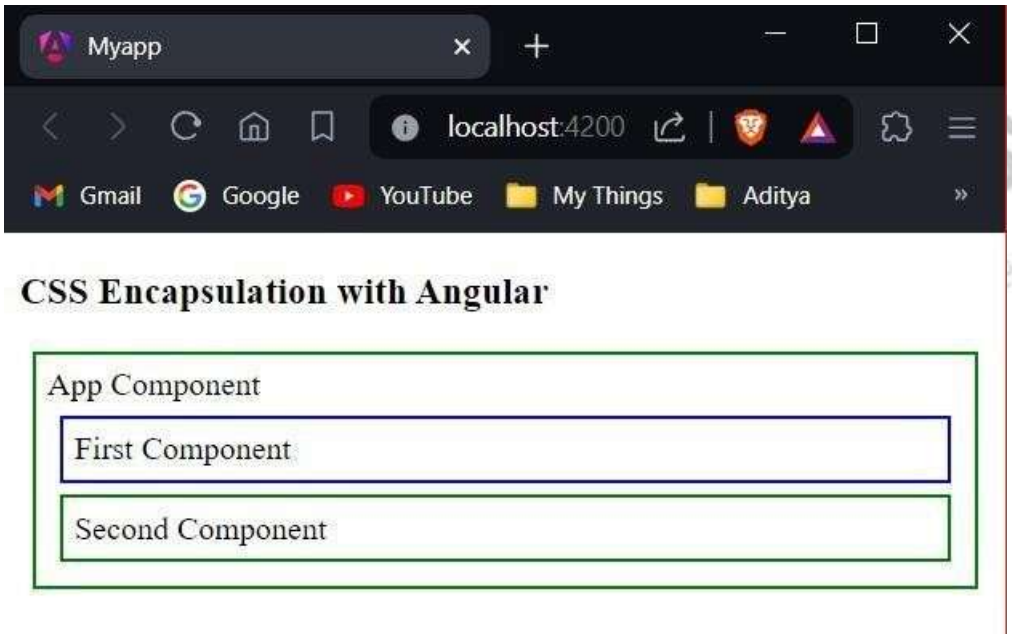
```
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css'],
encapsulation: ViewEncapsulation.None
})
```

```
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:



6d)Aim:

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

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:

```
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:

```
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));
}
}
```

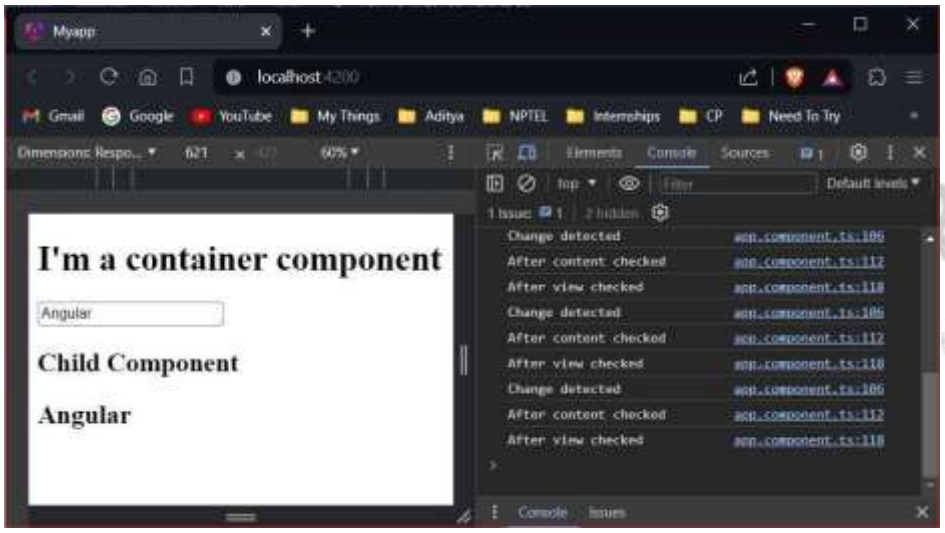
app.component.html:

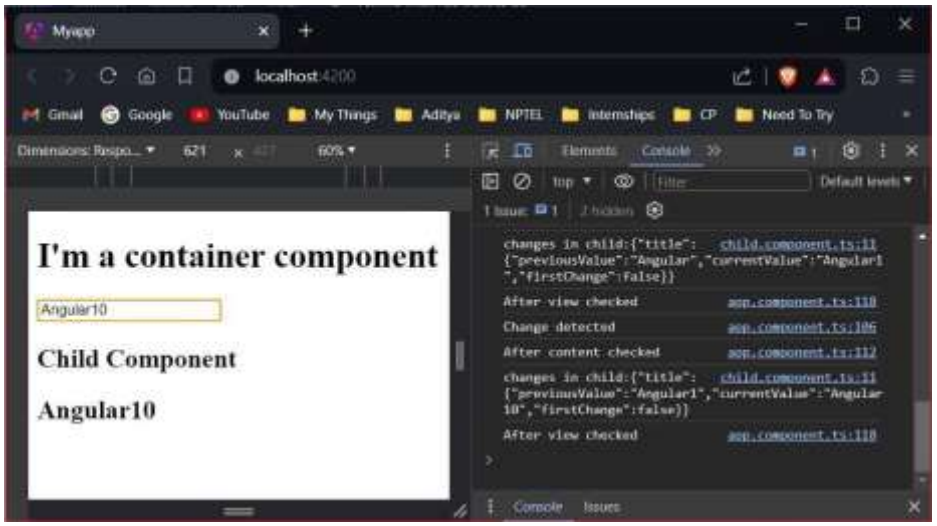
```
<div>
<h1>I'm a container component</h1>
<input type="text" [(ngModel)]="data" />
<app-child [title]="data"></app-child>
</div>
```

child.component.html:

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

Output:





Experiment-7

7a)Aim:

To create a course registration form as a template-driven form

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>
```

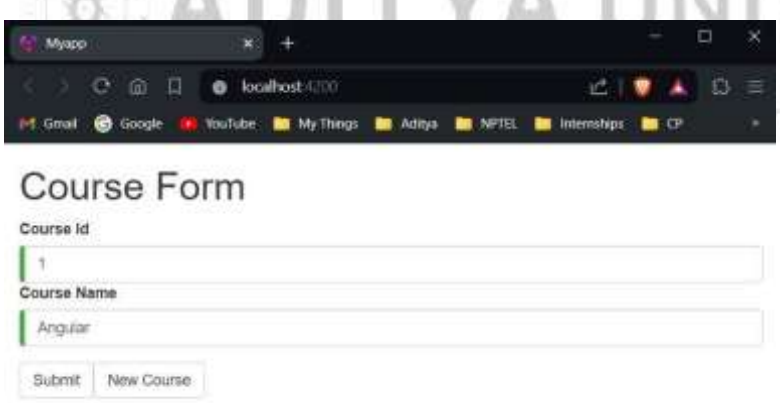
course-form.component.css:

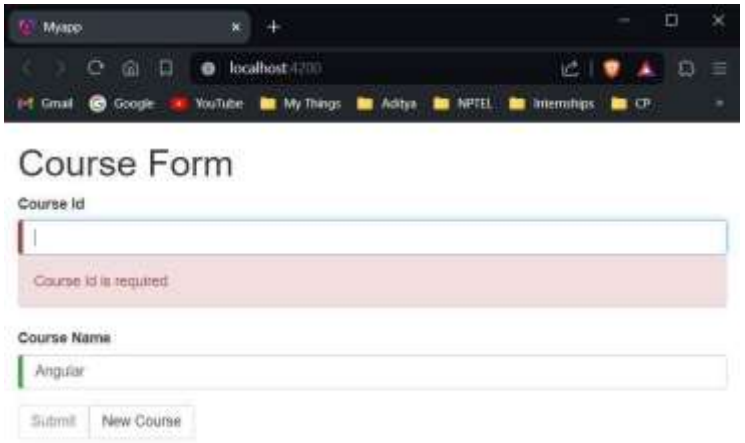
```
input.ng-valid[required] {
border-left: 5px solid #42A948;
}
input.ng-dirty.ng-invalid:not(form) {
border-left: 5px solid #a94442;
}
```

app.component.html:

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

Output:





7b)Aim:

To create an employee registration form as a reactive form.

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>
<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:

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

app.component.ts:

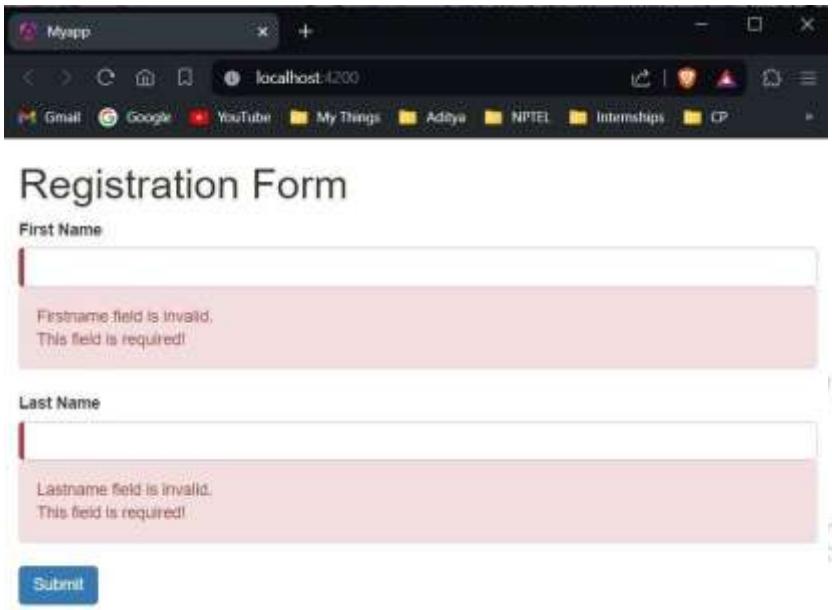
```
<app-registration-form></app-registration-form>
```

app.module.ts:

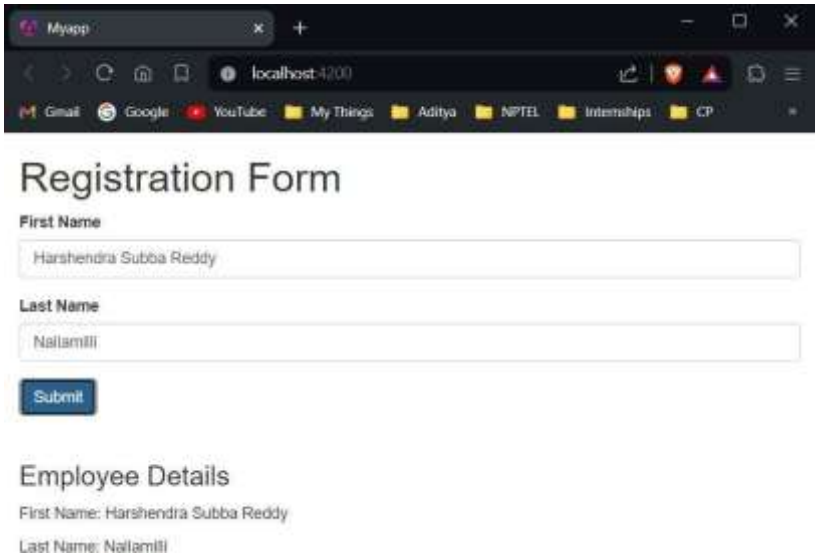
```
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ReactiveFormsModule } from '@angular/forms';
import { RegistrationFormComponent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



7c)Aim:

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

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>
<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:

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

```
border-left: 5px solid #a94442;
}
```

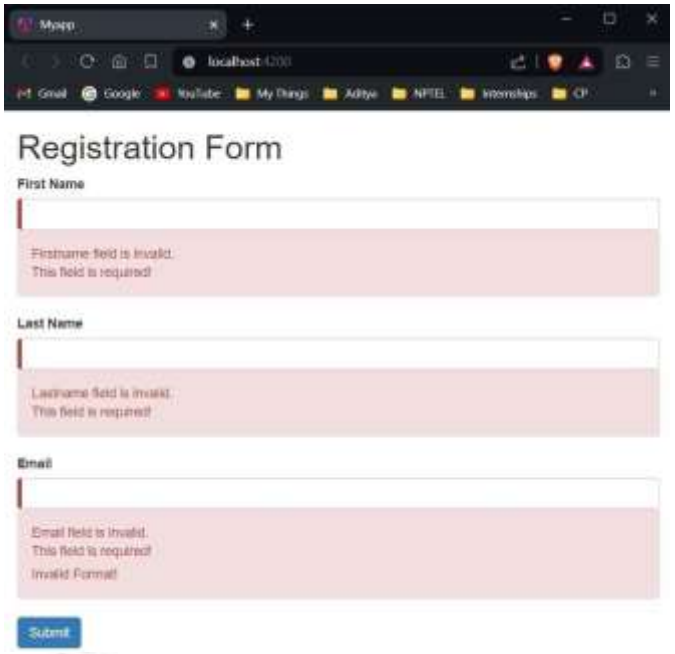
app.component.ts:

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

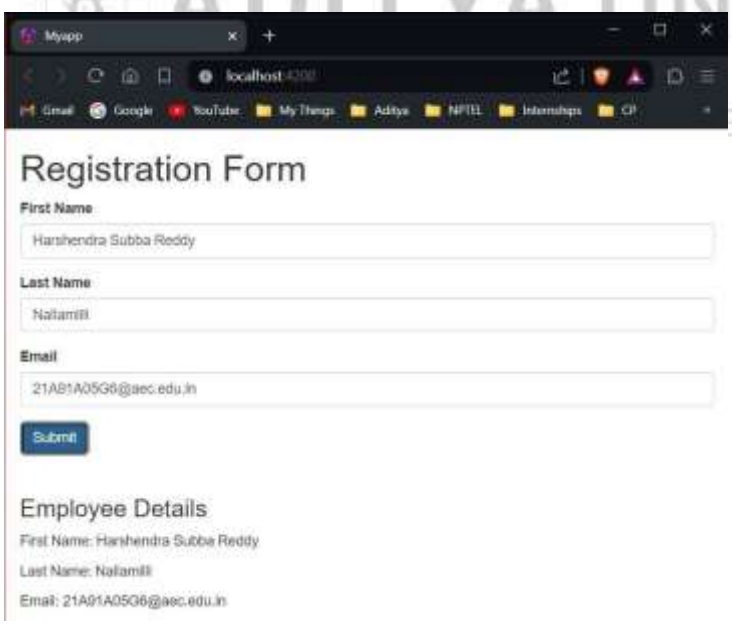
app.module.ts:

```
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

8a)Aim:

To create a custom validator for the email field in the course registration form.

Program:

email.validator.ts:

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

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:

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

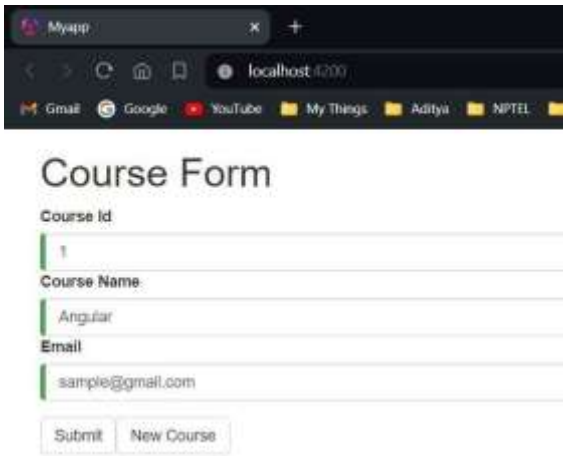
app.module.ts:

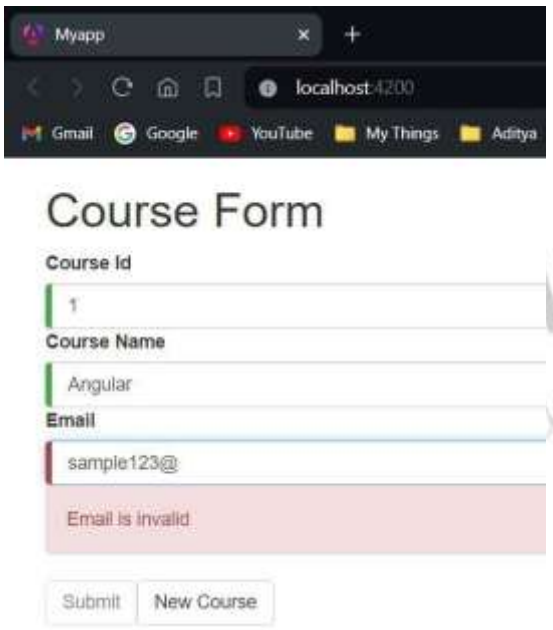
```
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';
import { CourseFormComponent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

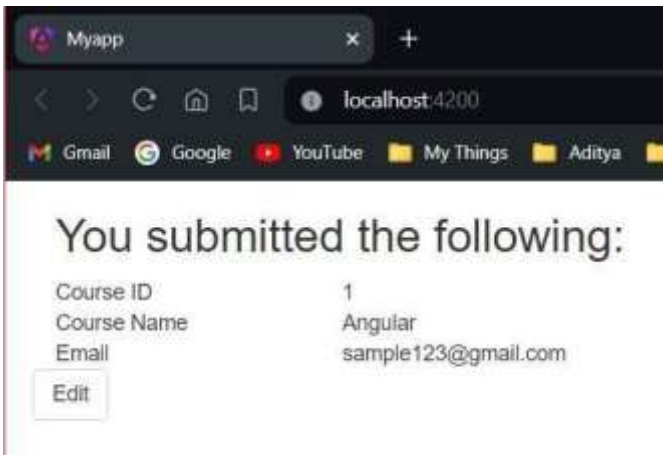
app.component.html:

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

Output:







8b)Aim:

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.

Program:

book.ts:

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

books-data.ts:

```
import { Book } from './book';
export let BOOKS: Book[] = [
{ id: 1, name: 'HTML 5' },
{ id: 2, name: 'CSS 3' },
{ id: 3, name: 'Java Script' },
{ id: 4, name: 'Node.js' },
{ id: 5, name: '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;
}
```

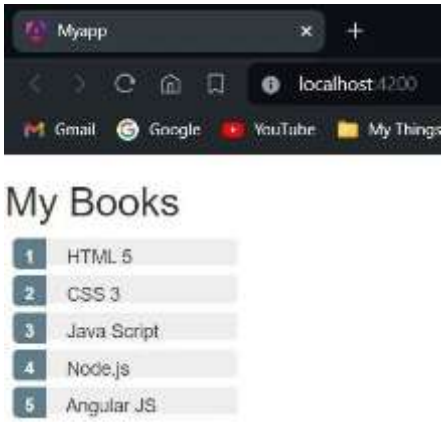
app.component.html:

```
<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 { }
```


Output:



8c)Aim:

To create and use an observable in Angular.

Program:

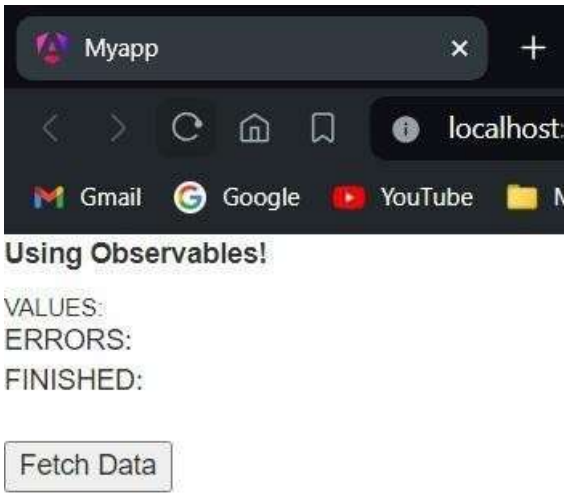
app.component.ts:

```
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);
}
}
```

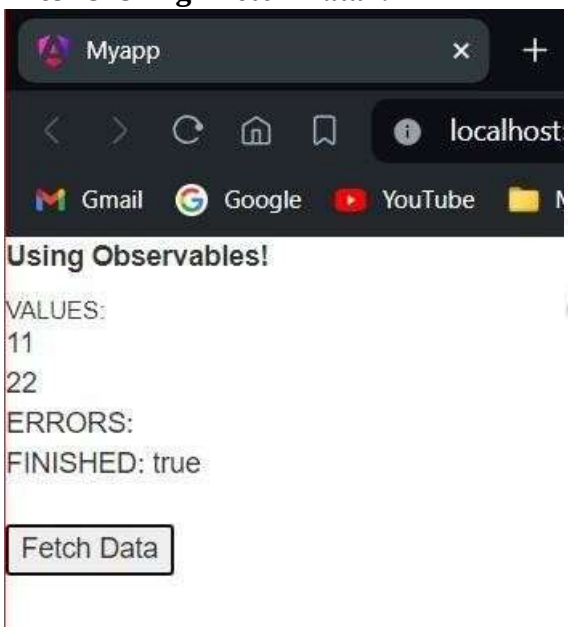
app.component.html:

```
<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

Aim:

To create an application for Server Communication using HttpClient

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));
}
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:

```
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;
: 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
</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

Aim:

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

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'})
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:

```
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:

EXPERIMENT-11

11a) Aim:

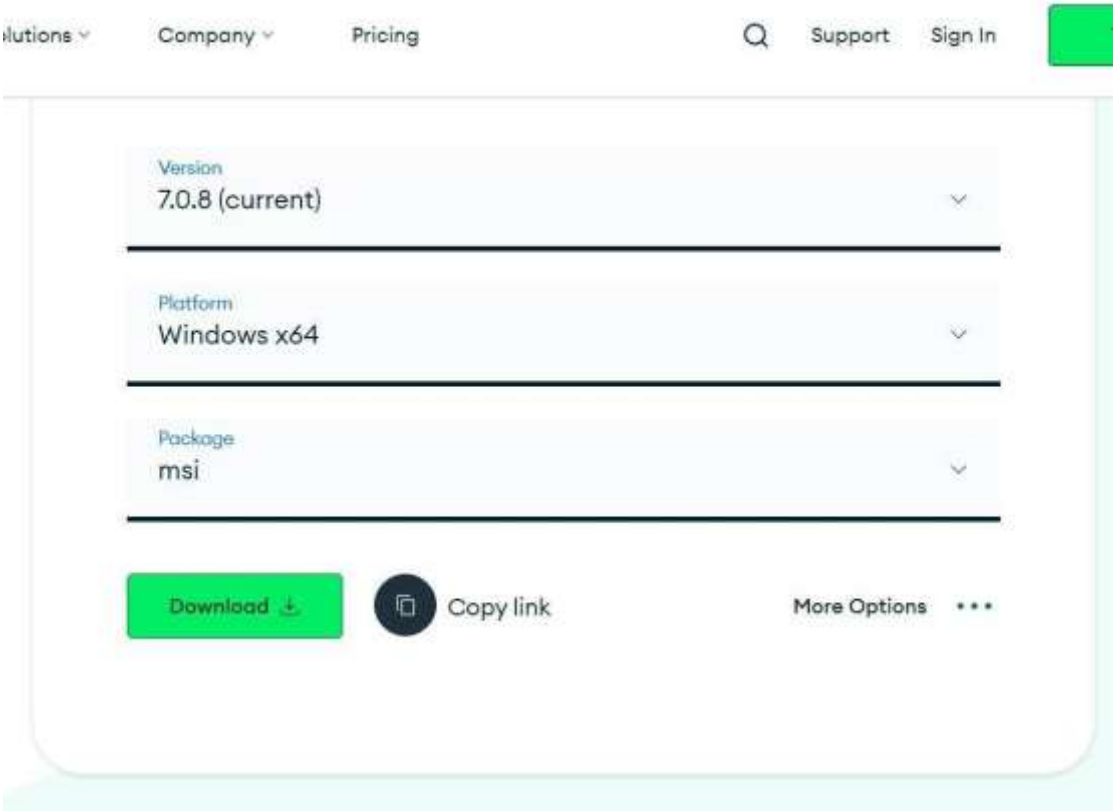
To Install MongoDB and configure ATLAS

Description:

Installation of MongoDB:

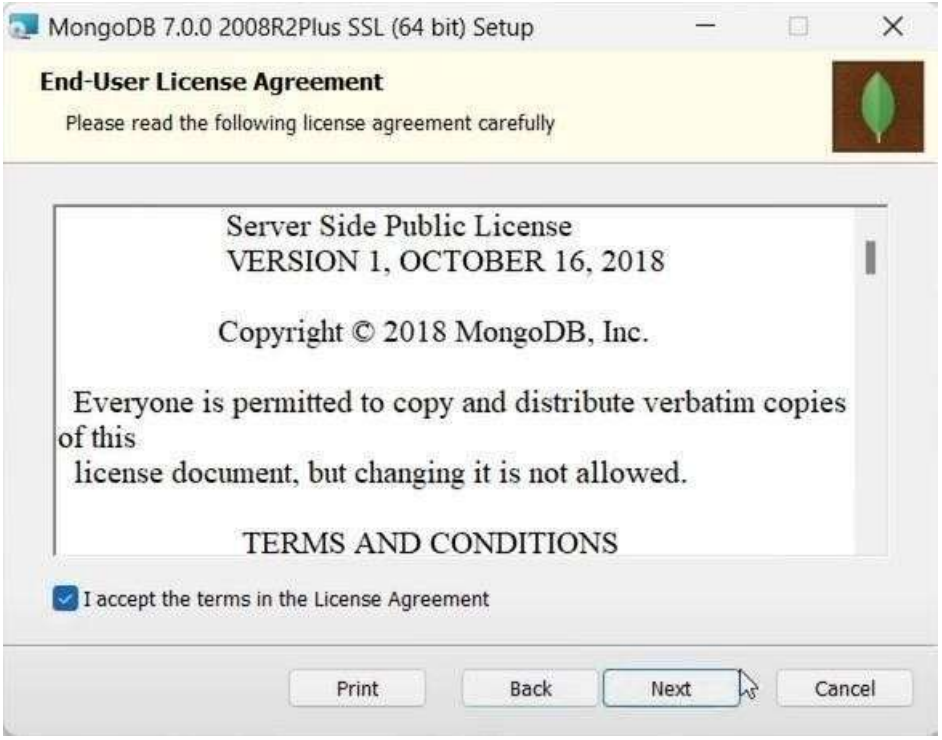
Step-1: Search "MongoDB Community Download" in browser and open first link.

Step-2: Select all the required options as below and click download.

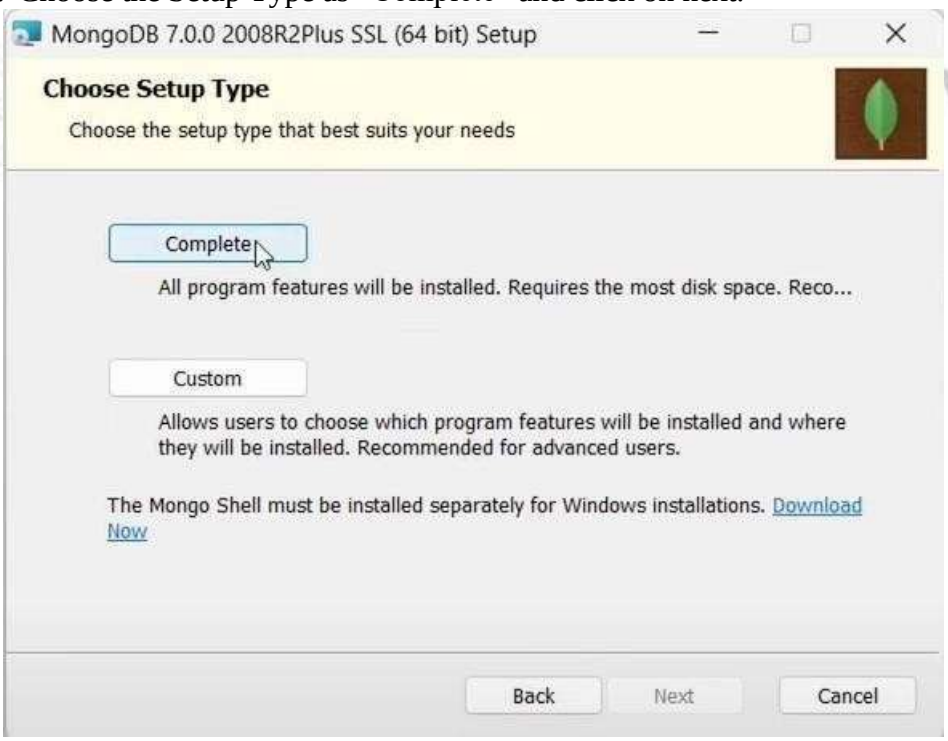


Step-3: Double click on the downloaded file and click next.

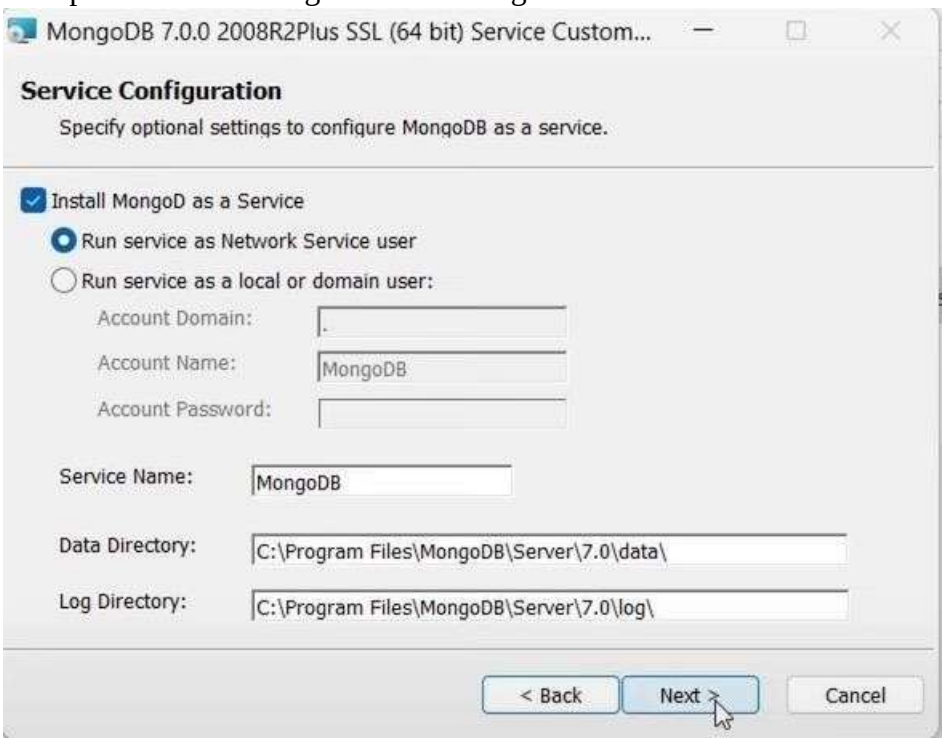
Step-4: Accept the license agreement and click next.



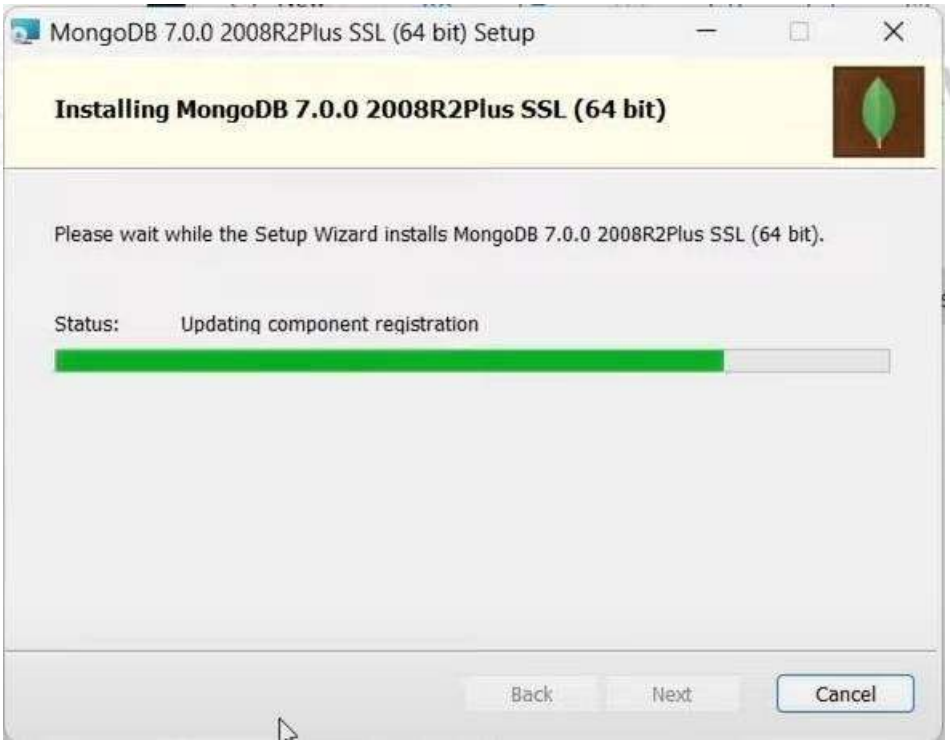
Step-5: Choose the Setup Type as "Complete" and click on next.



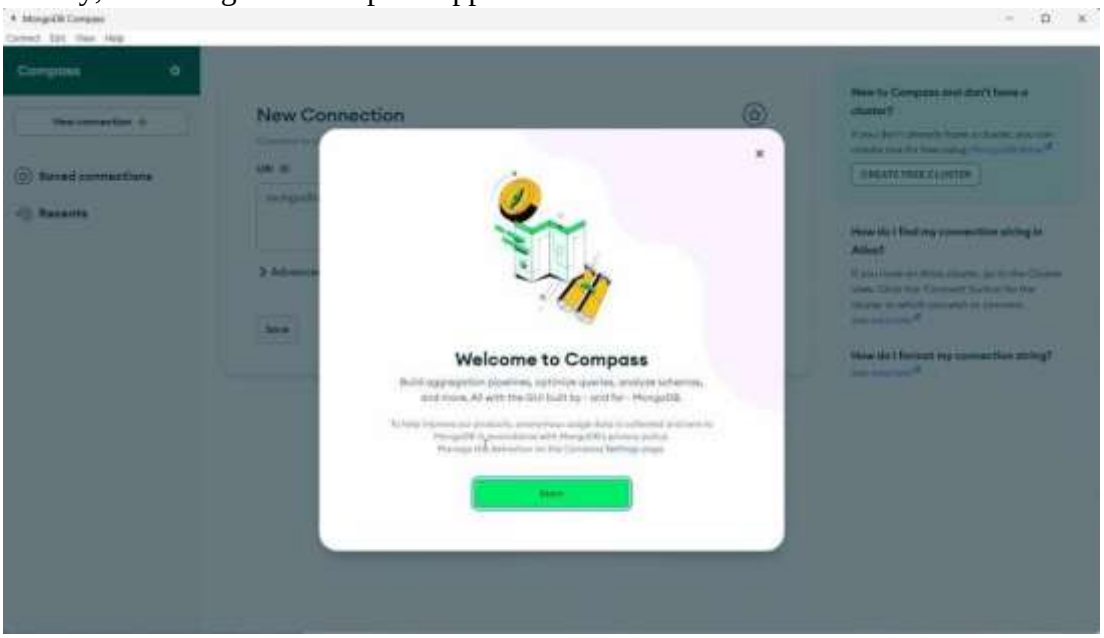
Step-6: Keep the service configuration unchanged and click next.



Step-7: Hit Install and wait for some time



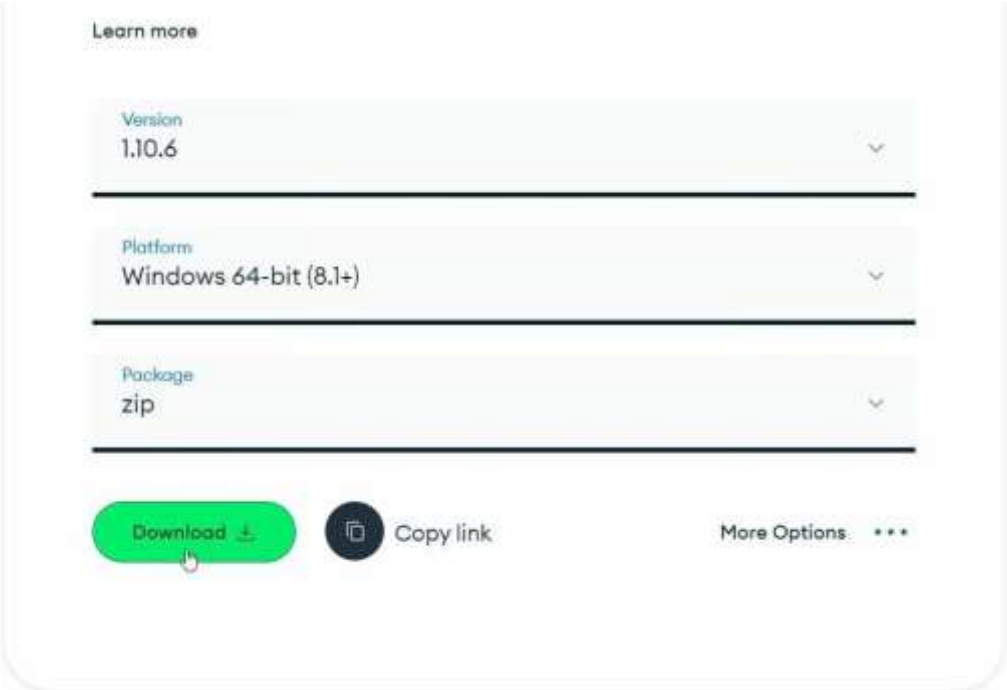
Finally, our MongoDB Compass Application has been installed.



Configuring MongoDB Shell:

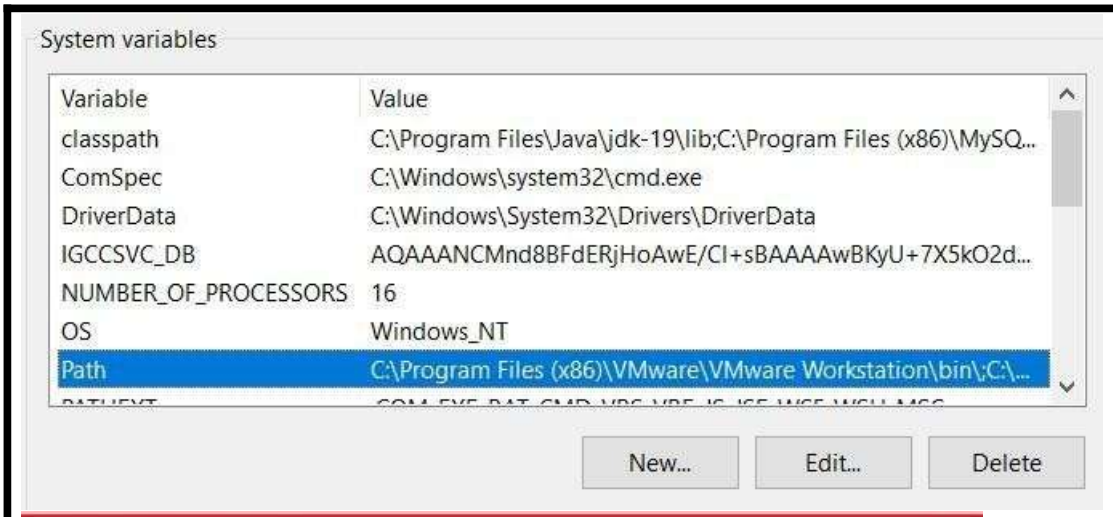
Step-1: Search "MongoDB Shell Download" in browser and open first link.

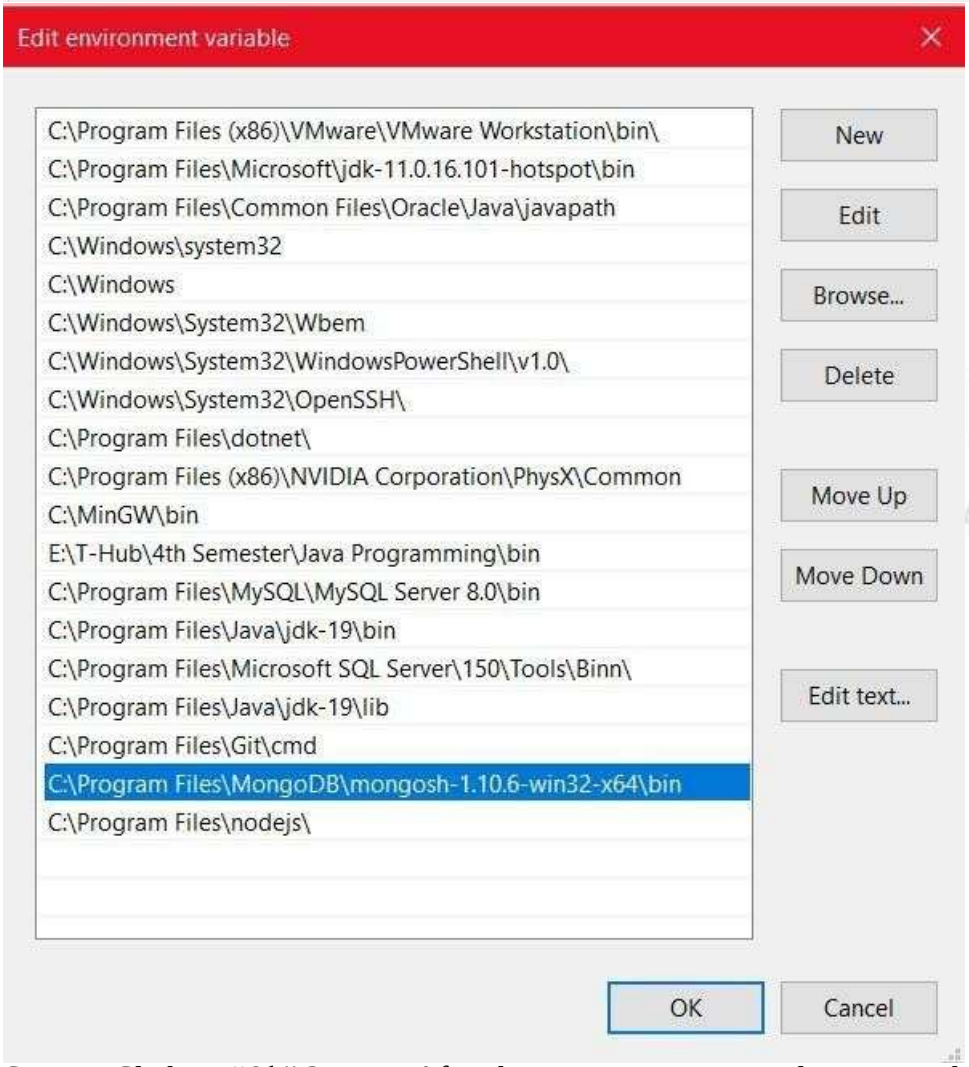
Step-2: Select all the required options as below and click download.



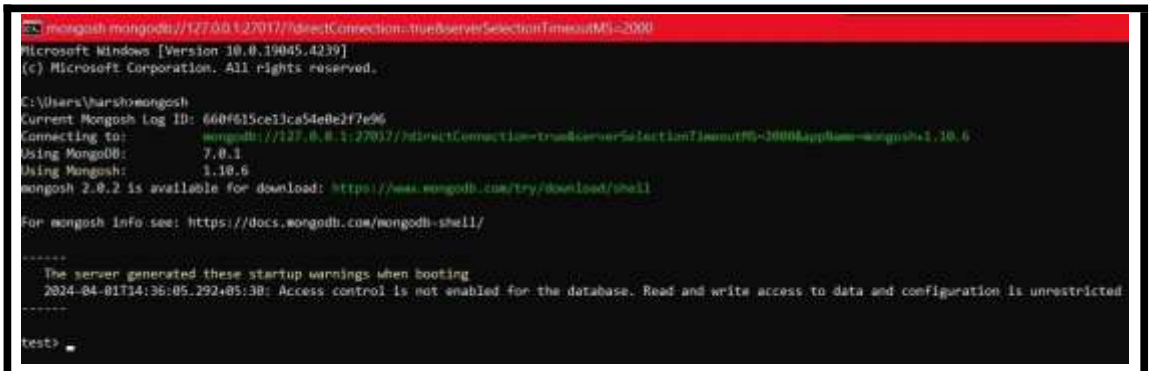
Step-3: After downloading the zip file, extract the contents inside "C:\Program Files\MongoDB". Later, go inside the extracted folder and bin folder inside. Copy the path.

Step-4: Now go to "Environment Variables" and add the copied path into "System Variables Path" as shown below.

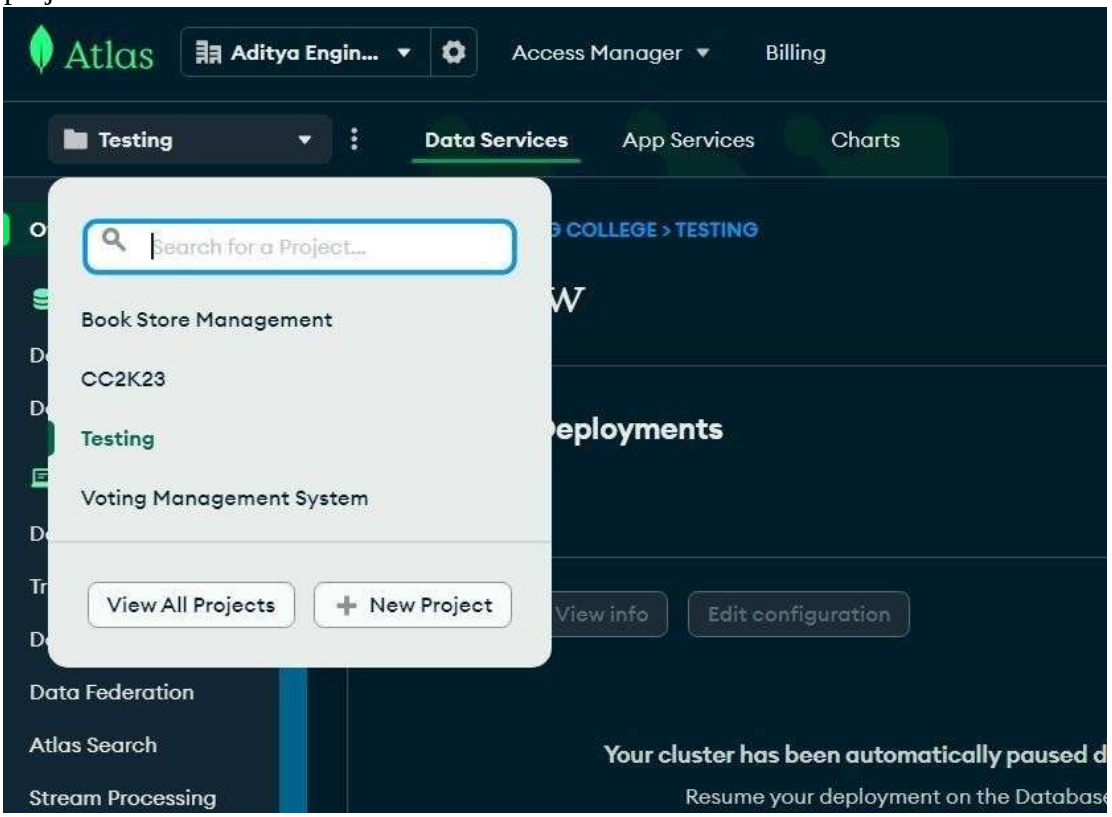




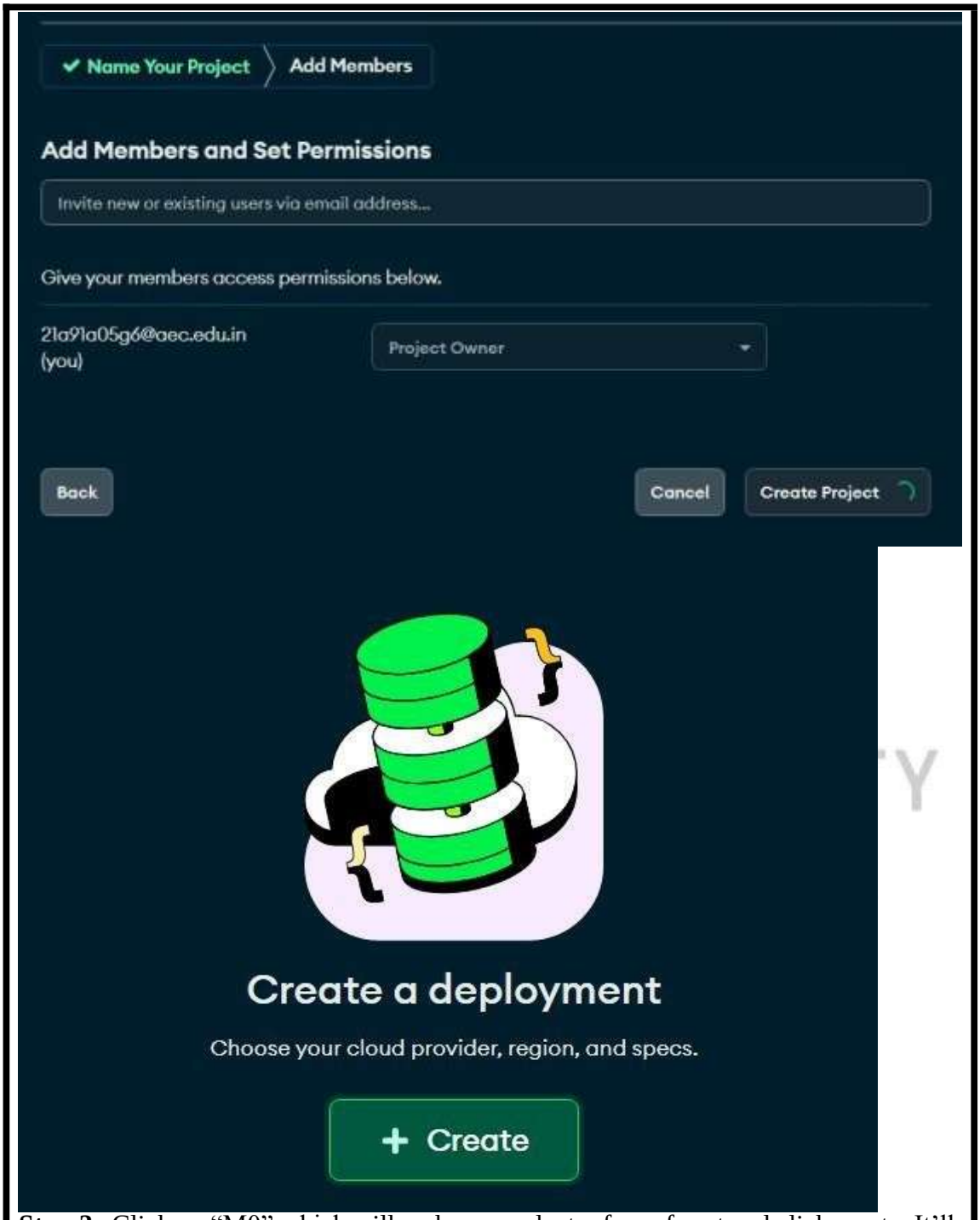
Step-5: Click on "Ok" 3 times. After this step, open command prompt and type in the command "mongosh" to access MongoDB Shell from anywhere within your system.



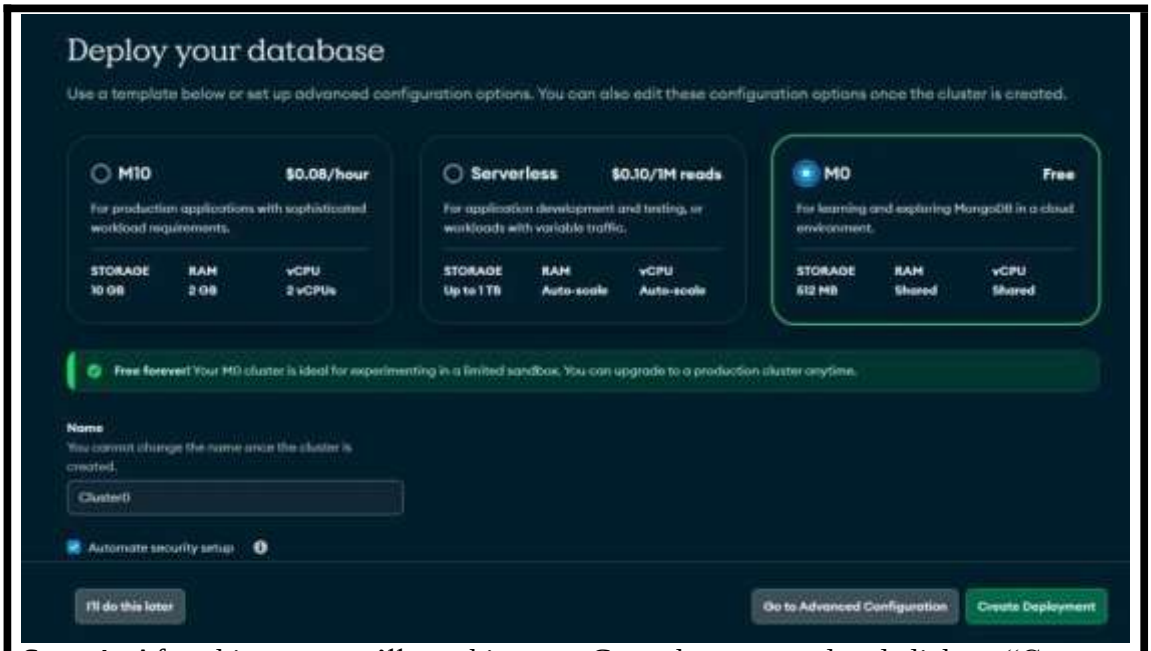
Creation of MongoDB ATLAS Cluster: It is a cloud-hosted database-as-a-service.
Step-1: Create an ATLAS Account on "cloud.mongodb.com". Then create a new project.



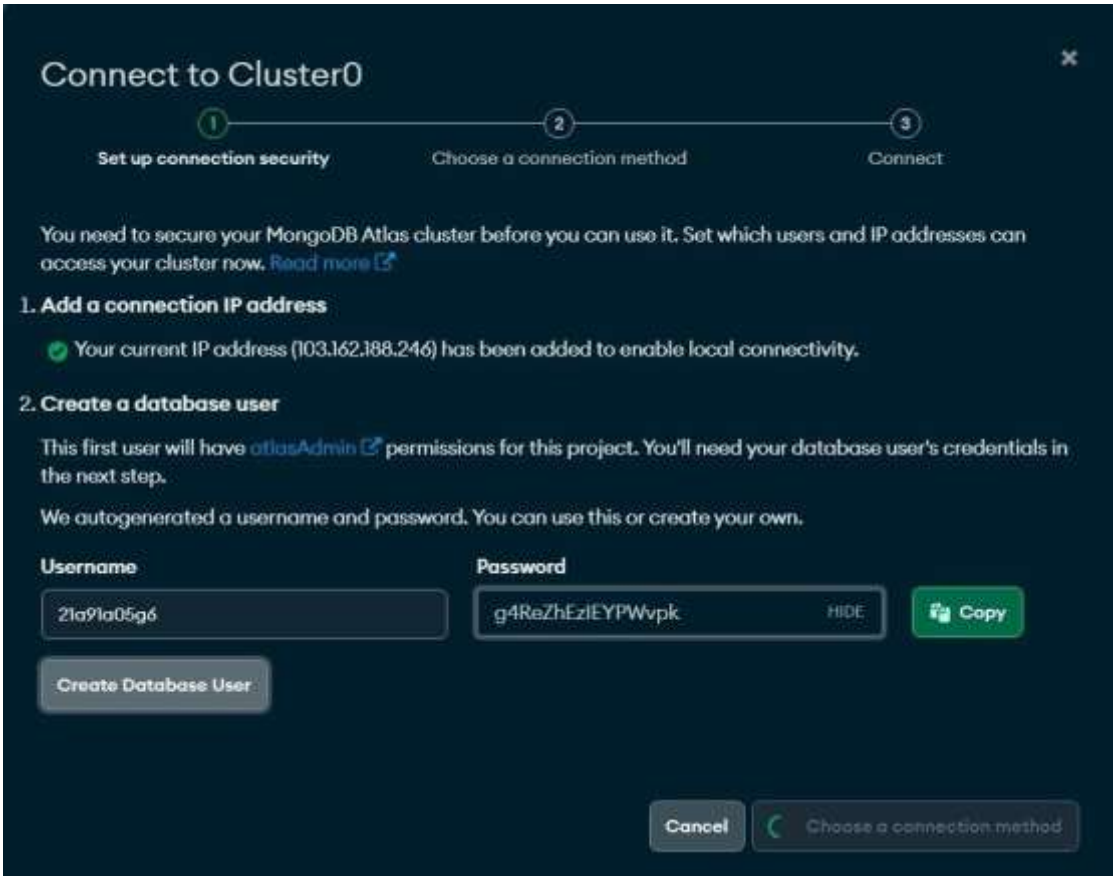
Step-2: Name your project as "MST" and click on "Create Project". After creation of project, create a new Deployment.



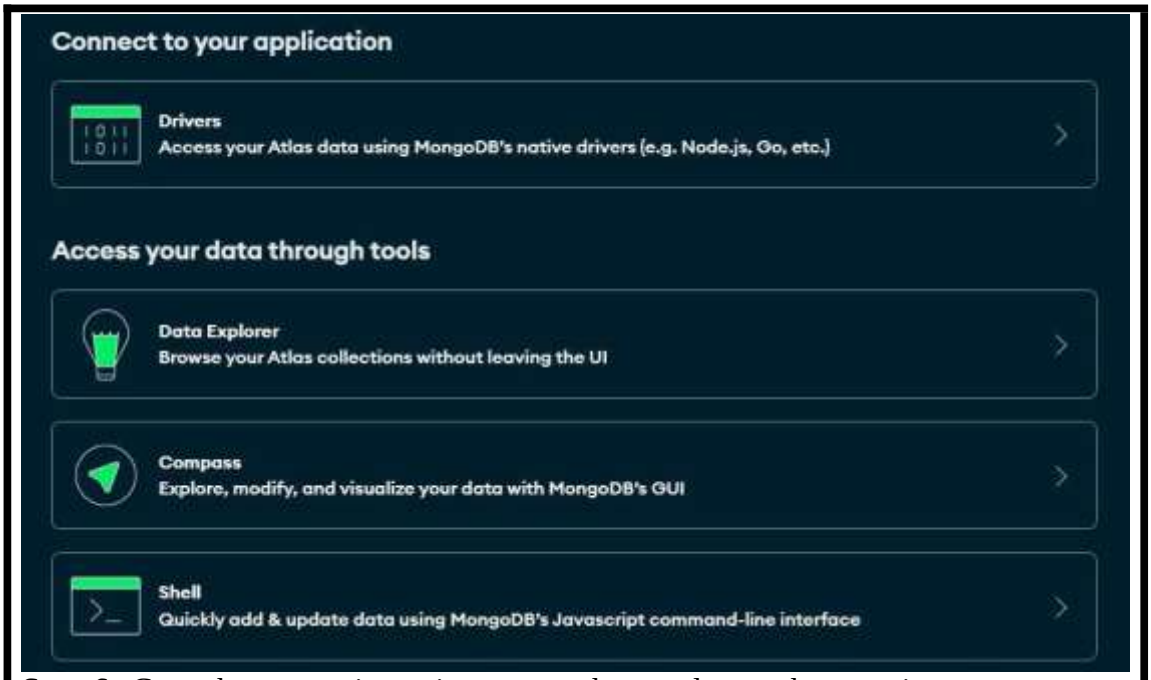
Step-3: Click on "M0" which will make your cluster free of cost and click create. It'll take 2 mins of time to create.



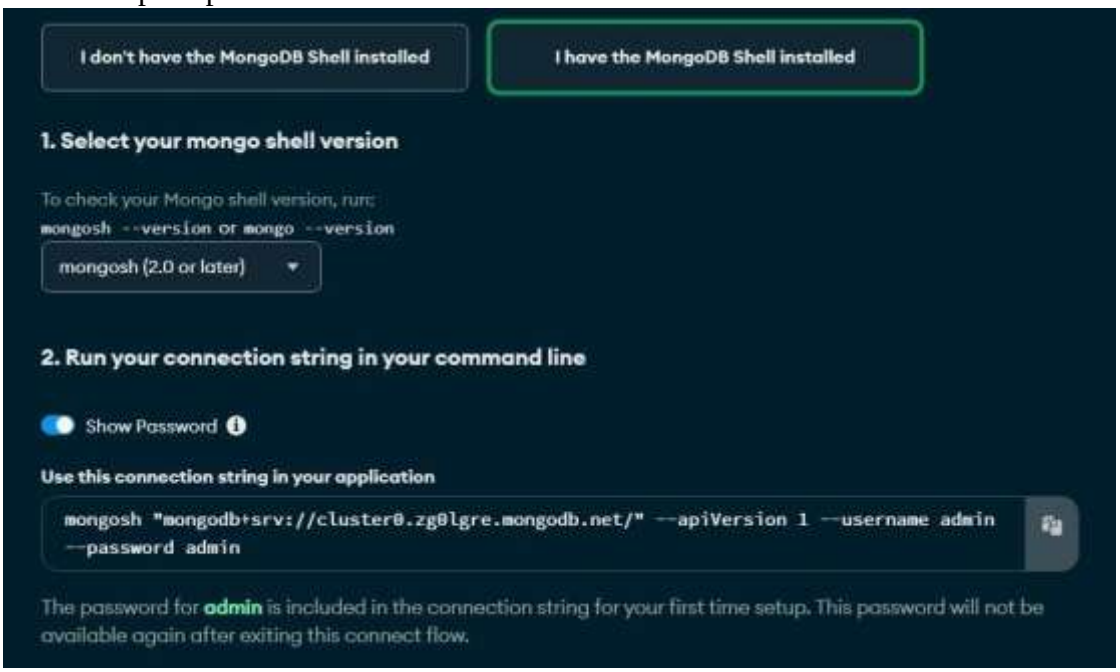
Step-4: After this step, you'll see this page. Copy the password and click on "Create Database User" and click "Choose a Connection Method".



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*".

```
Atlas atlas-9jld67-shard-0 [primary] test> show dbs
sample_mflix  110.77 MiB
admin         280.00 KiB
local          80.49 GiB
Atlas atlas-9jld67-shard-0 [primary] test>
```

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***".

```
Atlas atlas-9jld67-shard-0 [primary] test> use newdb
switched to db newdb
Atlas atlas-9jld67-shard-0 [primary] newdb>
```

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

```
Atlas atlas-9jld67-shard-0 [primary] newdb> show collections

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

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.

11b)Aim:

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

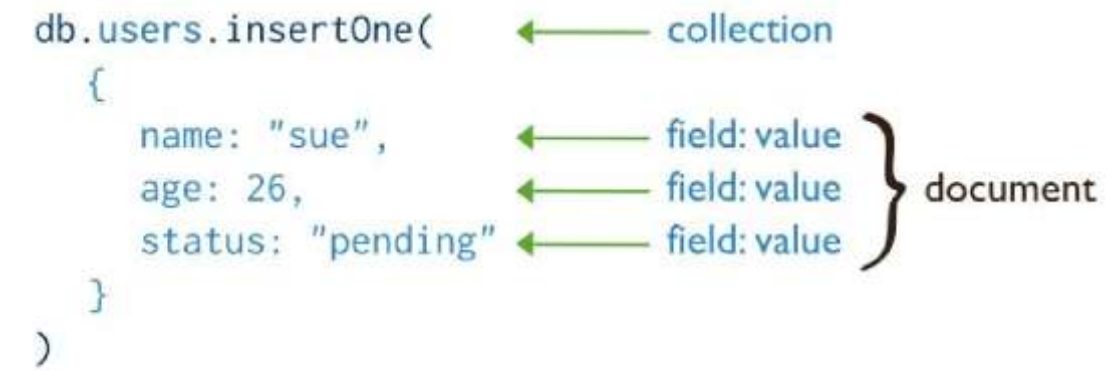
Description:

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




```
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

```
db.users.deleteMany(          ← collection
   { status: "reject" }       ← delete filter
)
```

```
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

12a)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.

```
mongosh mongodb+srv://<credentials>@cluster0.zg0lgre.mongodb.net/
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>
```

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**".

```
Atlas atlas-9jld67-shard-0 [primary] test> use newdb
switched to db newdb
Atlas atlas-9jld67-shard-0 [primary] newdb> 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] newdb>
```

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

```
Atlas atlas-9jld67-shard-0 [primary] newdb> db.user.insert({name: "Harsha", age: 19})
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("66109b87911700cc041d43f3") }
}
Atlas atlas-9jld67-shard-0 [primary] newdb> show dbs
newdb          40.00 KiB
sample_mflix  113.42 MiB
test           64.00 KiB
admin         288.00 KiB
local          80.50 GiB
Atlas atlas-9jld67-shard-0 [primary] newdb>
```

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>
```

12b)Aim:

To write MongoDB queries to work with records using find(), limit(), sort(), createIndex(), aggregate().

Description:

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:

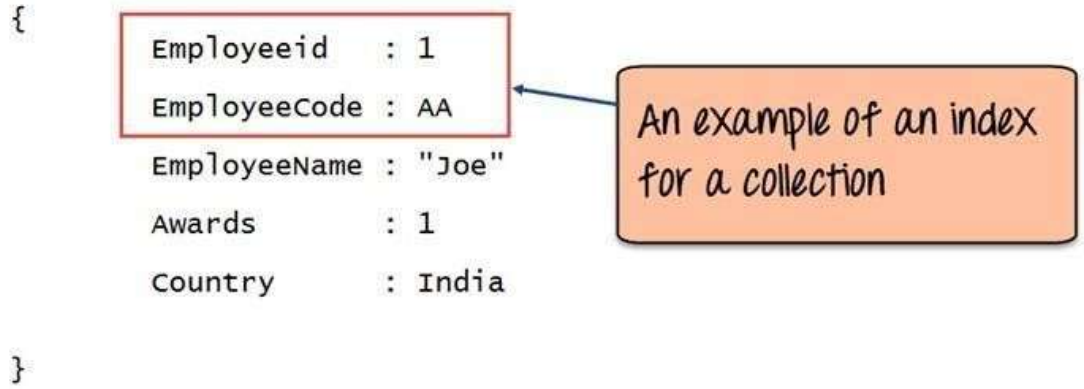
```
Atlas atlas-9jld67-shard-0 [primary] test> db.soc.find().sort({name:1})
[
  { _id: ObjectId("66109ebff9d7c1105912b052"), name: 'Angular' },
  { _id: ObjectId("66109ebff9d7c1105912b055"), name: 'CSS' },
  { _id: ObjectId("66109ebff9d7c1105912b054"), name: 'HTML' },
  { _id: ObjectId("66109ebff9d7c1105912b053"),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

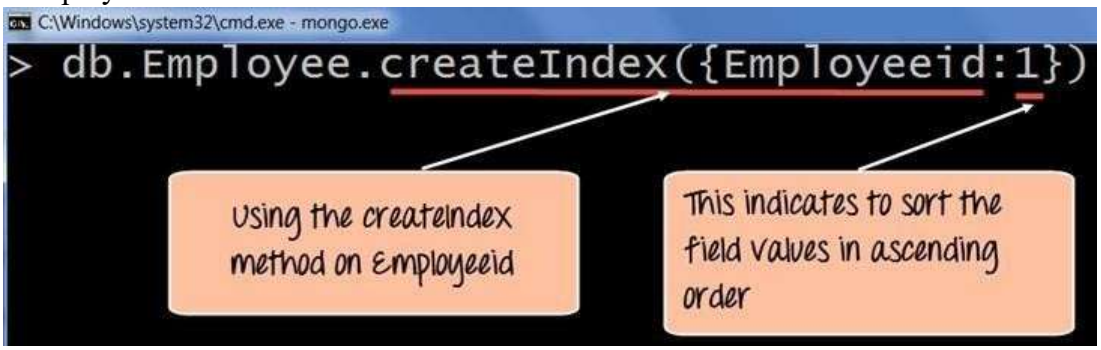
createIndex(): If you had a collection with thousands of documents with no indexes, and then you query to find certain documents, then in such case MongoDB would need to scan the entire collection to find the documents. But if you had indexes, MongoDB would use these indexes to limit the number of documents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

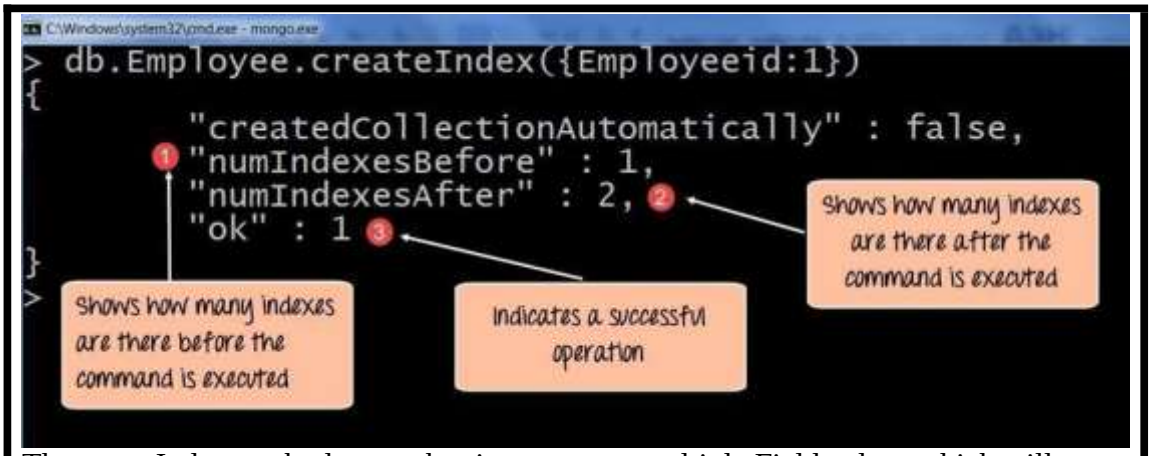
In the example below, the Employeeid "1" and EmployeeCode "AA" are used to index the documents in the collection. So when a query search is made, these indexes will be used to quickly and efficiently find the required documents in the collection. So even if the search query is based on the EmployeeCode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



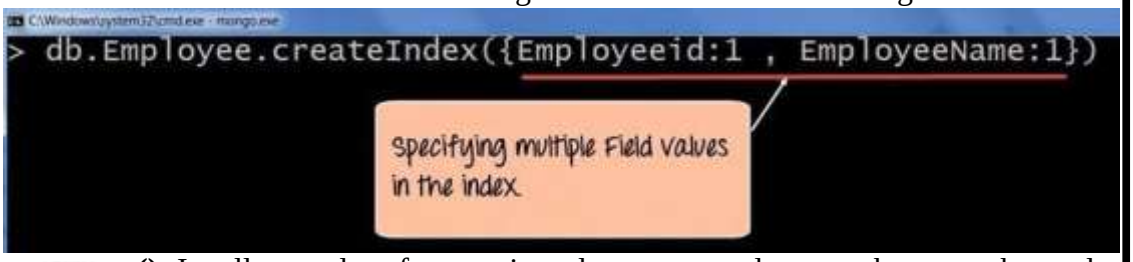
Creating an Index in MongoDB is done by using the "createIndex" method.

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".





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.