

Experiment-2: Course Name: Angular JS

2-a) Module Name: Structural Directives - ngIf

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

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

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

Program:

app.component.ts:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  isAuthenticated!: boolean;
  submitted=false;
  userName!: string;
  onSubmit(name: string,password: string){
    this.submitted=true;
    this.userName=name;
    if(name==="admin" && password==="admin"){
      this.isAuthenticated=true;
    }
    else{
      this.isAuthenticated=false;
    }
  }
}
```

app.component.html:

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

```

<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 { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';

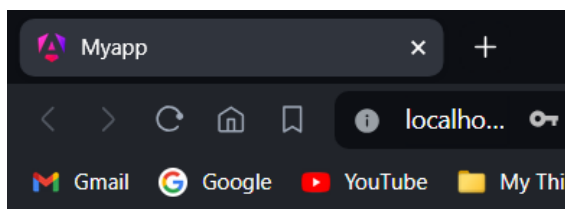
```

```

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

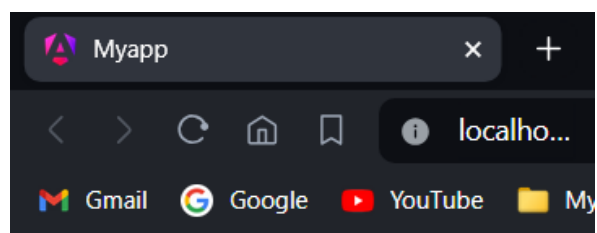
```

Output:



User Name:

Password:



Welcome admin

2-b) Module Name: ngFor

Create a courses array and rendering it in the template using ngFor directive in a list format.

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

Description: NgFor is used as a structural directive that renders each element for the given collection each element can be displayed on the page.

Syntax: <li *ngFor='condition'>

Program:

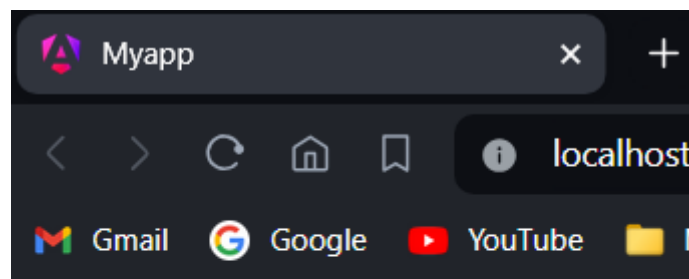
app.component.ts:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  courses: any[] = [
    {id:1,name:"TypeScript"},
    {id:2,name:"Angular"},
    {id:3,name:"Node.js"},
    {id:4,name:"MongoDB"}
  ];
}
```

app.component.html:

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

Output:



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

2-c) Module Name: ngSwitch

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

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

Description: The NgSwitch directive specifies an expression to match against. The NgSwitchCase directive defines the expressions to match.

- It renders every view that matches.
- If there are no matches, the view with the NgSwitchDefault directive is rendered.
- Elements outside of any NgSwitchCase or NgSwitchDefault directive but within the NgSwitch statement but are preserved at the location.

Program:

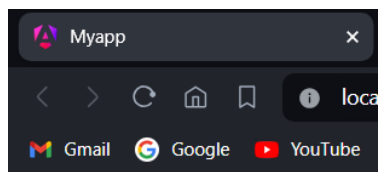
app.component.ts:

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

app.component.html:

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

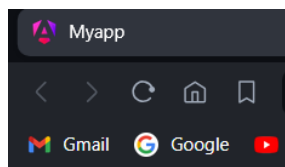
Output:



Current choice is 0

Default Choice

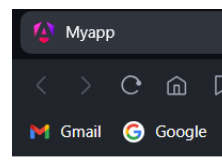
Next Choice



Current choice is 1

First Choice

Next Choice

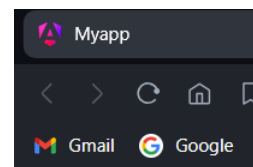


Current choice is 2

Second Choice

Second Choice Again

Next Choice



Current choice is 3

Third Choice

Next Choice

2-d) Module Name: Custom Structural Directive

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

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

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

Generate a directive called 'repeat' using the following command:

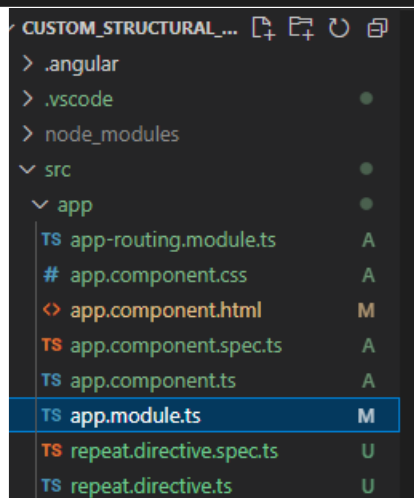
ng generate directive repeat

```
C:\Users\admin\custom_structural_directives>ng generate directive repeat
? Would you like to share pseudonymous usage data about this project with the Angular Team
at Google under Google's Privacy Policy at https://policies.google.com/privacy. For more
details and how to change this setting, see https://angular.io/analytics. Yes

Thank you for sharing pseudonymous usage data. Should you change your mind, the following
command will disable this feature entirely:

  ng analytics disable

Global setting: enabled
Local setting: enabled
Effective status: enabled
CREATE src/app/repeat.directive.spec.ts (224 bytes)
CREATE src/app/repeat.directive.ts (141 bytes)
UPDATE src/app/app.module.ts (468 bytes)
```



Program:

app.module.ts:

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

```

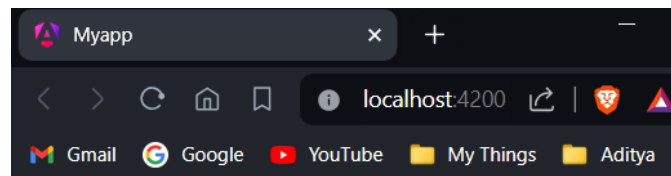
app.component.html:

```

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

```

Output:



Custom Structural Directives

Angular JS

Angular JS

Angular JS

Angular JS

Angular JS

Angular JS

Experiment-4: Course Name: Angular JS

4-a) Module Name: Property Binding

Binding image with class property using property binding.

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

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

Program:

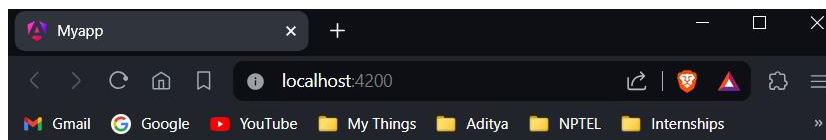
app.component.ts:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title="Property Binding";
  imgUrl: string="https://i.ytimg.com/vi/NNS5Piu-EII/hq720.jpg?sqp=-
oaymwEhCK4FEIIDSFryq4qpAxMIARUAAAAAGAEIAADIQj0AgKJD&rs=AOOn4CLCFq
zGKbnbnk8Z3Aa9t7jyUZ7jT_w";
}
```

app.component.html:

```
<h2>Property Binding</h2>
<img [src]="imgUrl" width="650px" height="350px" alt="wild">
```

Output:



Property Binding



4-b) Module Name: Attribute Binding

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

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

Description: In AngularJS, you can dynamically control the colspan attribute of a table cell using AngularJS directives and expressions. AngularJS allows you to bind data to HTML attributes, including colspan, enabling you to manipulate the table structure based on dynamic data.

Program:

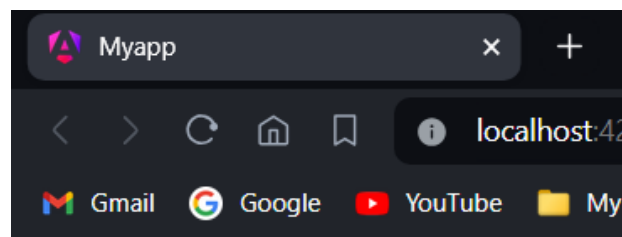
app.component.ts:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title="Property Binding";
  colspanValue="2";
}
```

app.component.html:

```
<h2>Attribute Binding</h2>
<table border="1">
  <tr>
    <td [attr.colspan]="colspanValue">First</td>
    <td>Second</td>
  </tr>
  <tr><td>Third</td><td>Fourth</td><td>Fifth</td></tr>
  <tr><td>Sixth</td><td>Seventh</td><td>Eighth</td></tr>
</table>
```

Output:



Attribute Binding

First		Second
Third	Fourth	Fifth
Sixth	Seventh	Eighth

4-c) Module Name: Style and Event Binding

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

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

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

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

Program:

Style Binding:

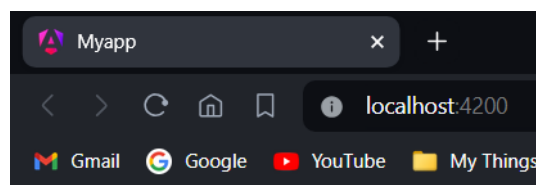
app.component.ts:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  isValid="blue";
  isValid1="13";
}
```

app.component.html:

```
<h2>Style Binding</h2>
<button [style.color]="isValid?'blue':'red'">Hello!</button><br /><br />
<button [style.font-size.px]="isValid1?11:26">Hehe</button>
```

Output:

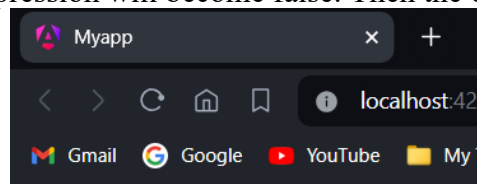


Style Binding

Hello!

Hehe

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



Style Binding

Hello!

Hehe

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:

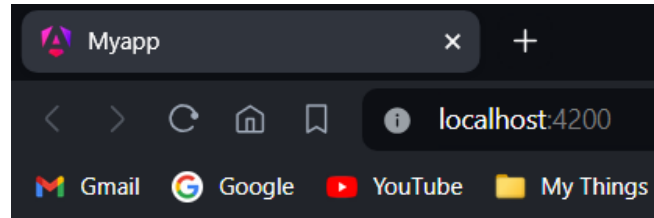
```
<h2>Event Binding</h2>
<table>
  <tr>
    <td>
      <label for="uname">Enter a User Name: </label>
      <input type="text" size="20px" id="uname" required autocomplete="off"
placeholder="Enter your username/id"><br /><br />
    </td>
  </tr>
  <tr>
    <td>
      <label for="pwd">Enter your Password: </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:

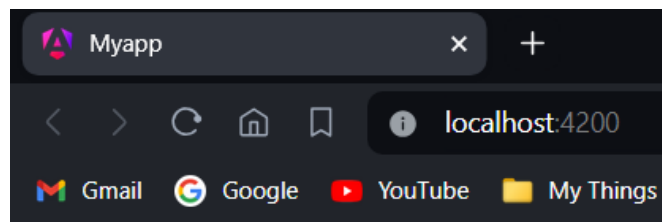


Event Binding

Enter a User Name:

Enter your Password:

Login



Event Binding

Enter a User Name:

Enter your Password:

Login

Your Name is: Harsha

Your Password is: hehehehe

Experiment-6: Course Name: Angular JS

6-a) Module Name: Passing data from Container Component to Child Component

Create an AppComponent that displays a dropdown with a list of courses as values in it. Create another component called the CoursesList component and load it in AppComponent which should display the course details. When the user selects a course from the dropdown, corresponding course details should be loaded.

Aim: Create an AppComponent that displays a dropdown with a list of courses as values in it. Create another component called the CoursesList component and load it in AppComponent which should display the course details. When the user selects a course from the dropdown, corresponding course details should be loaded.

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

Program:

app.module.ts:

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

app.component.ts:

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

course-list.component.ts:

```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'app-course-list',
  templateUrl: './course-list.component.html',
  styleUrls: ['./course-list.component.css'],
})
export class CourseListComponent {
  courses = [
    { courseId: 1, courseName: "Node JS" },
    { courseId: 2, courseName: "Typescript" },
    { courseId: 3, courseName: "Angular" },
    { courseId: 4, courseName: "React JS" },
  ];
  course!: any[];
  @Input() set cName(name: string) {
    this.course = [];
    for (var i = 0; i < this.courses.length; i++) {
      if (this.courses[i].courseName === name) {
        this.course.push(this.courses[i]);
      }
    }
  }
}
```

course-list.component.html:

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

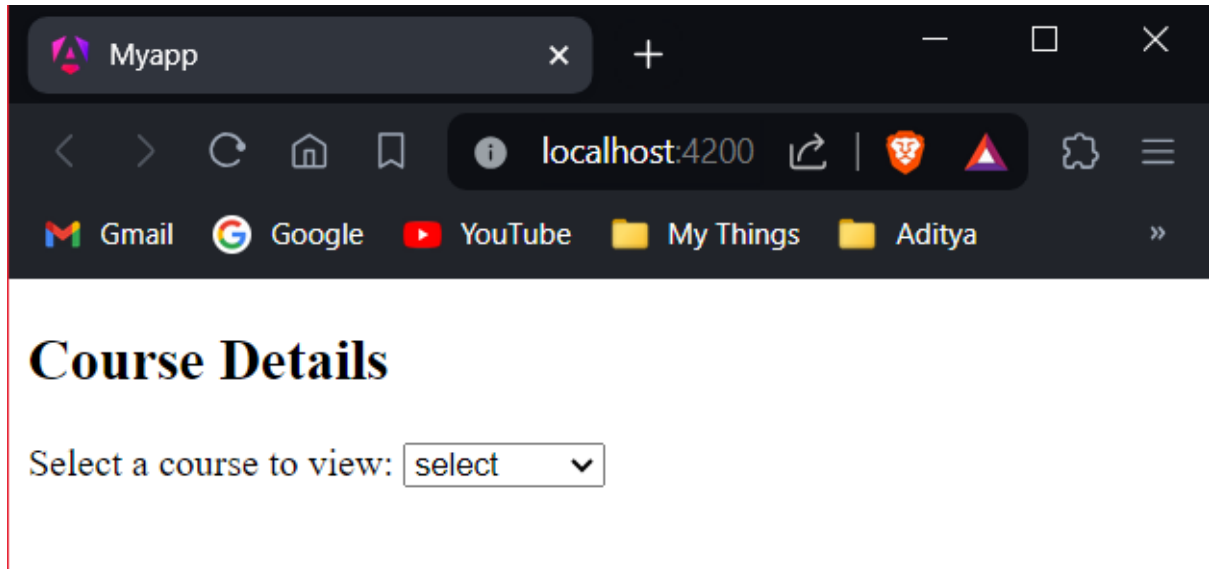
app.component.html:

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

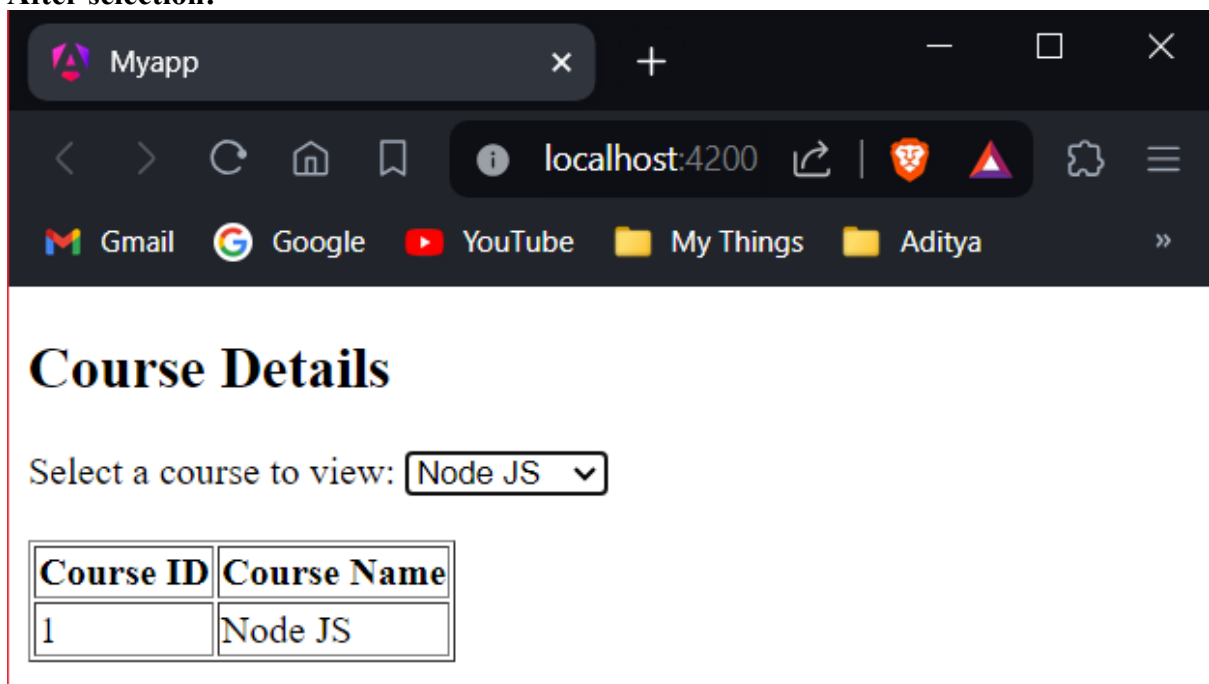
```
</select>
<br /><br />
<app-course-list [cName]="name"></app-course-list>
```

Output:

Before selection:



After selection:



6-b) Module Name: Passing data from Child Component to ContainerComponent

Create an AppComponent that loads another component called the CoursesListComponent. Create another component called CoursesListComponent which should display the courses list in a table along with a register button in each row. When a user clicks on the register button, it should send that courseName value back to AppComponent where it should display the registration successful message along with courseName.

Aim: Create an AppComponent that loads another component called the CoursesListComponent. Create another component called CoursesListComponent which should display the courses list in a table along with a register button in each row. When a user clicks on the register button, it should send that courseName value back to AppComponent where it should display the registration successful message along with courseName.

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

Program:

app.module.ts:

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

app.component.ts:

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

```

    })
    export class AppComponent {
        message!: string;
        courseReg(courseName: string) {
            this.message = `Your registration for ${courseName} is successful`;
        }
    }
}

```

course-list.component.ts:

```

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

```

app.component.html:

```

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

```

course-list.component.html:

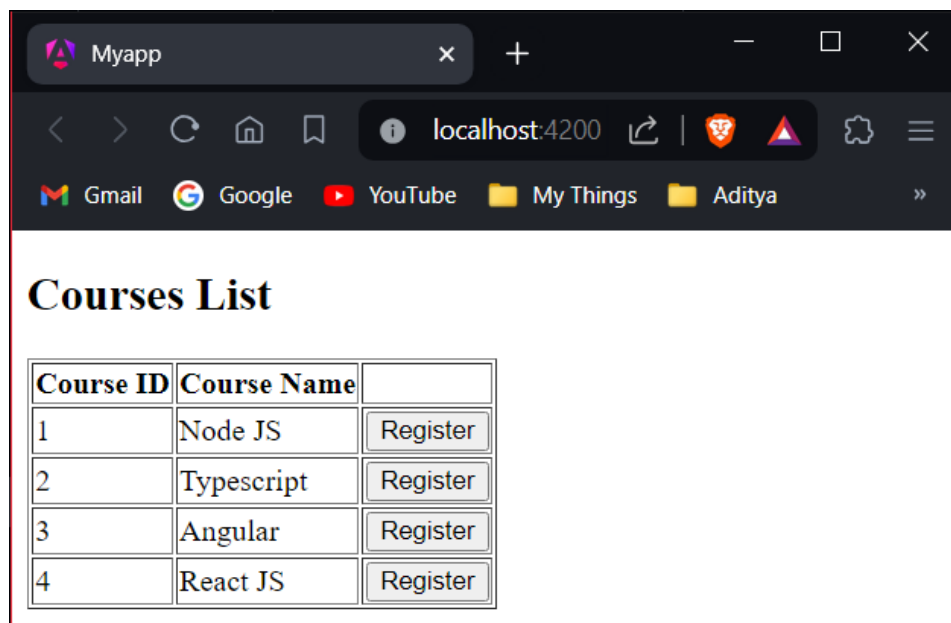
```

<table border="1">
    <thead>
        <tr>
            <th>Course ID</th>
            <th>Course Name</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        <tr *ngFor="let course of courses">
            <td>{{ course.courseId }}</td>
            <td>{{ course.courseName }}</td>
            <td><button (click)="register(course.courseName)">Register</button></td>
        </tr>
    </tbody>
</table>

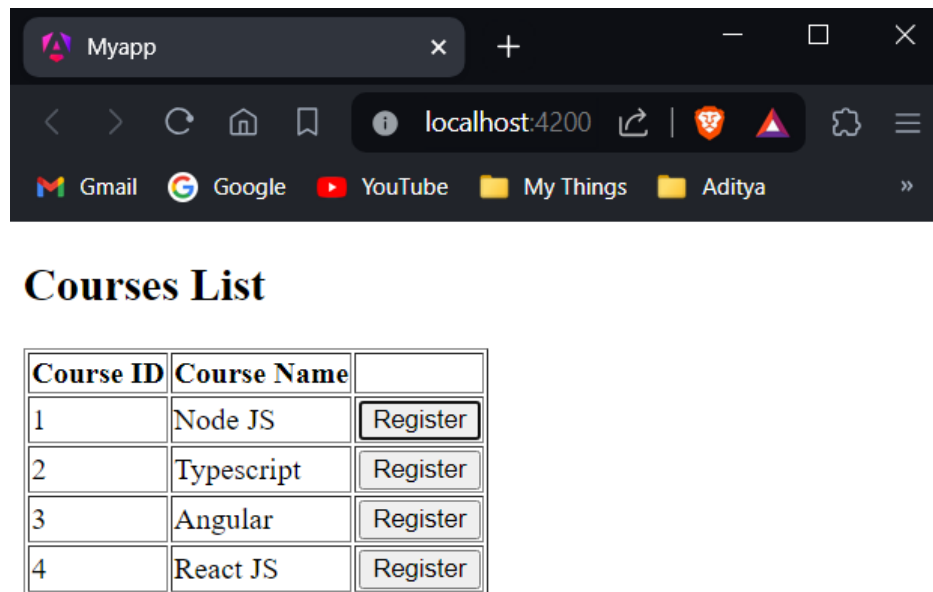
```


</table>

Output:



After Registration:



Your registration for Node JS is successful

6-c) Module Name: Shadow DOM

Apply ShadowDOM and None encapsulation modes to components.

Aim: Apply ShadowDOM and None encapsulation modes to components.

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

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

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

```
ng generate component first
```

Generate another component called “second” using the following command:

```
ng generate component second
```

app.module.ts:

```
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { AppRoutingModuleModule } 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,
    AppRoutingModuleModule
  ],
  providers: [],
  bootstrap: [
    AppComponent,
  ]
})
export class AppModule { }
```

ViewEncapsulation.ShadowDOM:

first.component.css:

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

first.component.html:

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

second.component.css:

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

second.component.html:

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

second.component.ts:

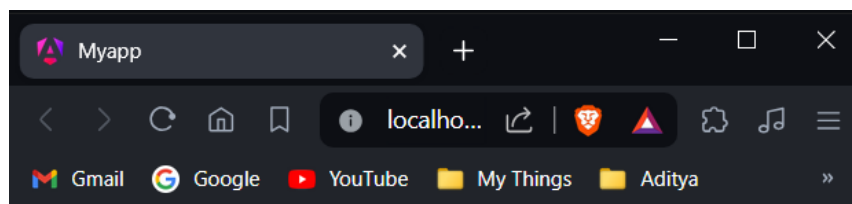
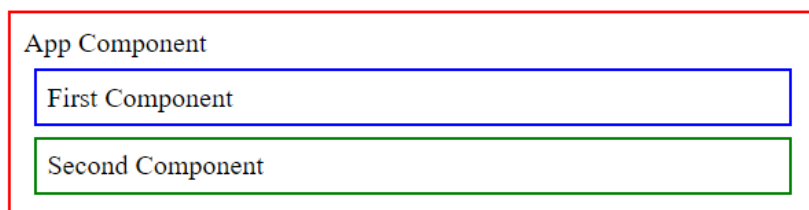
```
import { Component, ViewEncapsulation } from '@angular/core';  
@Component({  
  selector: 'app-second',  
  templateUrl: './second.component.html',  
  styleUrls: ['./second.component.css'],  
  // Experiment-6-c  
  encapsulation: ViewEncapsulation.ShadowDom  
})  
export class SecondComponent {}
```

app.component.css:

```
.cmp {  
  padding: 8px;  
  margin: 6px;  
  border: 2px solid red;  
}
```

app.component.html:

```
<h3>CSS Encapsulation with Angular</h3>  
<div class="cmp">  
  App Component  
  <app-first></app-first>  
  <app-second></app-second>  
</div>
```

Output:**CSS Encapsulation with Angular**

ViewEncapsulation.None:

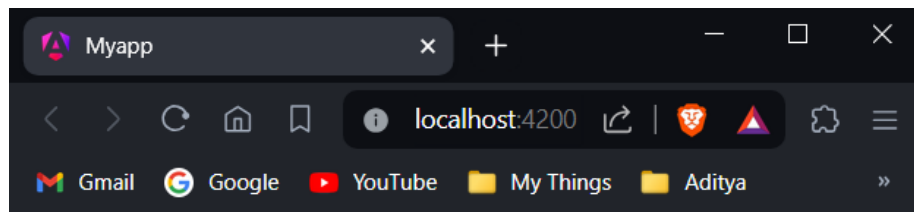
app.component.ts:

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

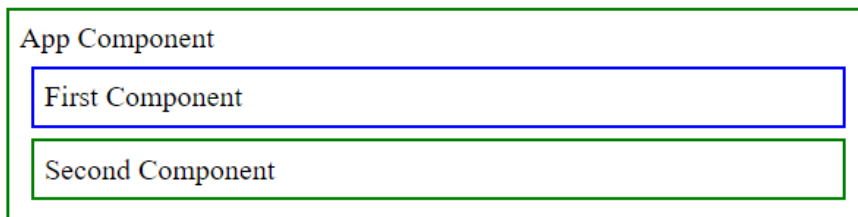
second.component.ts:

```
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:



CSS Encapsulation with Angular



6-d) Module Name: Component Life Cycle

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

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

Description: A component has a life cycle that is managed by Angular. It includes creating a component, rendering it, creating and rendering its child components, checks when its data-bound properties change, and destroy it before removing it from the DOM. Angular has some methods/hooks which provide visibility into these key life moments of a component and the ability to act when they occur.

Lifecycle Hooks:

ngOnChanges – It gets invoked when Angular sets data-bound input property i.e., the property attached with @Input(). This will be invoked whenever input property changes its value.

ngOnInit – It gets invoked when Angular initializes the directive or component

ngDoCheck – It will be invoked for every change detection in the application

ngAfterContentInit – It gets invoked after Angular projects content into its view

ngAfterContentChecked – It gets invoked after Angular checks the bindings of the content it projected into its view.

ngAfterViewInit – It gets invoked after Angular creates component's views

ngAfterViewChecked – Invokes after Angular checks the bindings of the component's views

ngOnDestroy – It gets invoked before Angular destroys directive or component

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

```
ng generate component child
```

Program:

app.module.ts:

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

```
export class AppModule { }
```

app.component.ts:

```
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));
  }
}
```

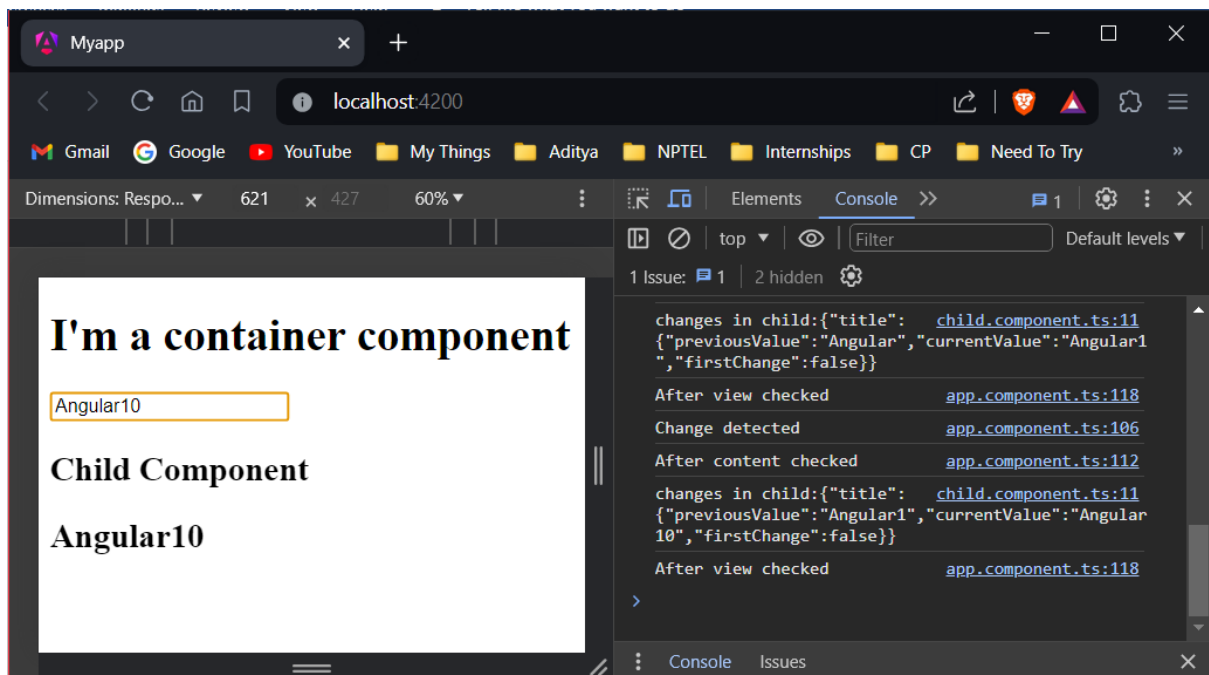
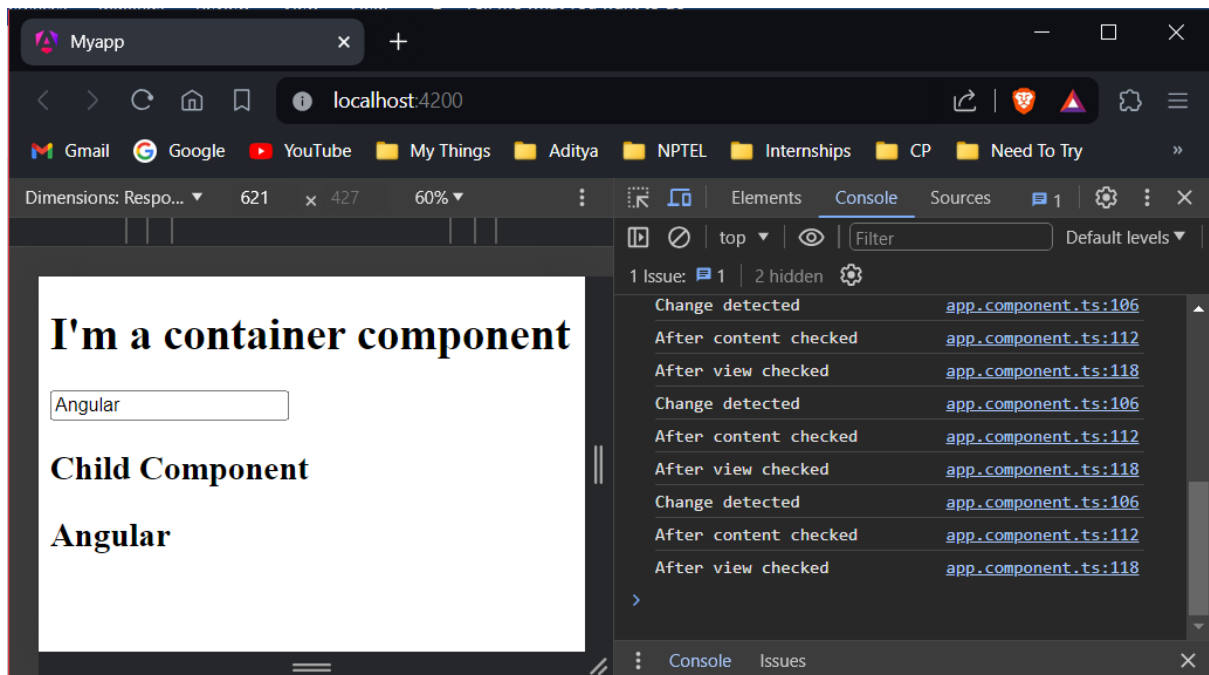
app.component.html:

```
<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-8: Course Name: Angular JS

8-a) Module Name: Custom Validators in Template Driven forms

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

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

Description: A template-driven form is the simplest way to build a form in Angular. It uses Angular's two-way data-binding directive (ngModel) to create and manage the underlying form instance. Additionally, as the name suggests, a template form is mainly driven by the view component. So, it uses directives placed in HTML rather than TypeScript or JavaScript to manage the form. While creating forms, there can be situations for validations for which built-in validators are not available. Few such examples include validating a phone number, validating if the password and confirm password fields matches or not, etc. In such situations, we can create custom validators to implement the required functionality.

Program:

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

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

course.ts:

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

course-form.component.ts:

```
import { Component } from '@angular/core';
import { Course } from '../course';
@Component({
  selector: 'app-course-form',
  templateUrl: './course-form.component.html',
  styleUrls: ['./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>

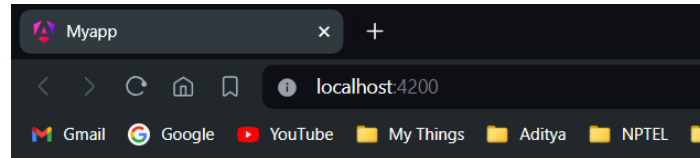
```

```

    </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 { AppRoutingModuleModule } 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,
    AppRoutingModuleModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [
    AppComponent,
  ]
})
export class AppModule { }
app.component.html:
<app-course-form></app-course-form>

```

Output:



Course Form

Course Id

1

Course Name

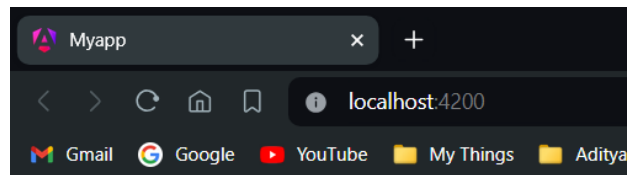
Angular

Email

sample@gmail.com

Submit

New Course



Course Form

Course Id

1

Course Name

Angular

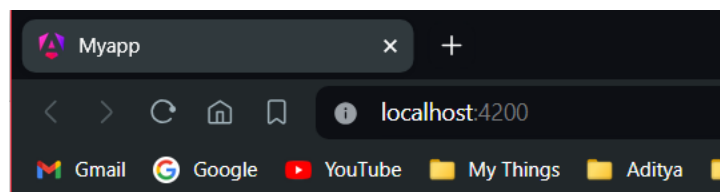
Email

sample123@

Email is invalid

Submit

New Course



You submitted the following:

Course ID

1

Course Name

Angular

Email

sample123@gmail.com

Edit

8-b) Module Name: Services Basics

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

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.

Description: A service in Angular is a class that contains some functionality that can be reused across the application. A service is a singleton object. Angular services are a mechanism of abstracting shared code and functionality throughout the application. Angular Services come as objects which are wired together using dependency injection. Angular provides a few inbuilt services also can create custom services. Services can be used to share the code across components of an application and to make HTTP requests.

To begin, generate a component called book with the following command:

```
ng generate component book
```

Program:

Create “book.ts” inside book component folder:

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

Create “books-data.ts” inside book component folder:

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

Move inside book folder: cd .\src\app\book\
Generate a service called “book”: ng generate service book

Come back to myapp folder: cd ../../..

book.service.ts:

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

book.component.ts:

```
import { Component, OnInit } from '@angular/core';  
import { Book } from './book';  
import { BookService } from './book.service';  
@Component({  
  selector: 'app-book',  
  templateUrl: './book.component.html',  
})
```

```

    styleUrls: ['./book.component.css']
  })
  export class BookComponent implements OnInit {
    books!: Book[];
    constructor(private bookService: BookService) { }
    getBooks() {
      this.books = this.bookService.getBooks();
    }
    ngOnInit() {
      this.getBooks();
    }
  }
}

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

book.component.css:
.books {
  margin: 0 0 2em 0;
  list-style-type: none;
  padding: 0;
  width: 13em;
}
.books li {
  cursor: pointer;
  position: relative;
  left: 0;
  background-color: #eee;
  margin: 0.5em;
  padding: 0.3em 0;
  height: 1.5em;
  border-radius: 4px;
}
.books li:hover {
  color: #607d8b;
  background-color: #ddd;
  left: 0.1em;
}
.books .badge {
  display: inline-block;
  font-size: small;
  color: white;
  padding: 0.8em 0.7em 0 0.7em;
  background-color: #607d8b;
  line-height: 0.5em;
  position: relative;
  left: -1px;
}

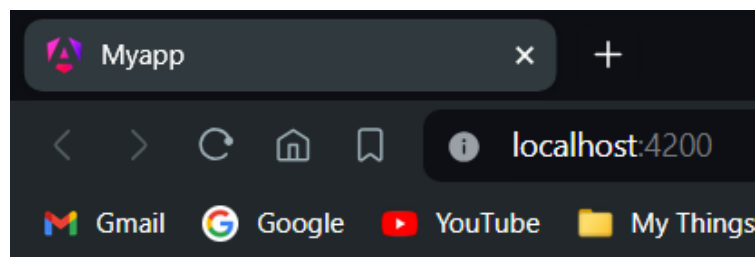
```

```

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

```

Output:



My Books

- 1 HTML 5
- 2 CSS 3
- 3 Java Script
- 4 Node.js
- 5 Angular JS

8-c) Module Name: RxJS Observables

Create and use an observable in Angular.

Aim: To create and use an observable in Angular.

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

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

Program:

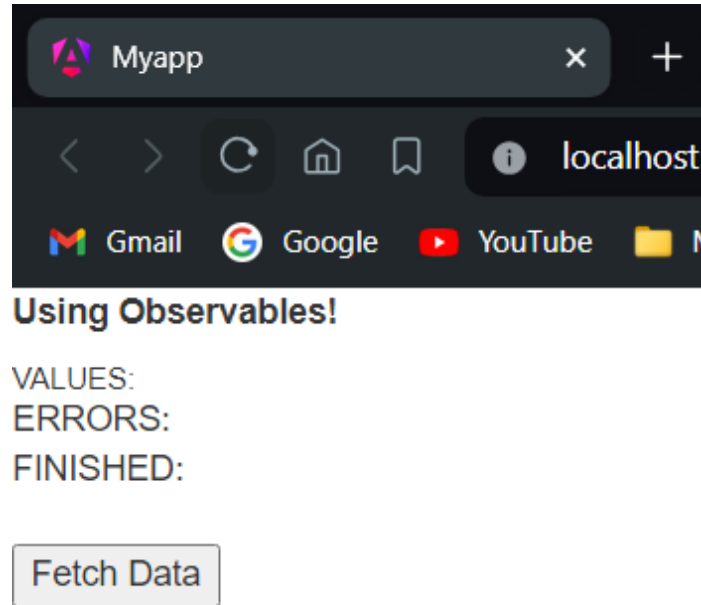
app.component.ts:

```
import { Component } from '@angular/core';
import { Observable } from 'rxjs';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
})
export class AppComponent {
  data!: Observable<number>;
  myArray: number[] = [];
  errors!: boolean;
  finished!: boolean;
  fetchData(): void {
    this.data = new Observable(observer => {
      setTimeout(() => { observer.next(11); }, 1000),
      setTimeout(() => { observer.next(22); }, 2000),
      setTimeout(() => { observer.complete(); }, 3000);
    });
    this.data.subscribe((value) => this.myArray.push(value),
      error => this.errors = true,
      () => this.finished = true);
  }
}
```

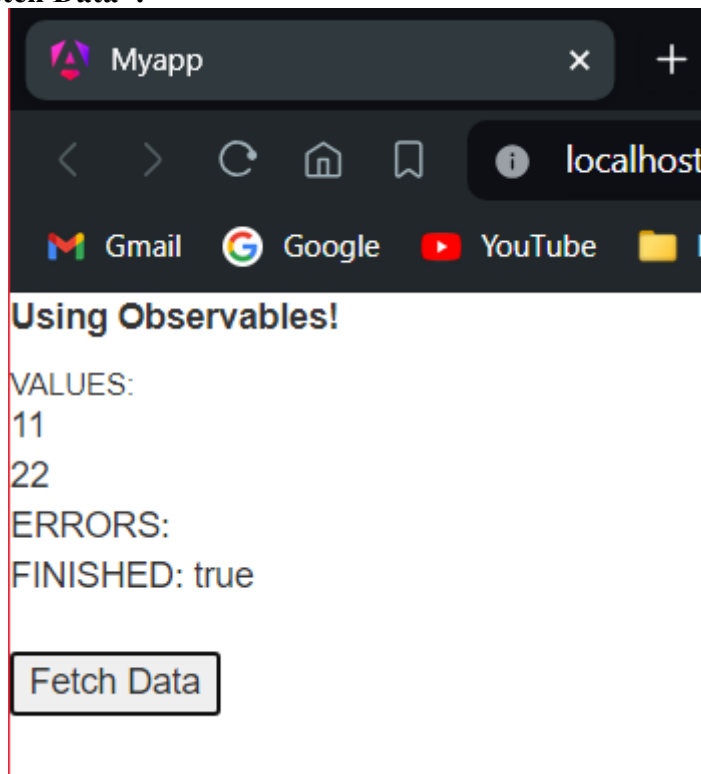
app.component.html:

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

Output:



After clicking “Fetch Data”:



Experiment-10: Course Name: Angular JS

10-a) Module Name: Routing Basics, Router Links

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

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

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

Program:

dashboard.component.ts:

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

dashboard.component.html:

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

book.service.ts:

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpResponse, 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: HttpResponse): 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:

Tour of Books1

Dashboard Books

Top Books

CSS 3

Java Script

Ajax Programming

jQuery

Tour of Books1

Dashboard

Books

My Books

1 HTML 5

2 CSS 3

3 Java Script

4 Ajax Programming

Tour of Books1

Dashboard

Books

CSS 3 details!

id: 2

name: CSS 3

Back

```

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(
  { status: "reject" }
)

```



```

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

```

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

Experiment-12: Course Name: MongoDB Essentials - A Complete MongoDB Guide

12-a) Module Name: Create and Delete Databases and Collections

Write MongoDB queries to Create and drop databases and collections.

Aim: To write MongoDB queries to Create and drop databases and collections.

Description:

Database:

A database is an organized collection of structured or unstructured information stored electronically on a machine locally or in the cloud. Databases are managed using a Database Management System (DBMS). The DBMS acts as an interface between the end user (or an application) and the database. Databases use a query language for storing or retrieving data.

Collection:

A collection is a grouping of MongoDB documents. Documents within a collection can have different fields. A collection is the equivalent of a table in a relational database system. A collection exists within a single database.

Queries:

Create Database:

Once you have access to a cluster via the shell, you can see all the databases in the cluster that you have access using “**show dbs**” command.

```
C:\> 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> _
```

12-b) Module Name: Introduction to MongoDB Queries

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

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

Description and Queries:

find(): To select data from a collection in MongoDB, we can use the find() method. This method accepts a query object. If left empty, all documents will be returned.

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

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

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

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


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

Sorting data in ascending order:

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

Sorting data in descending order:

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

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

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

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

```
{
  Employeeid : 1
  EmployeeCode : AA
  EmployeeName : "Joe"
  Awards : 1
  Country : India
}
```

An example of an index for a collection

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

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

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

using the createIndex method on Employeeid

This indicates to sort the field values in ascending order

```

C:\Windows\system32\cmd.exe - mongo.exe
> db.Employee.createIndex({Employeeid:1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}

```

Shows how many indexes are there before the command is executed

Shows how many indexes are there after the command is executed

Indicates a successful operation

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.

```

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

```

Specifying multiple Field values in the index.

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.

```

db.train.aggregate([{$group : { _id : "$id", total : { $sum : "$fare" } } })

```

Stage Expression Accumulator

```

db.train.aggregate( [
  { $match : { class : "first-class" } },
  { $group : { _id : "id", total : { $sum : "$fare" } } }
] )

```

pipeline stages

