R. C. Patel Institute of Management Research and Development, Shirpur

BCA 507(A) Lab Web Development Technology-III

INDEX

Practical No	Practical Title	Remark
1.	Create a user-friendly interface with a clean and proper design	
2.	Develop AngularJS program to create a login form, with validation for the username and password fields.	
3.	To demonstrate AngularJS services and data bind.	
4.	To display tasks in a list with checkboxes for marking completion as per user choice.	
5.	To use ng-if directive to display tasks in the UI.	
6.	To create database and structure in MongoDB.	
7.	To create collection and insert records in collections.	
8.	To demonstrate insert, update, delete, select operations in MongoDB.	
9.	Develop a task manager application using AngularJS for the frontend and MongoDB for the backend	
10.	To Create Student interface to stored and update the information.	
11.	To display students information reports (All, Parametized)	
12.	Implement a user interface where users can view, add, update, and delete tasks with MongoDB Database.	

Practical 1:- Create a user-friendly interface with a clean and proper design.

Practical 1 a) Creating Simple AngularJS Application.

Step - 1: Load framework - Being a pure JavaScript framework, it can be added using <Script> tag.

```
<script src
="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
</script>
```

Or [If you install AngularJS]

```
<script src="node_modules/angular/angular.js"> </script>
```

Step - 2: Define AngularJS application using ngapp directive

```
<div ng-app = ""> ... </div>.
```

Step - 3: Define a model name using ng-model directive

```
<p>Enter your Name: <input type = "text" ng-model =
"name"></p>
```

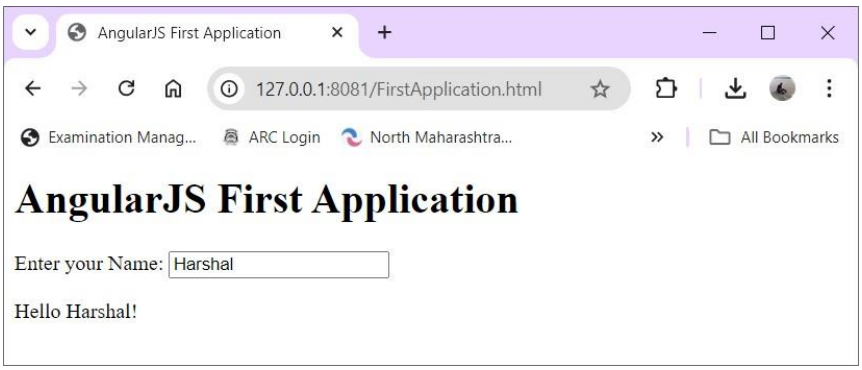
Step - 4: Bind the value of above model defined using ng-bind directive

```
<p>Hello <span ng-bind = "name"></span>!</p>
```

Complete AngularJS Example:

```
<html>
<head>
<title>AngularJS First Application</title>
<script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
</script>
</head>
<body>
<h1>AngularJS First Application</h1>
<div ng-app = "">
<p>Enter your Name: <input type = "text" ng-model = "name"></p>
<p>Hello <span ng-bind = "name"></span>!</p>
</div>
</body>
</html>
```

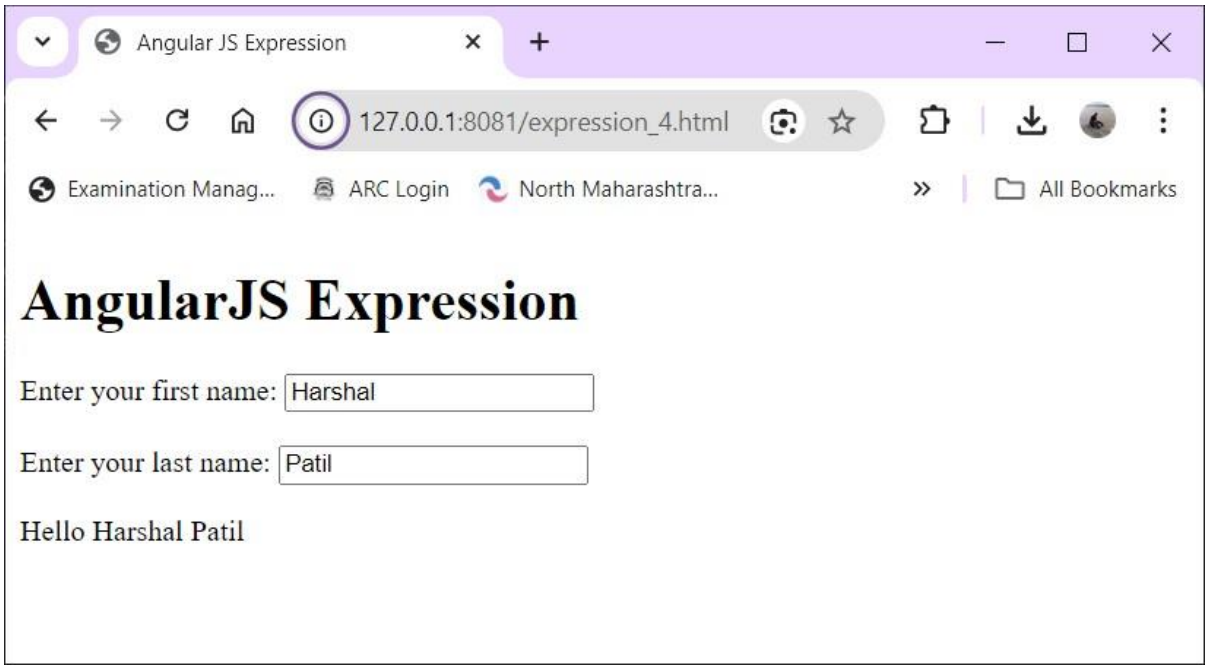
Output:



Practical 1 b) Creating Simple AngularJS Expression Application.

```
<!DOCTYPE html>
<html>
<head>
	<title>Angular JS Expression</title>
	<script src="node_modules/angular/angular.js">
	</script>
</head>

<body ng-app="">
	<form>
		<h1>AngularJS Expression</h1>
		<label>Enter your first name:</label>
		<input type="text" ng-model="firstname">
		<br><br>
		<label>Enter your last name:</label>
		<input type="text" ng-model="lastname">
	</form>
	<p>Hello {{firstname+" "+lastname}}</p>
</body>

</html>
```

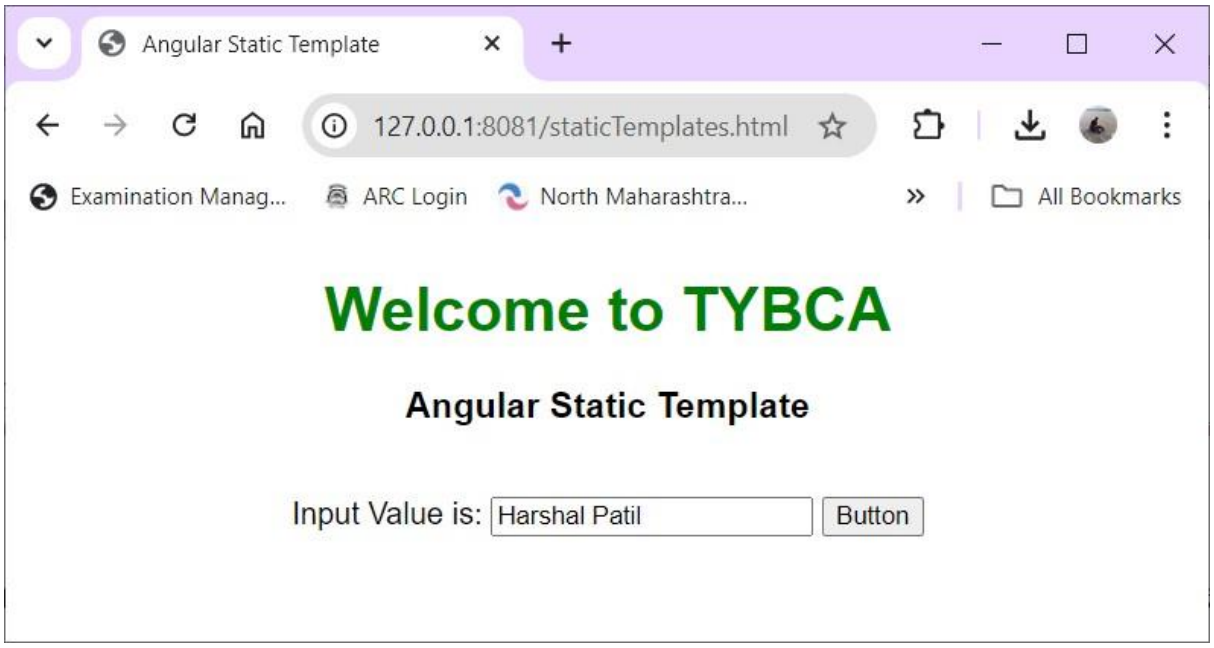
Output:

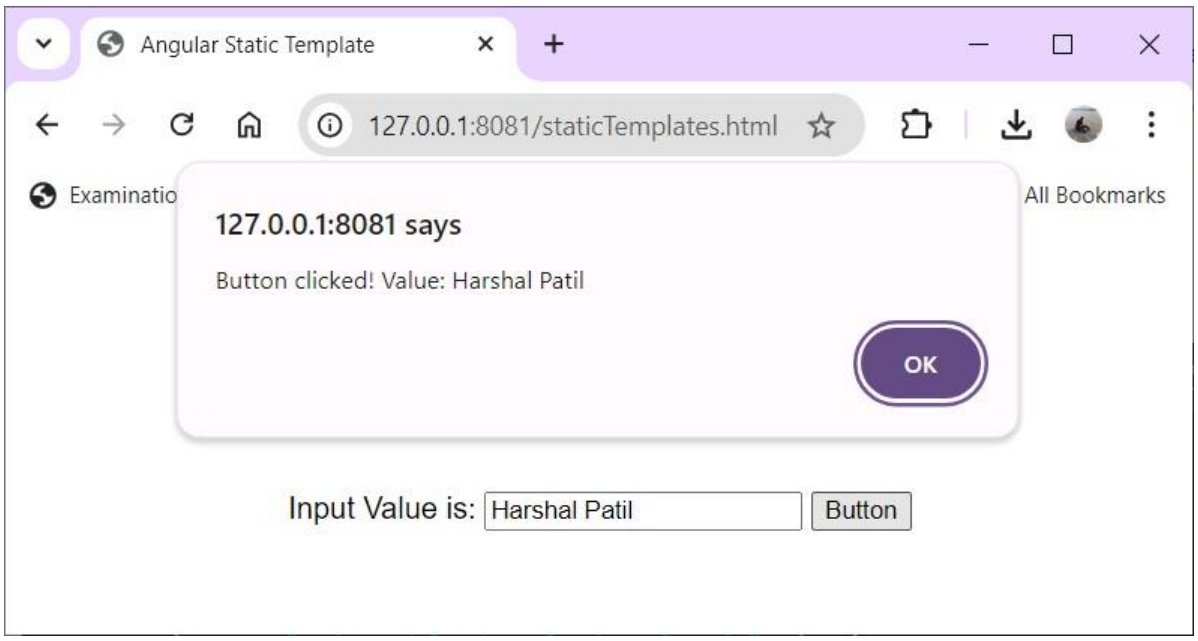


Practical 1 c) Creating Simple AngularJS Static Template Application

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
	<script src
="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
</script>
	<title>Angular Static Template</title>
	<style>
		body {
			font-family: Arial, Helvetica, sans-serif;
			text-align: center;
		}
		h1 { color: green; }
	</style>
</head>
<body ng-controller="hvpController">
	<h1>Welcome to TYBCA</h1>
	<h3>Angular Static Template</h3>
	<br> Input Value is:
	<input ng-model="hvp" value="Your Value Here">
	<button ng-click="h1()">Button</button>
</body>
<script>
	angular.module('myApp', [])
		.controller('hvpController', function($scope) {
			$scope.h1 = function() {
				alert('Button clicked! Value: ' + $scope.hvp);
			};
		});
</script>
</html>
```

Output:


Angular Static Template
127.0.0.1:8081/staticTemplates.html
Examination Manag...
ARC Login
North Maharashtra...
All Bookmarks
Welcome to TYBCA
Angular Static Template
Input Value is: Harshal Patil
Button

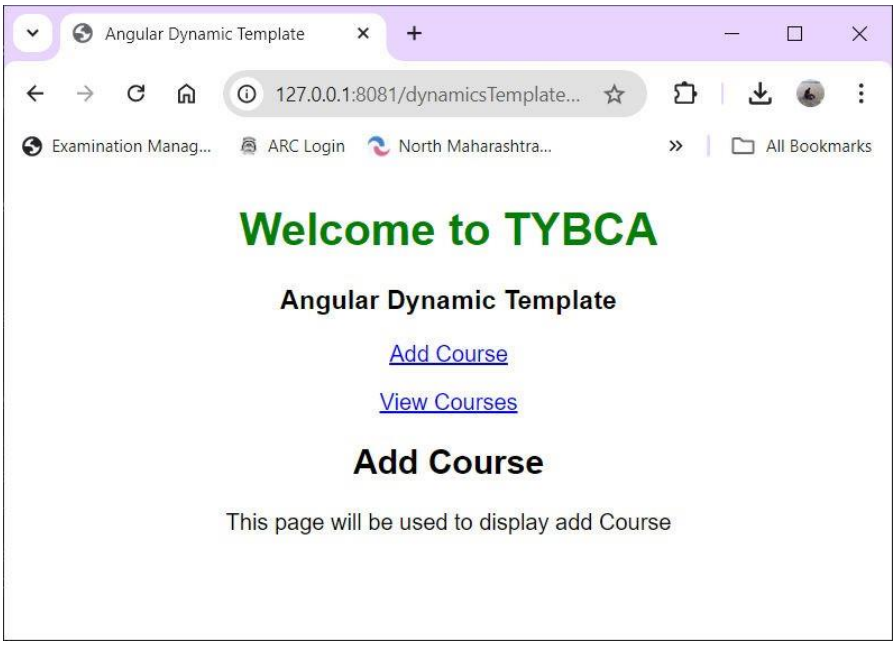

Angular Static Template
127.0.0.1:8081/staticTemplates.html
Examinatio
All Bookmarks
127.0.0.1:8081 says
Button clicked! Value: Harshal Patil
OK
Input Value is: Harshal Patil
Button

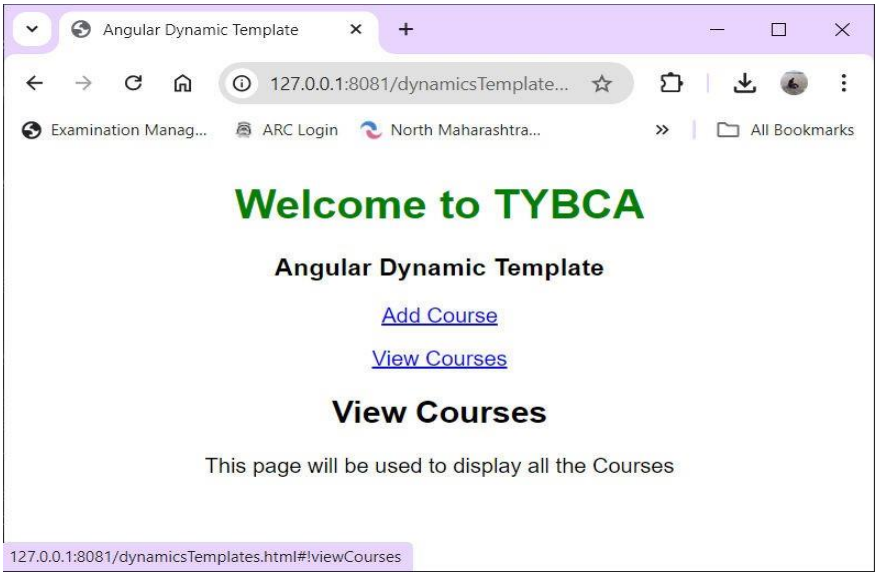
Practical 1 d) Creating Simple AngularJS Dynamic Template Application

```
<!DOCTYPE html>
<html>
<head>
   <title>Angular Dynamic Template</title>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular-
route.js"></script>
   <style>
      body {
         text-align: center;
         font-family: Arial, Helvetica, sans-serif;
      }
      h1 { color: green; }
   </style>
</head>
<body>
   <h1>Welcome to TYBCA</h1>
   <h3>Angular Dynamic Template</h3>
   <div ng-app="hvp">
      <p>
         <a href="#!addCourse">Add Course</a>
      </p>
      <p>
         <a href="#!viewCourses">View Courses</a>
      </p>
      <div ng-view></div>
      <script type="text/ng-template" id="addCourse.html">
         <h2> Add Course </h2> {{message}}
      </script>
      <script type="text/ng-template" id="viewCourses.html">
         <h2> View Courses </h2> {{message}}
      </script>
   </div>
   <script>
      var hvp = angular.module("hvp", ['ngRoute']);
      hvp.config(['$routeProvider', function ($routeProvider) {
         $routeProvider
         .when('/addCourse', {
            templateUrl: 'addCourse.html',
            controller: 'AddCourseController'
         })
         .when('/viewCourses', {
            templateUrl: 'viewCourses.html',
            controller: 'ViewCoursesController'
         })
         .otherwise({
            redirectTo: '/addCourse'
         });
      } ]);
      hvp.controller('AddCourseController', function ($scope) {
```

```
        $scope.message = "This page will be used to display add Course";
    });
    hvp.controller('ViewCoursesController', function ($scope) {
        $scope.message = "This page will be used to display all the Courses";
    });
  </script>
</body>
</html>
```

Output:





Practical 1 e):- Create a user-friendly interface with a clean and proper design.

Creating a user-friendly interface with a clean and proper design using AngularJS involves several steps, from setting up the project to implementing various components and styling them effectively. Below is a basic guide to get you started:

Step 1: Set Up Your AngularJS Project

Create Folder **Pract-1** and follow following steps.

1. Install AngularJS:

npm install angular

2. Set Up Your Project Structure:

```
Pract-1/
├── index.html
├── app/
│   ├── app.module.js
│   ├── app.config.js
│   ├── components/
│   │   ├── navbar/
│   │   │   ├── navbar.component.js
│   │   │   └── navbar.template.html
│   │   ├── home/
│   │   │   ├── home.component.js
│   │   │   └── home.template.html
│   │   └── about/
│   │       ├── about.component.js
│   │       └── about.template.html
│   └── services/
│       └── data.service.js
└── assets/
    └── styles.css
```

Step 2: Create the Basic Application Structure

1. index.html:

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
   <meta charset="utf-8">
   <title>My AngularJS App</title>
   <link rel="stylesheet" href="assets/styles.css">
   <script src="node_modules/angular/angular.min.js"></script>
   <script src="node_modules/angular-route/angular-route.min.js"></script>
    <script src="app/app.module.js"></script>
   <script src="app/app.config.js"></script>
   <script src="app/components/navbar/navbar.component.js"></script>
   <script src="app/components/home/home.component.js"></script>
   <script src="app/components/about/about.component.js"></script>
</head>
<body>
   <navbar></navbar>
   <div ng-view></div>
</body>
</html>
```

2. app.module.js:

```
(function() {
   'use strict';

   angular.module('myApp', ['ngRoute']);
})();
```

3. app.config.js:

```
(function() {
   'use strict';
   angular.module('myApp').config(['$routeProvider', function($routeProvider) {
      $routeProvider
         .when('/', {
            template: '<home></home>'
         })
```

```
            .when('/about', {
                template: '<about></about>'
            })
            .otherwise({
                redirectTo: '/'
            });
    }]);
})();
```

Step 3: Create Components

1. Navbar Component:

- navbar.component.js:

```
(function() {
    'use strict';

    angular.module('myApp').component('navbar', {
        templateUrl: 'app/components/navbar/navbar.template.html'
    });
})();
```

- navbar.template.html:

```
<nav>
    <ul>
        <li><a href="#!/">Home</a></li>
        <li><a href="#!/about">About</a></li>
    </ul>
</nav>
```

2. Home Component:

- home.component.js:

```
(function() {
    'use strict';
    angular.module('myApp').component('home',  {
        templateUrl: 'app/components/home/home.template.html'
    });
})();
```

- home.template.html:

```
<div class="container">
  <h1>Welcome to the Home Page</h1>
  <p>This is the home page of our AngularJS app.</p>
</div>
```

3. About Component:

- about.component.js:

```
(function() {
  'use strict';
  angular.module('myApp').component('about',  {
    templateUrl: 'app/components/about/about.template.html'
  });
})();
```

- about.template.html:

```
 <div class="container">
  <h1>About Us</h1>
  <p>This is the about page of our AngularJS app.</p>
</div>
```

Step 4: Add Some Basic Styling

Create a `**styles.css**` file in the `**assets**` directory and add some basic styles:

```
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  color: #333;
}
nav {
  background-color: #4CAF50;
  color: white;
  padding: 15px;
}
nav ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
```

```
}
nav ul li {
    display: inline;
    margin-right: 15px;
}
nav ul li a {
    color: white;
    text-decoration: none;
}
nav ul li a:hover {
    text-decoration: underline;
}
.container {
    padding: 20px;
}
```

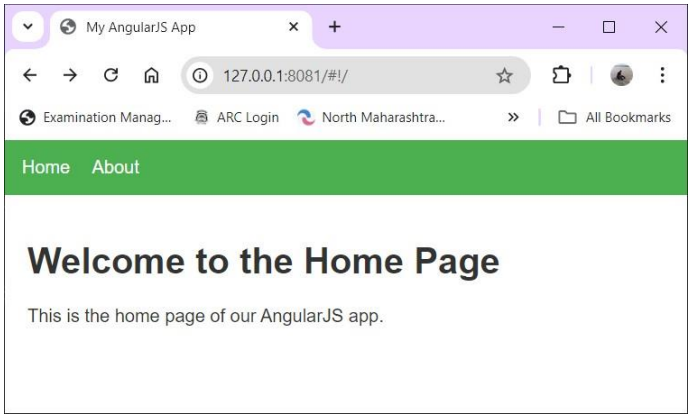
Step 5: Run Your Application

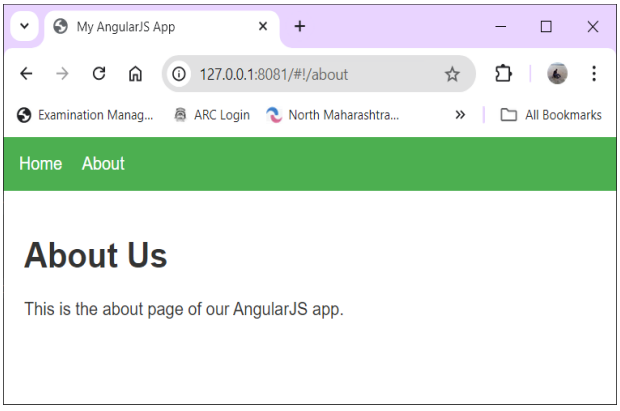
To run your AngularJS application, you can use a simple HTTP server.

http-server

Then, navigate to `http://localhost:8001` in your browser to see your AngularJS app in action.

Output:



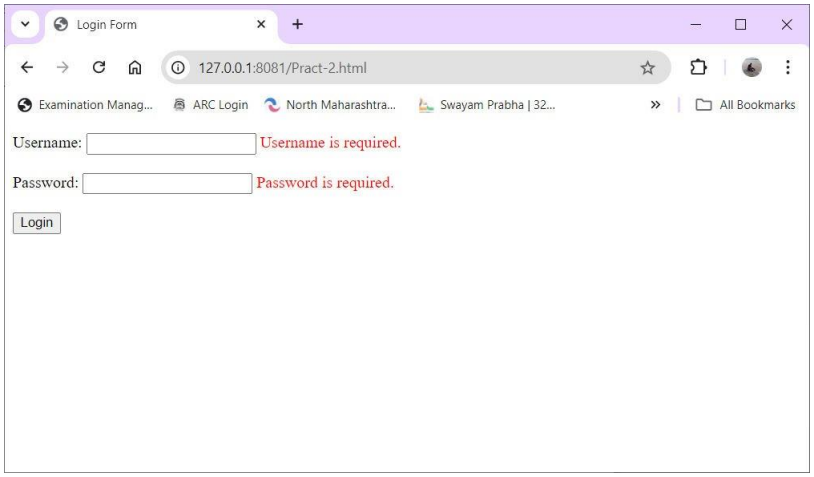


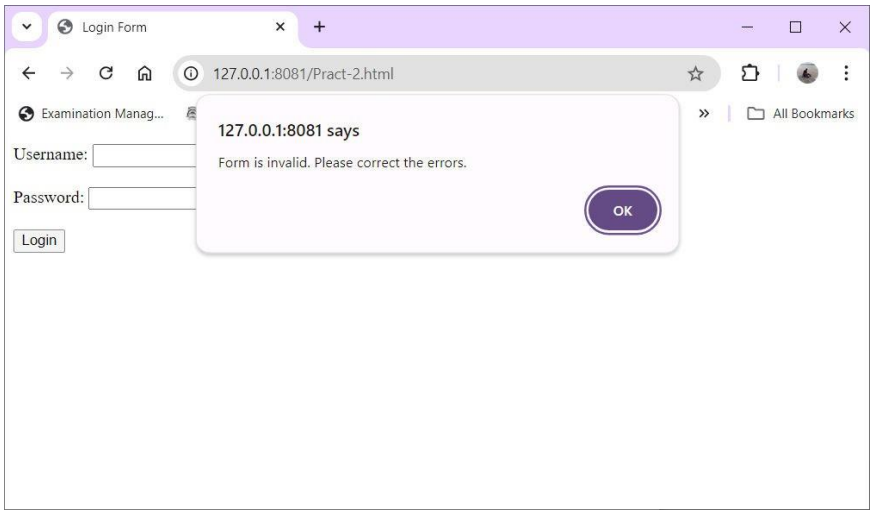
Practical 2:- Develop AngularJS program to create a login form, with validation for the username and password fields.

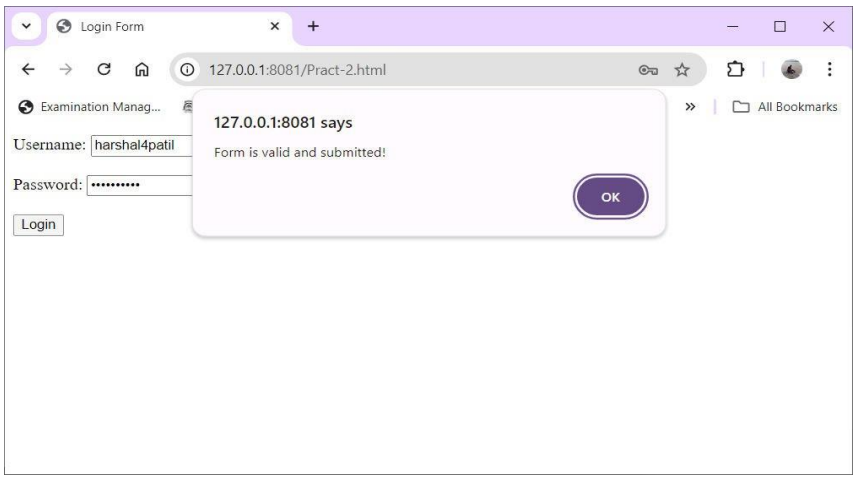
```
<!DOCTYPE html>
<html ng-app="loginApp">
<head>
   <title>Login Form</title>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
   <style>
      .error {
         color: red;
      }
   </style>
</head>
<body ng-controller="LoginController">
   <form name="loginForm" ng-submit="submitForm()" novalidate>
      <div>
         <label for="username">Username:</label>
         <input type="text" id="username" name="username" ng-
model="user.username" required>

         <span class="error" ng-show="loginForm.username.$touched &&
loginForm.username.$invalid">
            Username is required.
         </span>
      </div>
      <br>
      <div>
         <label for="password">Password:</label>
         <input type="password" id="password" name="password" ng-
model="user.password" required>
         <span class="error" ng-show="loginForm.password.$touched &&
loginForm.password.$invalid">
            Password is required.
         </span>
      </div>
      <br>
      <button type="submit">Login</button>
   </form>

   <script>
      angular.module('loginApp', [])
         .controller('LoginController', ['$scope', function($scope) {
            $scope.user = {};

            $scope.submitForm = function() {
               if ($scope.loginForm.$valid) {
                  alert('Form is valid and submitted!');
                  // Add your login logic here
               } else {
                  alert('Form is invalid. Please correct the errors.');
```

```
            }
        };
    }]);
  </script>
</body>
</html>
```

Output:





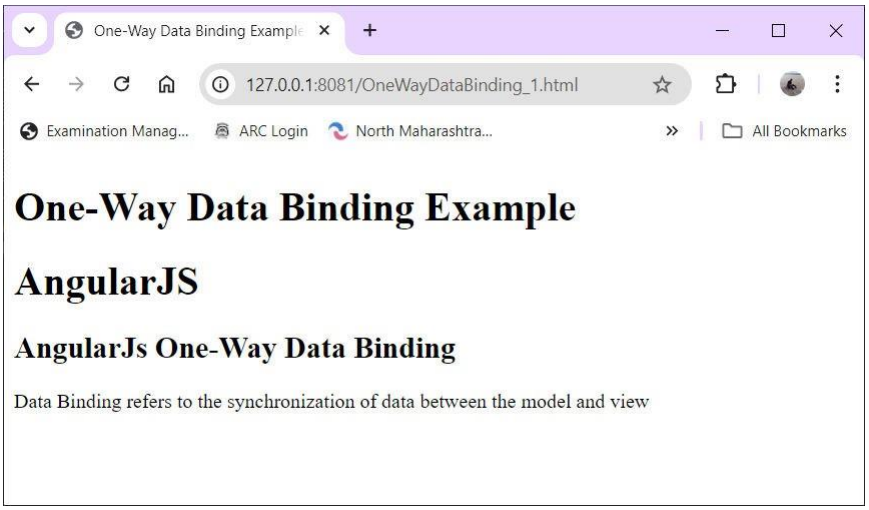


Practical 3:- To demonstrate AngularJS services and data bind.

Practical – 3 a) To demonstrate AngularJS One Way Data Binding.

```
<!DOCTYPE html>
<html>
<head>
  <title>One-Way Data Binding Example</title>
  <script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
  </script>
</head>
<body ng-app="myApp">
<h1>One-Way Data Binding Example</h1>
  <div ng-controller="myCtrl">
          <h1>{{title1}}</h1>
          <h2>{{title2}}</h2>
          <p>{{description}}</p>
  </div>
  <script type="text/javascript">
     var app = angular.module("myApp", []);
     app.controller("myCtrl", function ($scope) {
        $scope.title1 = "AngularJS";
        $scope.title2 = "AngularJs One-Way Data Binding";
        $scope.description = "Data Binding refers "
                  + "to the synchronization of data "
                  + "between the model and view";
     });
  </script>
</body>
</html>
```

Output:



Practical – 3 b) To demonstrate AngularJS One Way Data Binding.

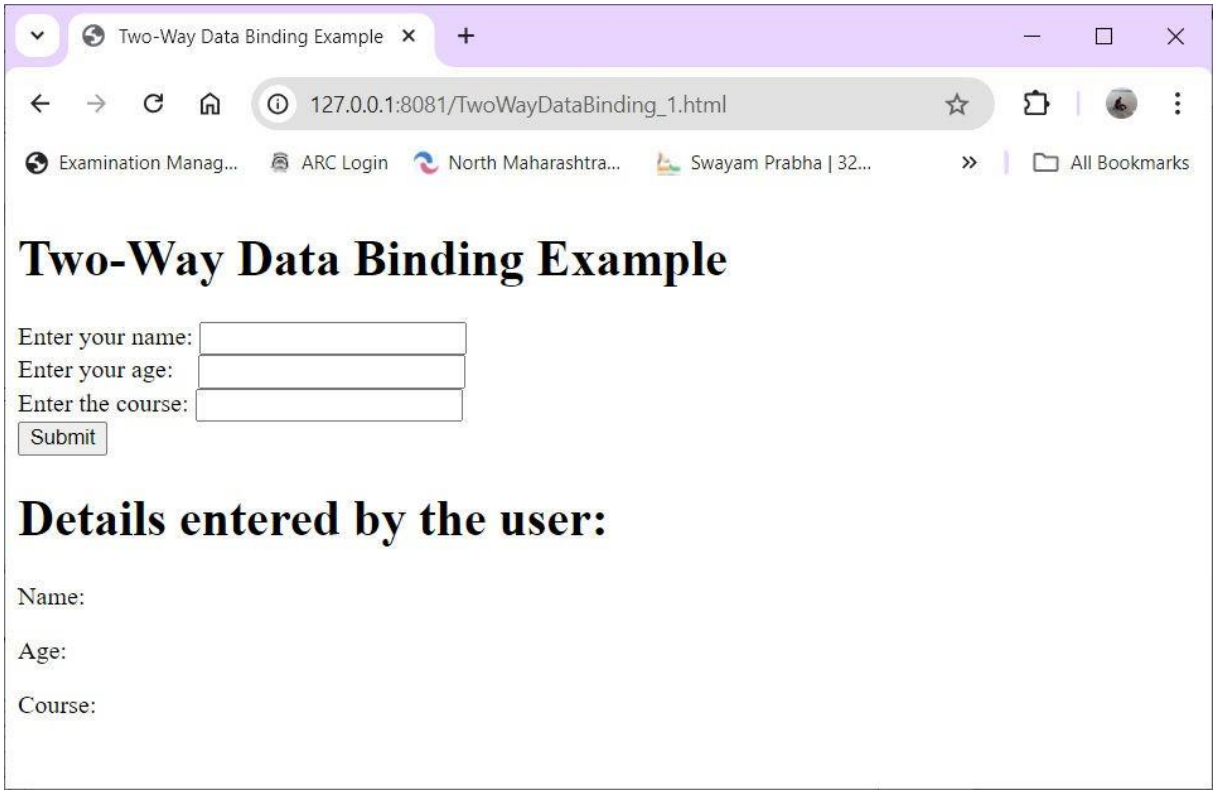
```
<!DOCTYPE html>
<html>
<head>
	<title>One-Way Data Binding Example</title>
	<script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
	</script>
</head>
<body ng-app="myApp">
<h1>One-Way Data Binding Example</h1>
	<div ng-controller="myCtrl">
		<p>Firstname:
			<span ng-bind="firstname"></span>
		</p>
		<p>Lastname:
			<span ng-bind="lastname"></span>
		</p>
	</div>
	<script type="text/javascript">
	  var app = angular.module("myApp", []);
	  app.controller("myCtrl", function ($scope) {
	    $scope.firstname = "Purva";
	    $scope.lastname = "Patil";
	  });
	</script>
</body>
</html>
```

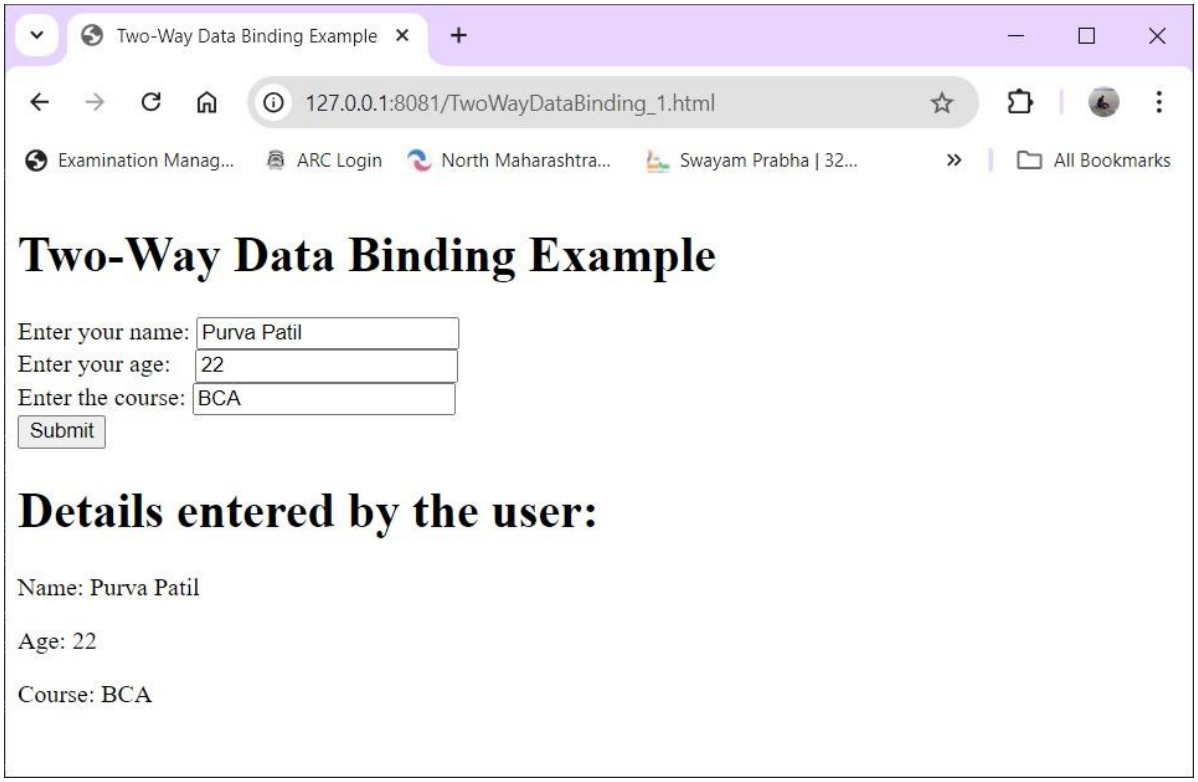
Output:

Practical – 3 c) To demonstrate AngularJS Two Way Data Binding.

```
<!DOCTYPE html>
<html>

<head>
	<title>Two-Way Data Binding Example</title>
	<script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
	</script>
</head>
<body ng-app="myApp">
<h1>Two-Way Data Binding Example</h1>
	<div ng-controller="myCtrl">
		<form>
			<label>Enter your name:</label>
			<input type="text" ng-model="name">
			<br>
			<label>Enter your age:</label>
			<input type="number" ng-model="age">
			<br>
			<label>Enter the course you are interested in:</label>
			<input type="text" ng-model="course">
			<br>
			<input type="submit" ng-click="details()">
		</form>
		<div ng-show="showdetails">
			<h1>Details entered by the user:</h1>
			<p>Name: {{name}}</p>
			<p>Age: {{age}}</p>
			<p>Course: {{course}}</p>
		</div>
	</div>
	<script type="text/javascript">
	   var app = angular.module("myApp", []);
	   app.controller("myCtrl", function ($scope) {
	      $scope.showdetails = false;
	      $scope.details = function () {
	         $scope.showdetails = true;
	      }
	   });
	</script>
</body>
</html>
```

Output:


Two-Way Data Binding Example
127.0.0.1:8081/TwoWayDataBinding_1.html
Examination Manag...
ARC Login
North Maharashtra...
Swayam Prabha | 32...
All Bookmarks
Two-Way Data Binding Example
Enter your name:
Enter your age:
Enter the course:
Submit
Details entered by the user:
Name:
Age:
Course:

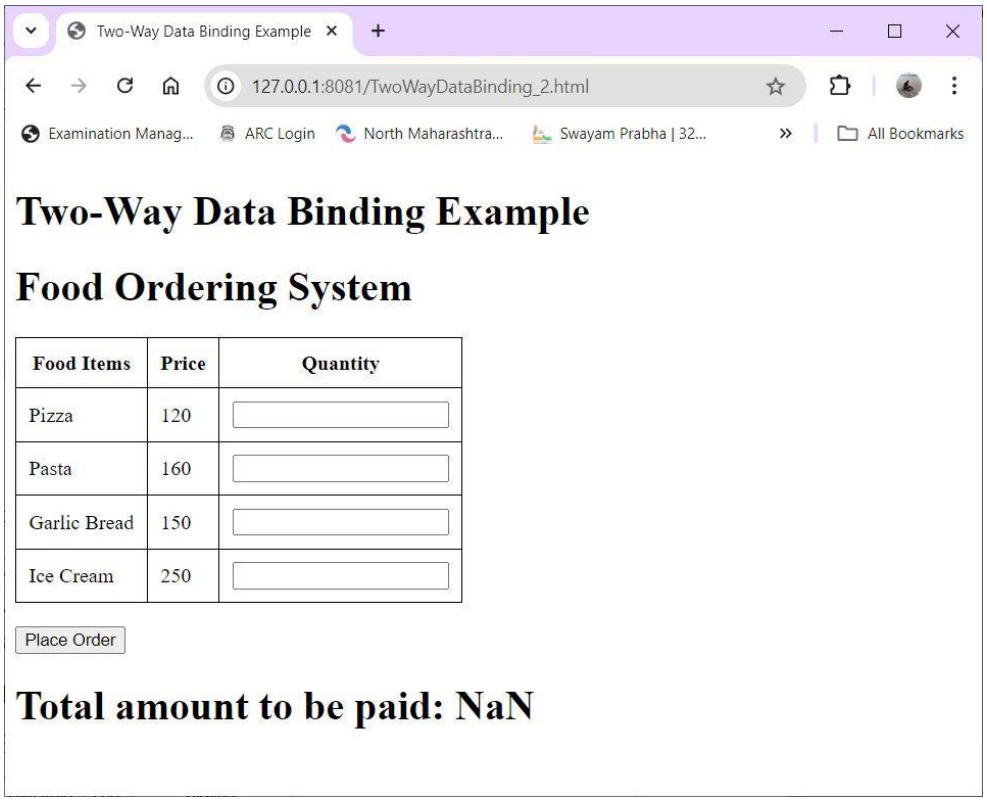

Two-Way Data Binding Example
127.0.0.1:8081/TwoWayDataBinding_1.html
Examination Manag...
ARC Login
North Maharashtra...
Swayam Prabha | 32...
All Bookmarks
Two-Way Data Binding Example
Enter your name: Purva Patil
Enter your age: 22
Enter the course: BCA
Submit
Details entered by the user:
Name: Purva Patil
Age: 22
Course: BCA

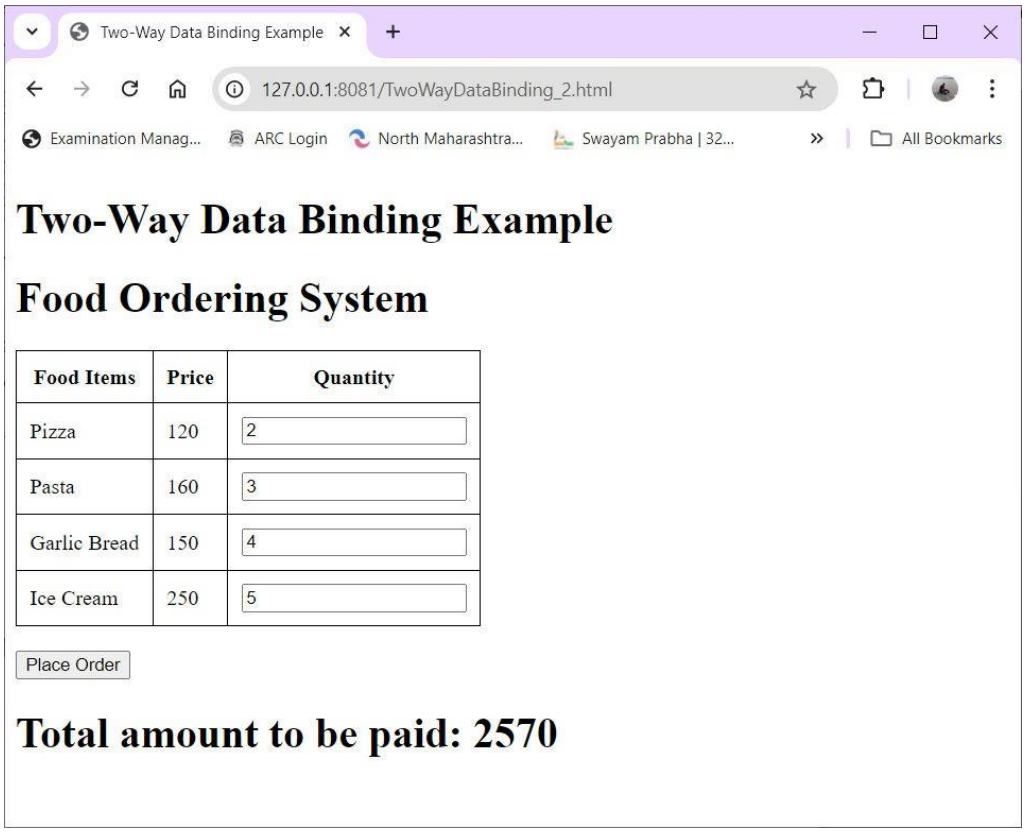
Practical – 3 d) To demonstrate AngularJS Two Way Data Binding.

```
<!DOCTYPE html>
<html>
<head>
	<title>Two-Way Data Binding Example</title>
	<script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
	</script>
	<style>
		table {
			border: 1px solid black;
			border-collapse: collapse;
		}

		tr, th, td {
			border: 1px solid black;
			border-collapse: collapse;
			padding: 10px;
		}
	</style>
</head>
<body ng-app="myApp">
<h1>Two-Way Data Binding Example</h1>
	<div ng-controller="myCtrl">
   <h1>Food Ordering System</h1>
		<table>
			<tr>
				<th>Food Items</th>
				<th>Price</th>
				<th>Quantity</th>
			</tr>
			<tr>
				<td>Pizza</td>
				<td>120</td>
				<td>
					<input type="number"
						ng-model="pizza">
				</td>
			</tr>
			<tr>
				<td>Pasta</td>
				<td>160</td>
				<td>
					<input type="number"
						ng-model="pasta">
				</td>
			</tr>
			<tr>
				<td>Garlic Bread</td>
				<td>150</td>
				<td>
					<input type="number"
```

```
                                        ng-model="garlicbread">
                        </td>
                    </tr>
                    <tr>
                        <td>Ice Cream</td>
                        <td>250</td>
                        <td>
                            <input type="number"
                                ng-model="icecream">
                        </td>
                    </tr>
                </table>
                <br>
                <button ng-click="calculate()">
                    Place Order
                </button>
                <div ng-show="amt!=0">
                    <h1>
                        Total amount to be paid: {{amt}}
                    </h1>
                </div>
        </div>
        <script type="text/javascript">
            var app = angular.module("myApp", []);
            app.controller("myCtrl", function ($scope) {
                $scope.title = "Food ordering";
                $scope.amt = 0;
                $scope.calculate = function () {
                    $scope.amt = 120 * $scope.pizza
                                        + 160 * $scope.pasta
                                        + 150 * $scope.garlicbread
                                        + 250 * $scope.icecream;
                }
            });
        </script>
</body>
</html>
```

Output:


Two-Way Data Binding Example
127.0.0.1:8081/TwoWayDataBinding_2.html
Examination Manag...
ARC Login
North Maharashtra...
Swayam Prabha | 32...
All Bookmarks
Two-Way Data Binding Example
Food Ordering System
Food Items
Price
Quantity
Pizza
120
Pasta
160
Garlic Bread
150
Ice Cream
250
Place Order
Total amount to be paid: NaN


Two-Way Data Binding Example
127.0.0.1:8081/TwoWayDataBinding_2.html
Examination Manag...
ARC Login
North Maharashtra...
Swayam Prabha | 32...
All Bookmarks
Two-Way Data Binding Example
Food Ordering System
Food Items
Price
Quantity
Pizza
120
2
Pasta
160
3
Garlic Bread
150
4
Ice Cream
250
5
Place Order
Total amount to be paid: 2570

Practical – 3 e) To demonstrate AngularJS Built-in Services - $http Services.

//DataService.txt - Data File

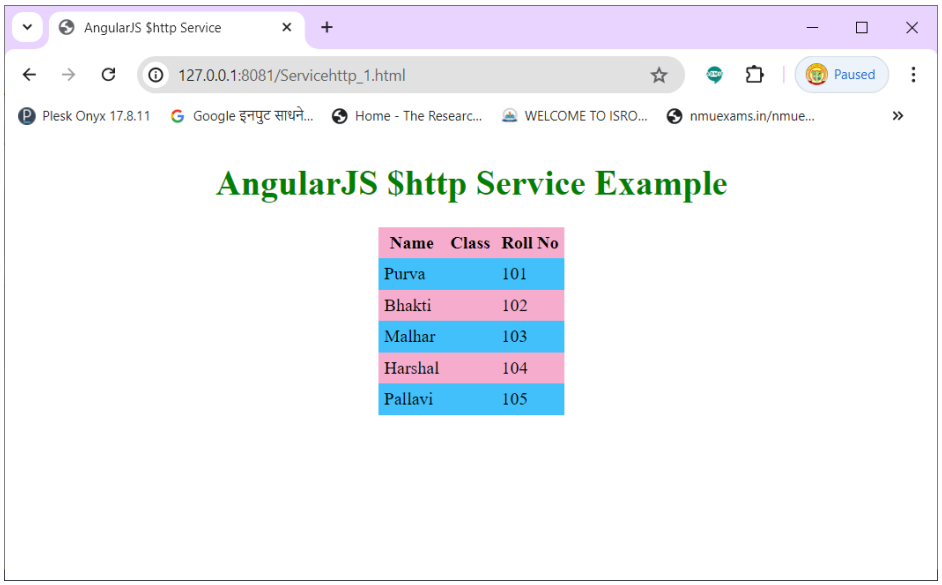
```
[
  {
    "Name" : "Purva",
    "Class" : "TYBCA",
    "RollNo" : 101
  },
  {
    "Name" : "Bhakti",
    "Class" : "TYBCA",
    "RollNo" : 102
  },
  {
    "Name" : "Malhar",
    "Class" : "TYBCA",
    "RollNo" : 103
  },
  {
    "Name" : "Harshal",
    "Class" : "TYBCA",
    "RollNo" : 104
  },
  {
    "Name" : "Pallavi",
    "Class" : "TYBCA",
    "RollNo" : 105
  }
]
```

//Servicehttp_1.html - Code File

```
<!DOCTYPE html>
<html>
<head>
  <title>AngularJS $http Service</title>
  <style>
    table, th, td {
      border: 1px #2E0854;
      border-collapse: collapse;
      padding: 5px;
    }

    table tr:nth-child(odd) {
      background-color: #F6ADCD;
    }

    table tr:nth-child(even) {
      background-color: #42C0FB;
    }
  </style>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js"><
/script>
  <script>
    var app = angular.module('myApp', []);

    app.controller('studentController', function ($scope, $http) {
      var url = "/DataService.txt";

      $http.get(url).then(function (response) {
        $scope.students = response.data;
      });
    });
  </script>
</head>
<body>
<center>
  <h1 style="color:green">AngularJS $http Service Example</h1>

  <div ng-app="myApp" ng-controller="studentController">
    <table>
      <tr>
        <th>Name</th>
        <th>Class</th>
        <th>Roll No</th>
      </tr>
      <tr ng-repeat="student in students">
        <td>{{ student.Name }}</td>
        <td>{{ student.Clas}}</td>
        <td>{{ student.RollNo}}</td>
      </tr>
```

```
        </table>
    </div>
</center>
</body>
</html>
```

Output:



Practical – 3 f) To demonstrate AngularJS Built-in Services - $resource Services.

1. Create a new project directory and initialize a new Node.js project:

- mkdir student-api
- cd student-api
- npm init –y

2. Create the Server Script in student-api directory:

Create a file named **server.js** and add the following code:

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();
const port = 3000;

app.use(bodyParser.json());
app.use(cors());

let students = [
    { id: 1, name: "Purva", class: "TYBCA" },
    { id: 2, name: "Bhakti", class: "TYBCA" },
    { id: 3, name: "Malhar", class: "TYBCA" }
];

// GET: Fetch all students
app.get('/api/students', (req, res) => {
    res.json(students);
});

// POST: Add a new student
app.post('/api/students', (req, res) => {
    const newStudent = req.body;
    newStudent.id = students.length ? students[students.length - 1].id + 1 : 1;
    students.push(newStudent);
    res.status(201).json(newStudent);
});

// DELETE: Delete a student by ID
app.delete('/api/students/:id', (req, res) => {
    const studentId = parseInt(req.params.id, 10);
    students = students.filter(student => student.id !== studentId);
    res.status(204).send();
});

app.listen(port, () => {
    console.log(`Server running at http://localhost:${port}/`);
});
```

3. Include the AngularJS Resource module: To use `$resource`, you need to include the `angular-resource.js` script in your HTML file.

In HTML File:

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular-resource.min.js"></script>

ServiceResouece_1.html File:

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>AngularJS $resource Example</title>
  <link rel="icon" href="favicon.ico">
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></sc
ript>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular-
resource.min.js"></script>
  <script src="app.js"></script>
</head>
<body ng-controller="studentController">
  <h1>AngularJS $resource Example</h1>
  <h2>Students Details</h2>
  <ul>
    <li ng-repeat="student in students">
      {{student.name}} ({{student.class}})
      <button ng-click="deleteStudent(student)">Delete</button>
    </li>
  </ul>

  <h2>Add New Student</h2>
  <form ng-submit="addStudent()">
    <label>Name:</label>
    <input type="text" ng-model="newStudent.name" required>
    <label>Class:</label>
    <input type="text" ng-model="newStudent.class" required>
    <button type="submit">Add Student</button>
  </form>
</body>
</html>
```

4. Add `ngResource` as a dependency: Include `ngResource` in your AngularJS module.

In Javascript File

var app = angular.module('myApp', ['ngResource']);

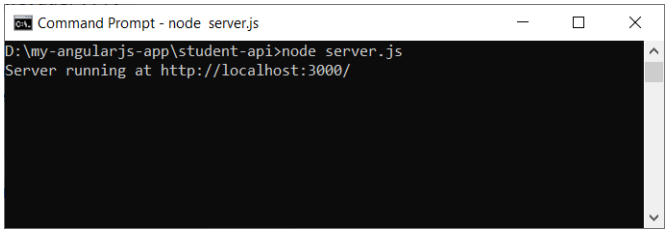
JavaScript (`app.js`):

```
var app = angular.module('myApp', ['ngResource']);
```

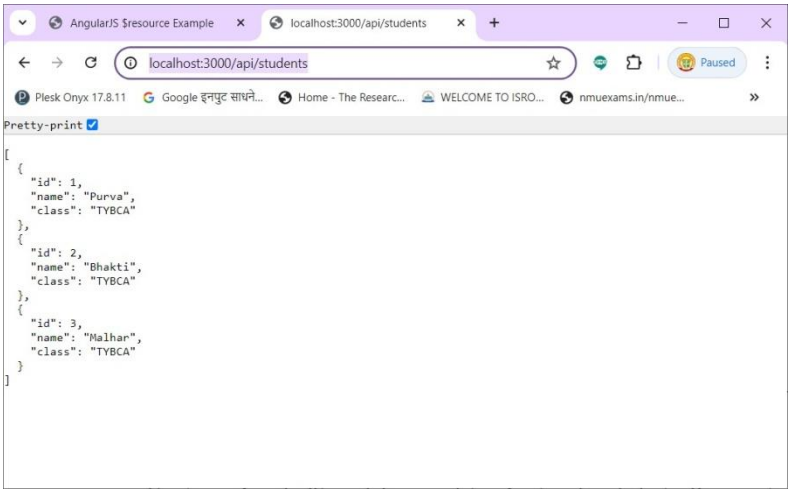
```
app.controller('studentController', function ($scope, $resource) {
  var Student = $resource('http://localhost:3000/api/students/:id', { id: '@id'
}, {
    update: {
      method: 'PUT'
    }
  });

  // Fetch all students
  $scope.students = Student.query();

  // Add a new student
  $scope.addStudent = function () {
    var newStudent = new Student();
    newStudent.name = $scope.newStudent.name;
    newStudent.class = $scope.newStudent.class;

    newStudent.$save(function () {
      $scope.students.push(newStudent);
      $scope.newStudent = {}; // Clear the form
    });
  };

  // Delete a student
  $scope.deleteStudent = function (student) {
    student.$delete(function () {
      var index = $scope.students.indexOf(student);
      if (index > -1) {
        $scope.students.splice(index, 1);
      }
    });
  };
});
```

Output:

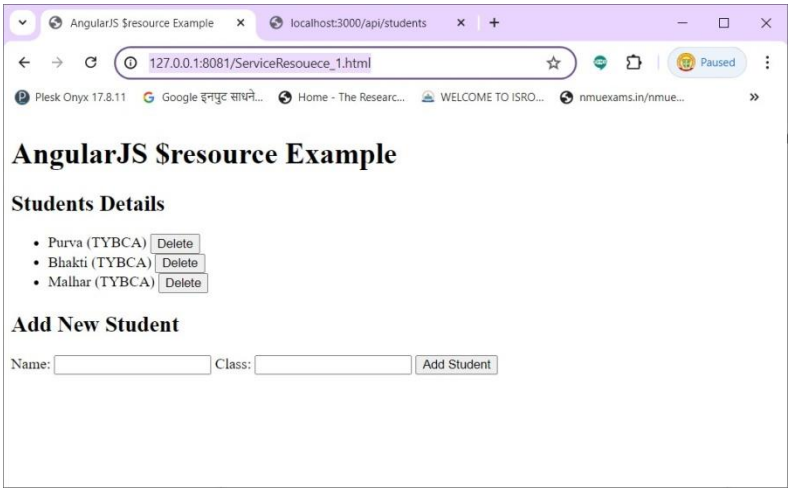
1. **Start the server: Run the server script using Node.js:**
   - **node server.js**



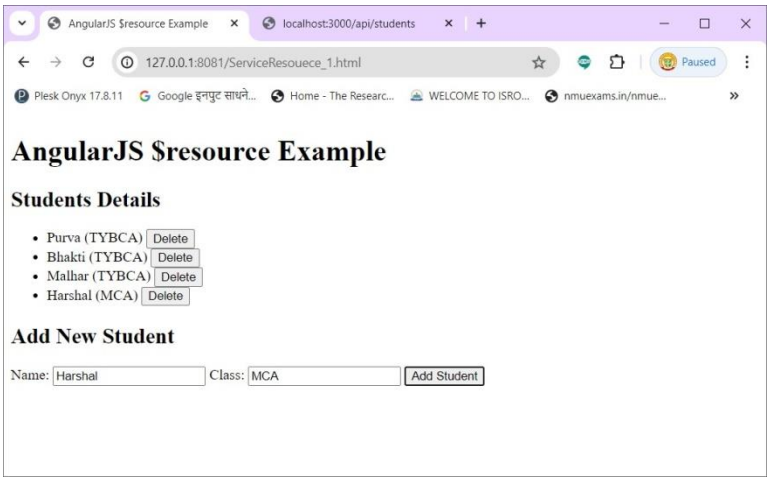
2. **Open browser and type url – http://localhost:3000/api/students**



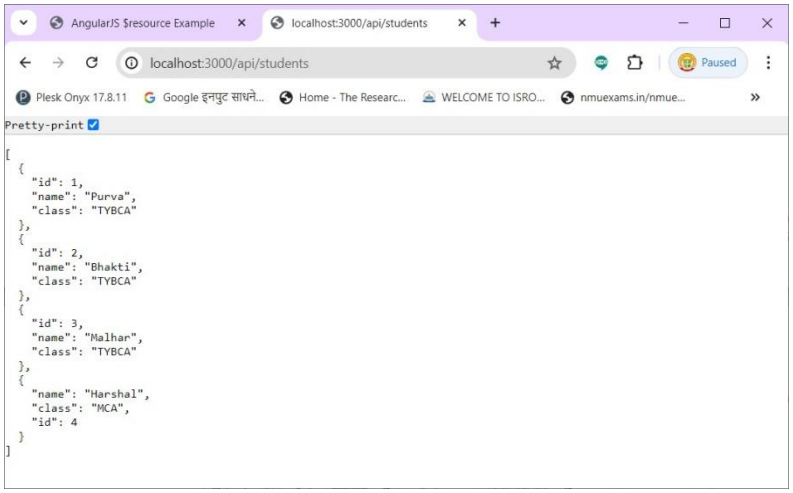
3. **Open browser run HTML file – http://127.0.0.1:8081/ServiceResouece_1.html**



4. **Add New Student:**



5. New Student successfully added on server side:



Practical – 3 g) To demonstrate AngularJS Custome Services using Service Method.

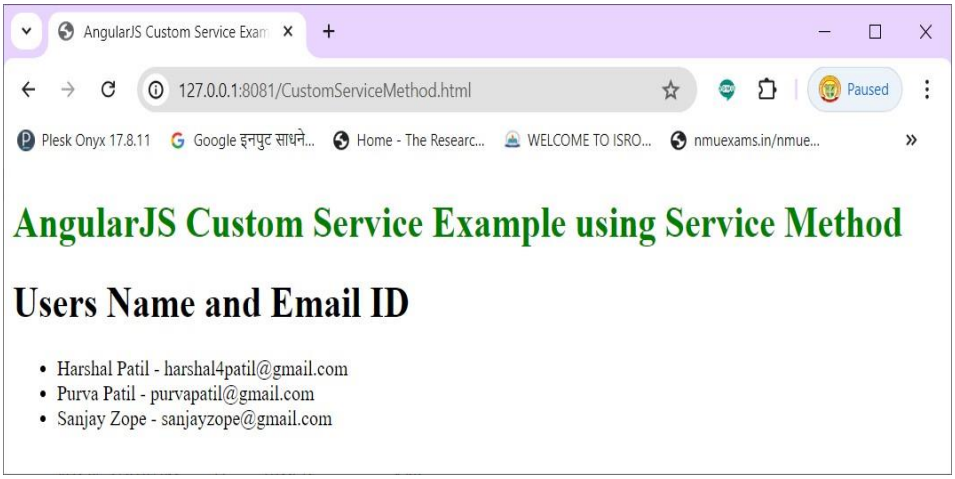
//CustomServiceMethod.js

```
// JScript source code
// Define the module
var app = angular.module('myApp', []);

// Define the custom service using the service method
app.service('UserService', function () {
   var users = [
      { name: 'Harshal Patil', email: 'harshal4patil@gmail.com' },
      { name: 'Purva Patil', email: 'purvapatil@gmail.com' },
      { name: 'Sanjay Zope', email: 'sanjayzope@gmail.com' }
   ];
   return {
      getUsers: function () {
         return users;
      }
   };
});

// Define the controller and inject the custom service
app.controller('MainController', ['$scope', 'UserService', function ($scope,
UserService) {
   $scope.users = UserService.getUsers();
} ]);
```

//CustomServiceMethod.html

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
   <title>AngularJS Custom Service Example using Service Method</title>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></sc
ript>
   <script src="CustomServiceMethod.js"></script>
</head>
<body style="text-align:justify">
<h1 style="color:green">
AngularJS Custom Service Example using Service Method
</h1>
   <div ng-controller="MainController">
      <h1>Users Name and Email ID</h1>
      <ul>
         <li ng-repeat="user in users">{{ user.name }} - {{ user.email }}</li>
      </ul>
   </div>
</body>
</html>
```

Output:


AngularJS Custom Service Exam
127.0.0.1:8081/CustomServiceMethod.html
Paused
Plesk Onyx 17.8.11
Google इनपुट साधने...
Home - The Researc...
WELCOME TO ISRO...
nmuexams.in/nmue...
AngularJS Custom Service Example using Service Method
Users Name and Email ID
Harshal Patil - harshal4patil@gmail.com
Purva Patil - purvapatil@gmail.com
Sanjay Zope - sanjayzope@gmail.com

Practical – 3 h) To demonstrate AngularJS Custome Services using Provider.

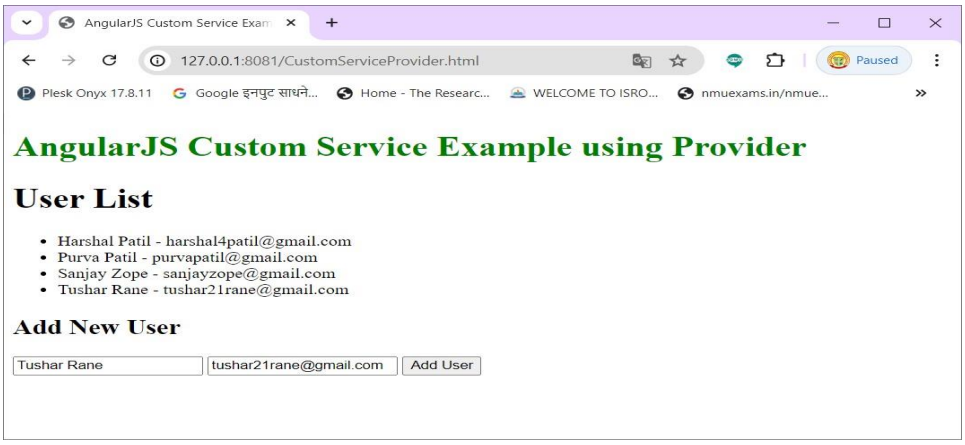
CustomServiceProvider.js

```
// Define the module
var app = angular.module('userApp', []);

// Configure the provider
app.config(['UserProviderProvider', function (UserProviderProvider) {
   // Set the initial users
   UserProviderProvider.setInitialUsers([
      { name: 'Harshal Patil', email: 'harshal4patil@gmail.com' },
      { name: 'Purva Patil', email: 'purvapatil@gmail.com' },
      { name: 'Sanjay Zope', email: 'sanjayzope@gmail.com' }
   ]);
} ]);

// Define the provider
app.provider('UserProvider', function () {
   var users = [];

   // Method to set the initial users
   this.setInitialUsers = function (initialUsers) {
      users = initialUsers;
   };

   // The $get method which returns the service instance
   this.$get = function () {
      return {
         getUsers: function () {
            return users;
         },
         addUser: function (user) {
            users.push(user);
         }
      };
   };
});

// Define the controller and inject the provider
app.controller('UserController', ['$scope', 'UserProvider', function ($scope,
UserProvider) {
   $scope.users = UserProvider.getUsers();

   $scope.newUser = { name: '', email: '' };

   $scope.addUser = function () {
      UserProvider.addUser($scope.newUser);
      $scope.newUser = { name: '', email: '' };  // Reset the form
   };
} ]);
```

CustomServiceProvider.html

```
<!DOCTYPE html>
<html ng-app="userApp">
<head>
  <title>AngularJS Custom Service Example using Provider</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></sc
ript>
  <script src="CustomServiceProvider.js"></script>
</head>
<body style="text-align:justify">
  <h1 style="color:green">
    AngularJS Custom Service Example using Provider
  </h1>
  <div ng-controller="UserController">
    <h1>User List</h1>
    <ul>
      <li ng-repeat="user in users">{{ user.name }} - {{ user.email }}</li>
    </ul>
    <h2>Add New User</h2>
    <form ng-submit="addUser()">
      <input type="text" ng-model="newUser.name" placeholder="Name"
required>
      <input type="email" ng-model="newUser.email" placeholder="Email"
required>
      <button type="submit">Add User</button>
    </form>
  </div>
</body>
</html>
```

Output:



Practical – 3 i) To demonstrate AngularJS Custome Services using Factory.

CustomServiceFactory.js

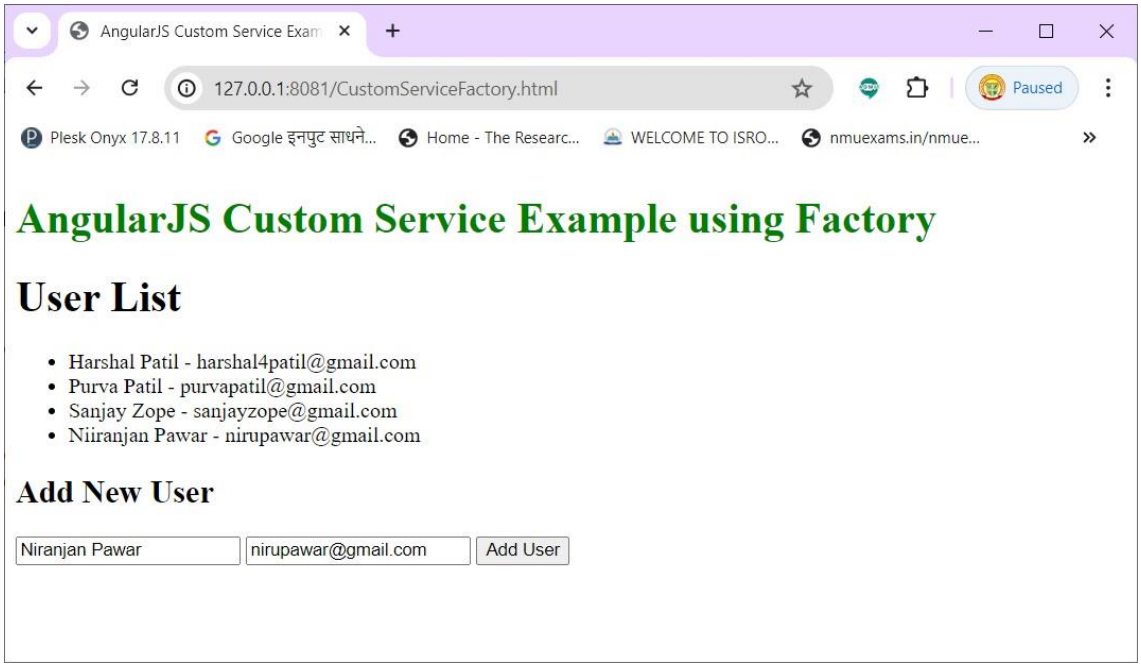
```
// Define the module
var app = angular.module('userApp', []);

// Define the factory
app.factory('UserFactory', function () {
    var users = [
        { name: 'Harshal Patil', email: 'harshal4patil@gmail.com' },
        { name: 'Purva Patil', email: 'purvapatil@gmail.com' },
        { name: 'Sanjay Zope', email: 'sanjayzope@gmail.com' }
    ];

    return {
        getUsers: function () {
            return users;
        },
        addUser: function (user) {
            users.push(user);
        }
    };
});

// Define the controller and inject the factory
app.controller('UserController', ['$scope', 'UserFactory', function ($scope,
UserFactory) {
    $scope.users = UserFactory.getUsers();

    $scope.newUser = { name: '', email: '' };

    $scope.addUser = function () {
        UserFactory.addUser($scope.newUser);
        $scope.newUser = { name: '', email: '' };  // Reset the form
    };
} ]);
```

CustomServiceFactory.html

```
<!DOCTYPE html>
<html ng-app="userApp">
<head>
    <title>AngularJS Custom Service Example using Factory</title>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></sc
ript>
    <script src="CustomServiceFactory.js"></script>
</head>
<body style="text-align:justify">
    <h1 style="color:green">
        AngularJS Custom Service Example using Factory
```

```
    </h1>
    <div ng-controller="UserController">
      <h1>User List</h1>
      <ul>
        <li ng-repeat="user in users">{{ user.name }} - {{ user.email }}</li>
      </ul>
      <h2>Add New User</h2>
      <form ng-submit="addUser()">
        <input type="text" ng-model="newUser.name" placeholder="Name"
required>
        <input type="email" ng-model="newUser.email" placeholder="Email"
required>
        <button type="submit">Add User</button>
      </form>
    </div>
</body>
</html>
```

Output:



Practical 4:- To display tasks in a list with checkboxes for marking completion as per user choice.

//Pract-4.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Task List</title>
   <link rel="stylesheet" href="styles.css">
</head>
<body>
   <div class="container">
      <h1>Task List</h1>
      <ul id="task-list">
         <!-- Tasks will be dynamically inserted here -->
      </ul>
      <input type="text" id="task-input" placeholder="Add a new task">
      <button onclick="addTask()">Add Task</button>
   </div>
   <script src="script.js"></script>
</body>
</html>
```

//Style.css

```
body {
   font-family: Arial, sans-serif;
   background-color: #f4f4f4;
   display: flex;
   justify-content: center;
   align-items: center;
   height:  100vh;
   margin: 0;
}

.container {
   background: white;
   padding: 20px;
   border-radius: 8px;
   box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h1 {
   margin-top: 0;
}

ul {
   list-style-type: none;
   padding: 0;
```

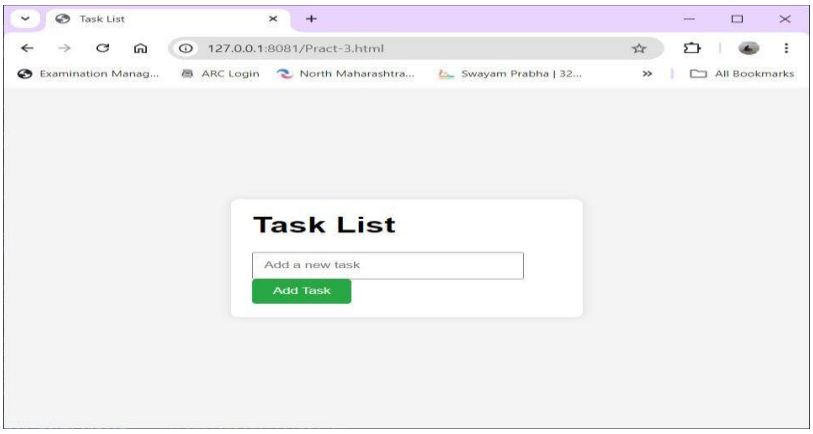
```
}

li {
    display: flex;
    align-items: center;
    padding: 10px 0;
}

input[type="text"] {
    padding: 10px;
    width: 80%;
    margin-right: 10px;
}

button {
    padding: 10px 20px;
    background-color: #28a745;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

button:hover {
    background-color: #218838;
}

input[type="checkbox"] {
    margin-right: 10px;
}
```

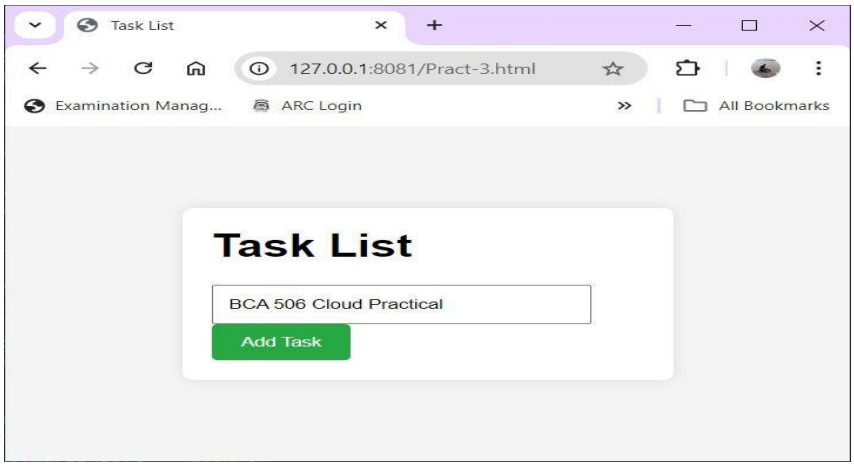
//Script.css

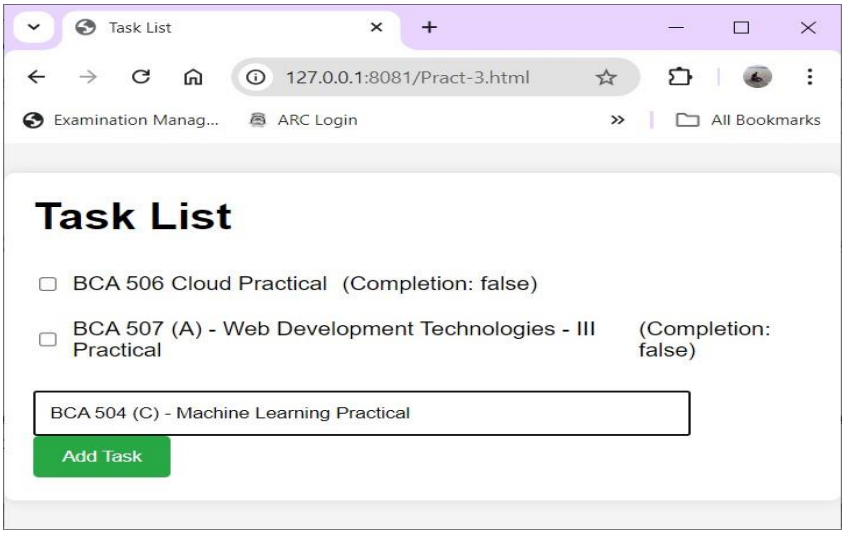
```
document.addEventListener('DOMContentLoaded', () => {
    const taskList = document.getElementById('task-list');
    const taskInput = document.getElementById('task-input');

    function addTask() {
        const taskText = taskInput.value.trim();
        if (taskText === '') return;

        // Create a new list item
        const li = document.createElement('li');

        // Create checkbox
        const checkbox = document.createElement('input');
        checkbox.type = 'checkbox';

        // Create span for task text
        const span = document.createElement('span');
        span.textContent = taskText;
```

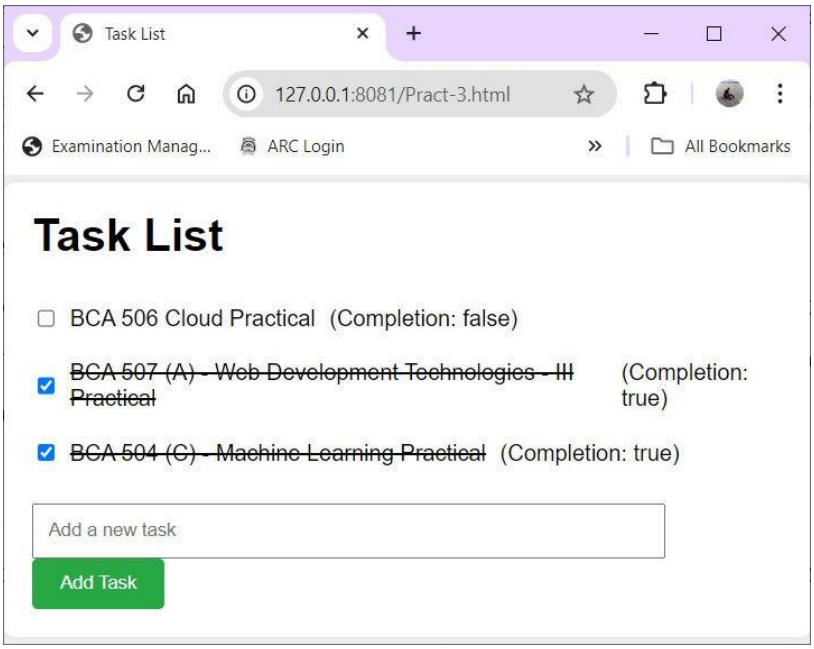
```
        // Create span for completion status
        const statusSpan = document.createElement('span');
        statusSpan.textContent = ' (Completion: false)';
        statusSpan.style.marginLeft = '10px';

        // Handle checkbox change
        checkbox.addEventListener('change', () => {
            if (checkbox.checked) {
                span.style.textDecoration = 'line-through';
                statusSpan.textContent = ' (Completion: true)';
            } else {
                span.style.textDecoration = 'none';
                statusSpan.textContent = ' (Completion: false)';
            }
        });

        // Append elements to list item
        li.appendChild(checkbox);
        li.appendChild(span);
        li.appendChild(statusSpan);

        // Append list item to task list
        taskList.appendChild(li);

        // Clear input field
        taskInput.value = '';
    }

    // Make addTask function globally accessible
    window.addTask = addTask;
});
```

Output:




Task List
127.0.0.1:8081/Pract-3.html
Examination Manag...
ARC Login
All Bookmarks
Task List
BCA 506 Cloud Practical
Add Task


Task List
127.0.0.1:8081/Pract-3.html
Examination Manag...
ARC Login
All Bookmarks
Task List
BCA 506 Cloud Practical (Completion: false)
BCA 507 (A) - Web Development Technologies - III Practical (Completion: false)
BCA 504 (C) - Machine Learning Practical
Add Task


Task List
127.0.0.1:8081/Pract-3.html
Examination Manag...
ARC Login
All Bookmarks
Task List
BCA 506 Cloud Practical (Completion: false)
~~BCA 507 (A) - Web Development Technologies - III Practical~~ (Completion: true)
~~BCA 504 (C) - Machine Learning Practical~~ (Completion: true)
Add a new task
Add Task

Practical 5 a):- To use ng-controller directive to display tasks in the UI.

```
<!DOCTYPE html>
<html>
<head>
        <title>
                ng-controller Directive
        </title>
        <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.2/angular.min.js">
        </script>
</head>
<body ng-app="app" style="text-align:center">
        <h1 style="color:green">
                ng-controller Directive Example
        </h1>
                <br>
        <div ng-controller="hvp">
                Name:
                        <input class="form-control"
                                type="text"
                                ng-model="name">
                <br><br>
                You entered:
                        <b> <span>{{name}}</span> </b>
        </div>
        <script>
          var app = angular.module('app', []);
          app.controller('hvp', function ($scope) {
             $scope.name = "Purva Harshal Patil";
          });
        </script>
</body>
</html>
```

Output:

Practical 5 b):- To use ng-bind directive to display tasks in the UI.

```
<!DOCTYPE html>
<html>
<head>
<title> ng-bind Directive </title>
<script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
</script>
</head>
<body ng-app="hvp"
style="text-align:center">
<h1 style="color:green"> ng-bind Directive Example </h1>
<div ng-controller="app">
        num1:
                <input type="number"
                        ng-model="num1"
                        ng-change="product()" />
        <br><br>
        num2:
                <input type="number"
                        ng-model="num2"
                        ng-change="product()" />
        <br><br>
        <b>Product:</b>
                <span ng-bind="result"></span>
</div>
<script>
   var app = angular.module("hvp", []);
   app.controller('app', ['$scope', function ($app) {
      $app.num1 = 1;
      $app.num2 = 1;
      $app.product = function () {
         $app.result = ($app.num1 * $app.num2);
      }
   } ]);
</script>
</body>
</html>
```

Output:

Practical 5 c):- To use ng-if directive to display tasks in the UI.

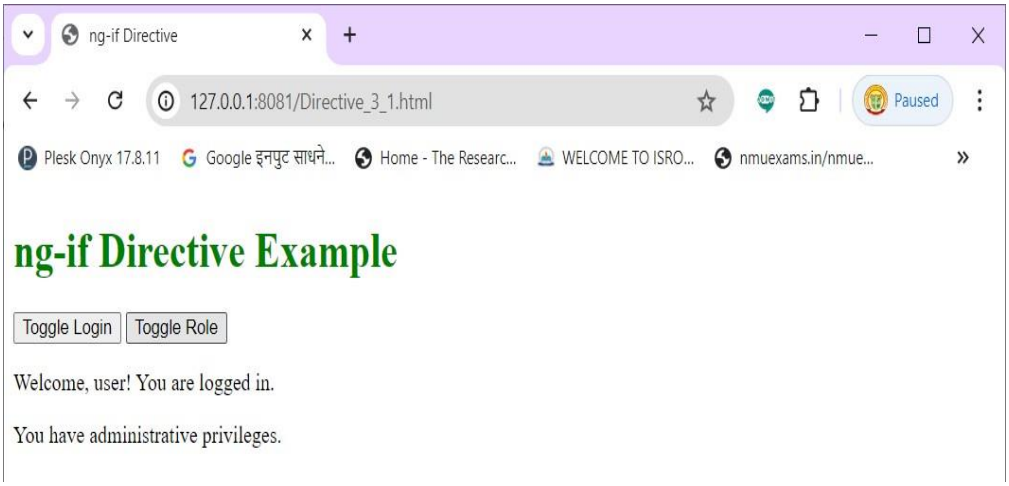
```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
<title> ng-if Directive </title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></
script>
  <script>
     var app = angular.module('myApp', []);
     app.controller('myCtrl', function ($scope) {
        $scope.showElement = true;
     });
  </script>
</head>
<body ng-controller="myCtrl">
<h1 style="color:green">
                    ng-if Directive Example
         </h1>
  <button ng-click="showElement = !showElement">Toggle
Element</button>
  <p ng-if="showElement">This paragraph is conditionally displayed.</p>
</body>
</html>
```

Output:

Practical 5 d):- To use ng-if directive to display tasks in the UI.

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
<title> ng-if Directive </title>
 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></
script>
 <script>
    var app = angular.module('myApp', []);
    app.controller('myCtrl', function ($scope) {
       $scope.user = {
          loggedIn: false,
          role: 'guest'
       };
    });
 </script>
</head>
<body ng-controller="myCtrl">
<h1 style="color:green">
                ng-if Directive Example
       </h1>
 <button ng-click="user.loggedIn = !user.loggedIn">Toggle Login</button>
 <button ng-click="user.role = (user.role === 'guest' ? 'admin' :
'guest')">Toggle Role</button>

 <p ng-if="user.loggedIn">Welcome, user! You are logged in.</p>
 <p ng-if="!user.loggedIn">Please log in.</p>
 <p ng-if="user.loggedIn && user.role === 'admin'">You have
administrative privileges.</p>
</body>
</html>
```

Output:



Practical 5 e):- To use ng-switch directive to display tasks in the UI.

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
<title> ng-switch Directive </title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></sc
ript>
  <script>
     var app = angular.module('myApp', []);
     app.controller('myCtrl', function ($scope) {
        $scope.selection = 'option1';
     });
  </script>
</head>
<body ng-controller="myCtrl">
<h1 style="color:green">
          ng-switch Directive Example
</h1>
    <b>Select your favourite color:</b>
  <select ng-model="selection">
    <option value="red">Red</option>
    <option value="blue">Blue</option>
    <option value="green">Green</option>
  </select>

  <div ng-switch="selection">
    <div ng-switch-when="red">You selected Red Color</div>
    <div ng-switch-when="blue">You selected Blue Color</div>
    <div ng-switch-when="green">You selected Green Color</div>
    <div ng-switch-default>Please select an color</div>
  </div>
</body>
</html>
```

Output:

Practical 5 f):- To use ng-repeat directive to display tasks in the UI.

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
<title> ng-repeat Directive </title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></sc
ript>
  <script>
     var app = angular.module('myApp', []);
     app.controller('myCtrl', function ($scope) {
        $scope.names = ["Purva", "Bhakti", "Harshal"];
     });
  </script>
</head>
<body ng-controller="myCtrl">
<h1 style="color:green">
          ng-repeat Directive Example
</h1>
  <ul>
    <li ng-repeat="name in names">{{ name }}</li>
  </ul>
</body>
</html>
```

Output:

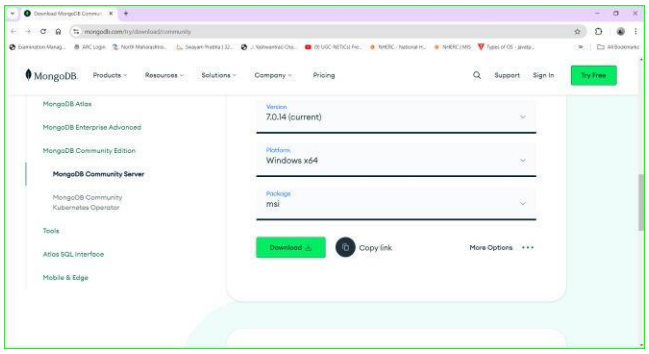
Practical 5 g):- To use ng-repeat directive using Object to display tasks in the UI.

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
<title> ng-repeat Directive </title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></sc
ript>
  <script>
     var app = angular.module('myApp', []);
     app.controller('myCtrl', function ($scope) {
        $scope.people = [
       { name: "Purva", age: 10 },
       { name: "Bhakti", age: 05 },
       { name: "Harshal", age: 38 }
     ];
     });
  </script>
</head>
<body ng-controller="myCtrl">
<h1 style="color:green">
          ng-repeat Directive Example using Object
</h1>
  <ul>
  <ul>
    <li ng-repeat="person in people">
     {{ person.name }} - {{ person.age }} years old
    </li>
  </ul>
</body>
</html>
```

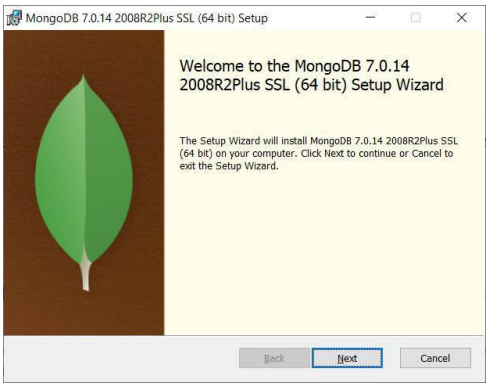
Output:

Practical 6:- To create database and structure in MongoDB using Mongosh.

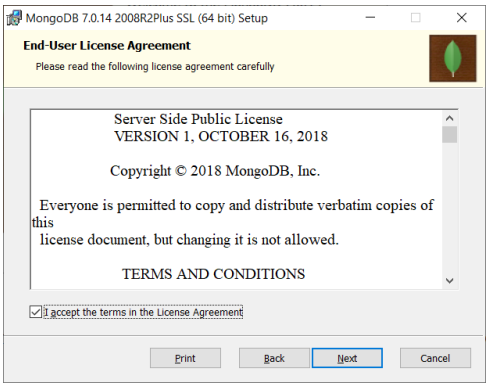
Step – 1 Go to the MongoDB Download Center to download the MongoDB Community Server:



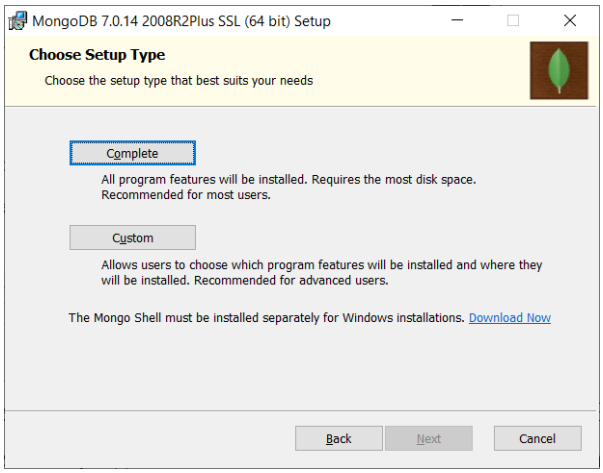
Step – 2 When the download is complete open the msi file and click the next button in the startup screen.



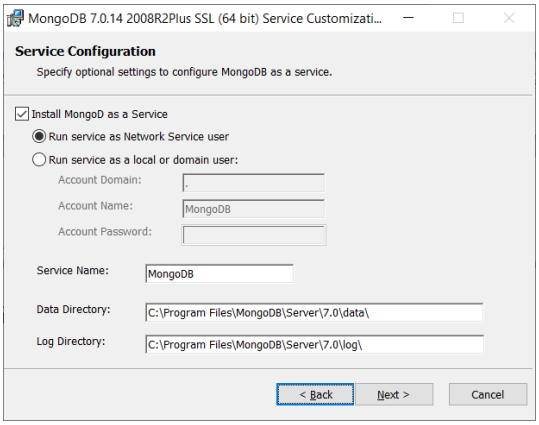
Step – 3 Now accept the End-User License Agreement and click the next button:



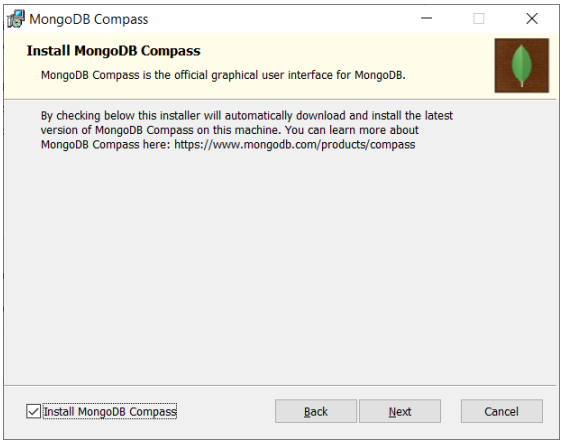
Step – 4 Now select the complete option to install all the program features. Here, if you can want to install only selected program features and want to select the location of the installation, then use the Custom option:



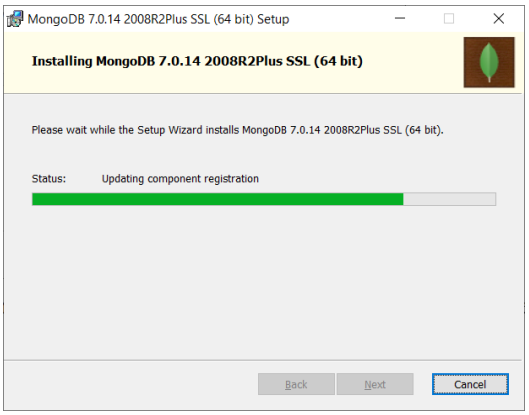
Step – 5 Select "Run service as Network Service user" and copy the path of the data directory.(C:\Program Files\MongoDB\Server\7.0\data\) Click Next:



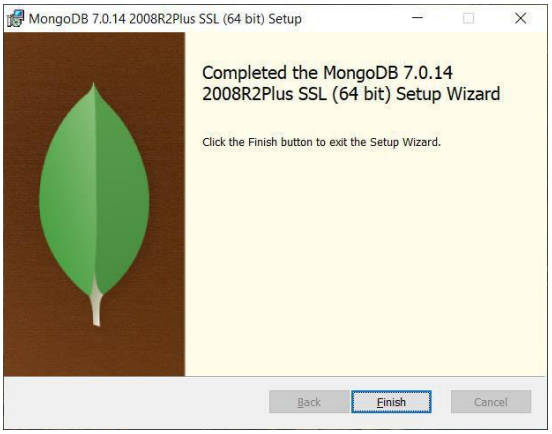
Step – 6 Click the Install button to start the MongoDB installation process



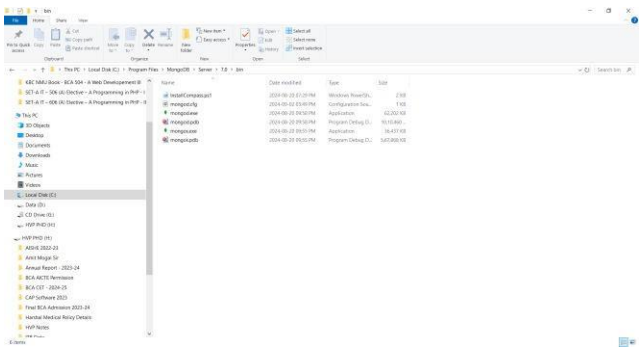
Step – 7 After clicking on the install button installation of MongoDB begins



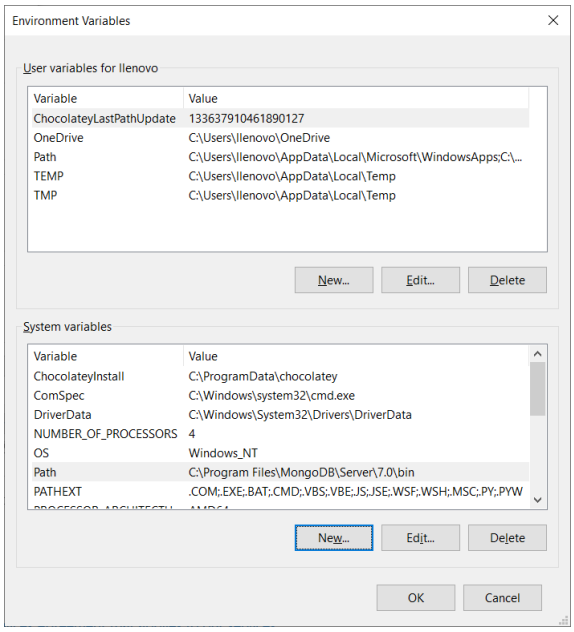
Step – 8 Now click the Finish button to complete the MongoDB installation process



Step – 9 Now we go to the location where MongoDB installed in step 5 in your system and copy the bin path:



Step – 10 Now, to create an environment variable open system properties >> Environment Variable >> System variable >> path >> Edit Environment variable and paste the copied link to your environment system and click Ok



Step – 11 After setting the environment variable, we will run the MongoDB server, i.e. mongod. So, open the command prompt and run the following command

mongod

Now, this time the MongoDB server (i.e., mongod) will run successfully.

Run mongo Shell

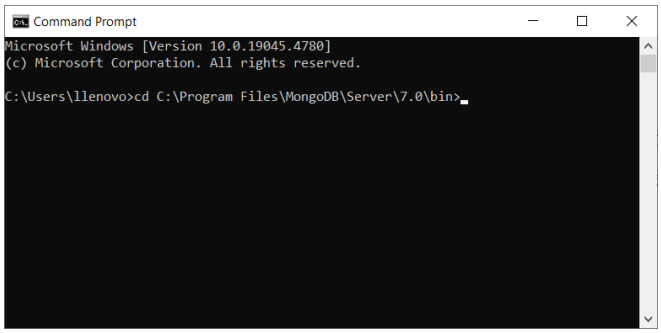
Step – 12 Check if mongosh.exe is present in the MongoDB installation directory. Look inside C:\Program Files\MongoDB\Server\7.0\bin for mongosh.exe.

Download MongoDB Shell:

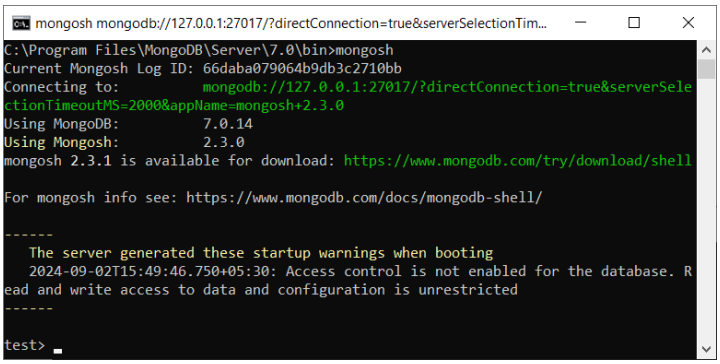
If mongosh is missing, download and install it from the official MongoDB website.

Run MongoDB

Step-1 To run MongoDb open command prompt and go to C:\Program Files\MongoDB\Server\7.0\bin>



Step-2 After that go to mongosh shell



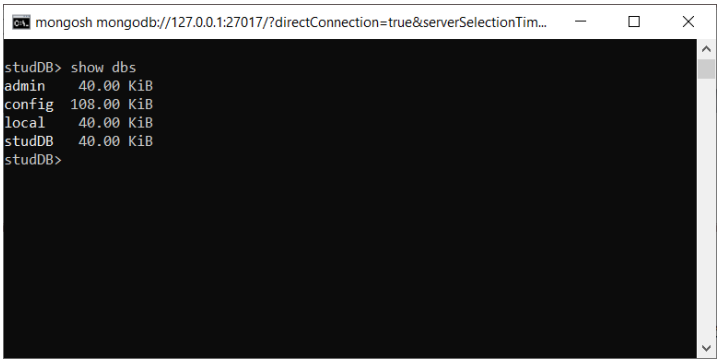
Step-3 Now you can create a new database, collections, and documents in your mongosh shell.

- The **use <<Database_name>>** command to create a new database in the system if it does not exist, if the database exists it uses that database:

use studDB

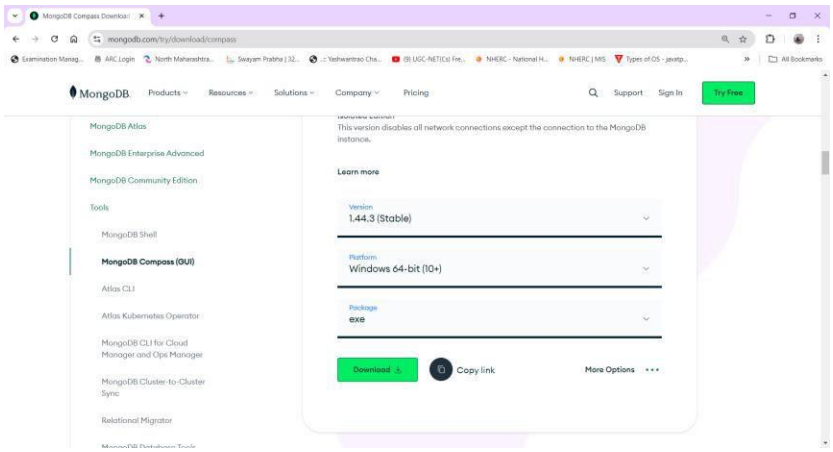
Now your database is ready of name **studDB**.

- To display all database use **show dbs** command

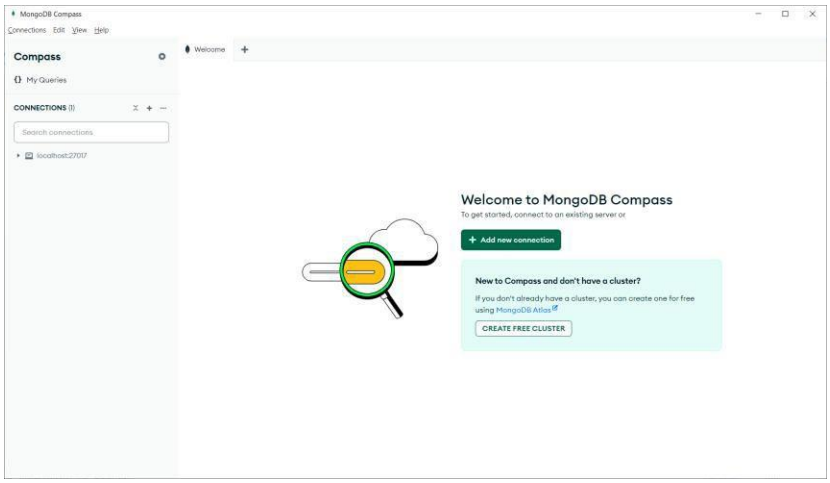


Practical 6:- To create database and structure in MongoDB using Mongo Compass GUI.

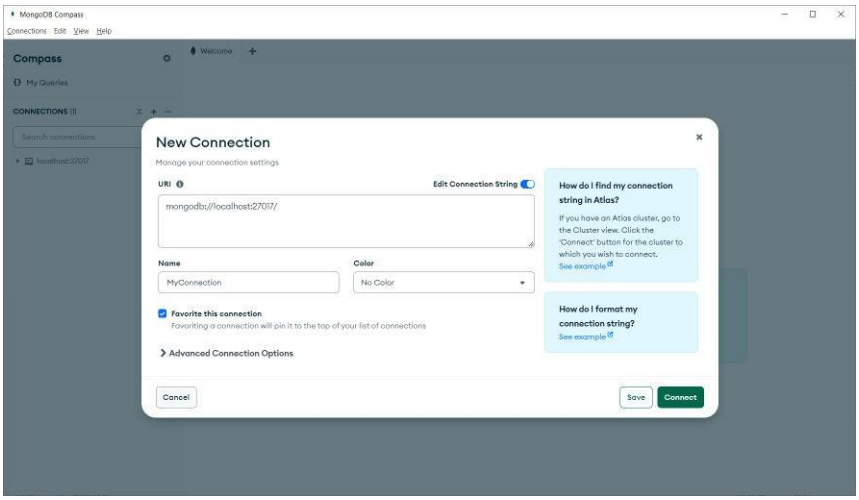
Step – 1 Go to the MongoDB Download Center to download the MongoDB Compass:



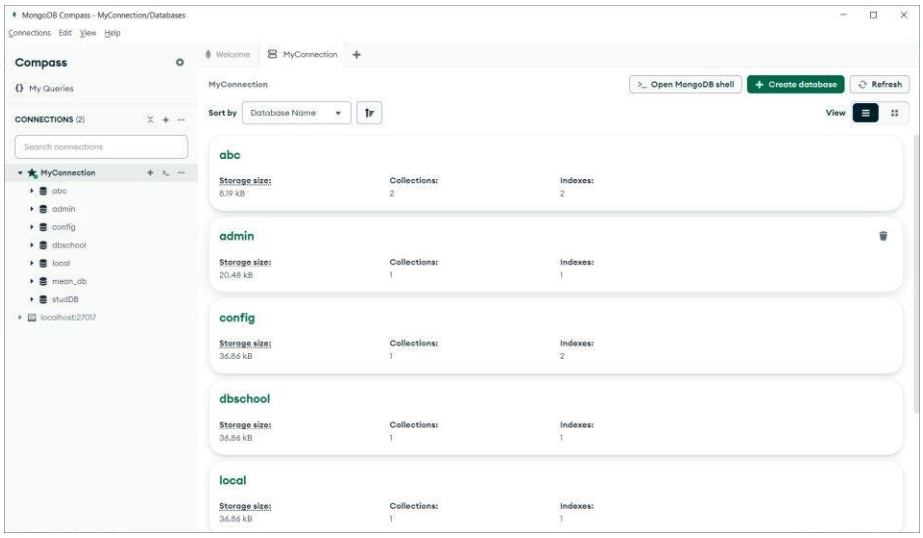
Step – 2 After installation of MongoDB Compass Open It.



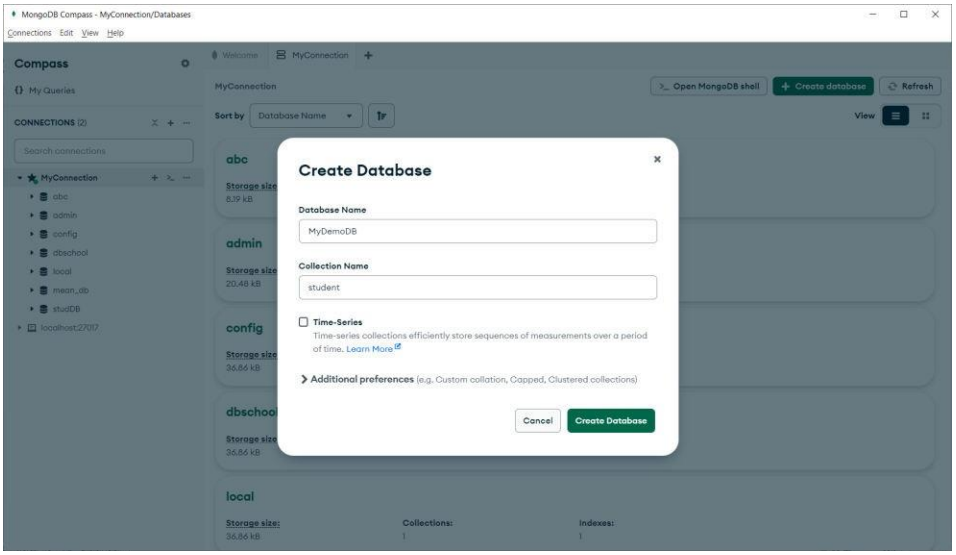
Step – 3 After Open MongoDB Compass Go to -> Add New Connection -> Enter Connection Name like MyConnection -> click Connect

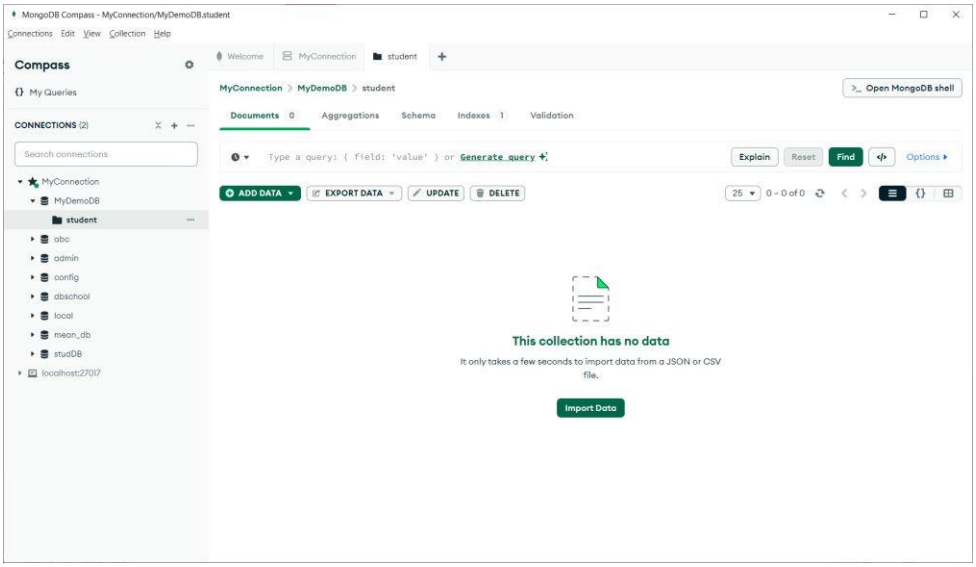


Step – 4 After Click Connect.



Step – 5 After that Click Create Database and Enter Database Name and Collection Name if you want then Click Create Database Button.

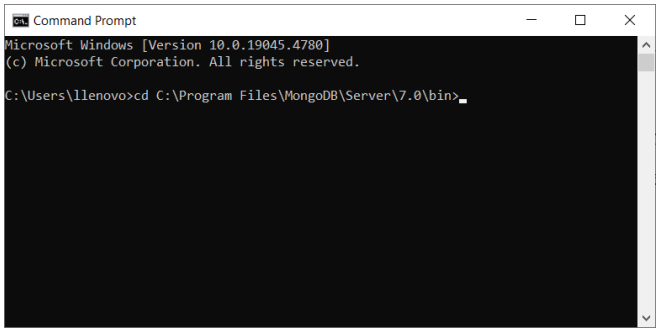




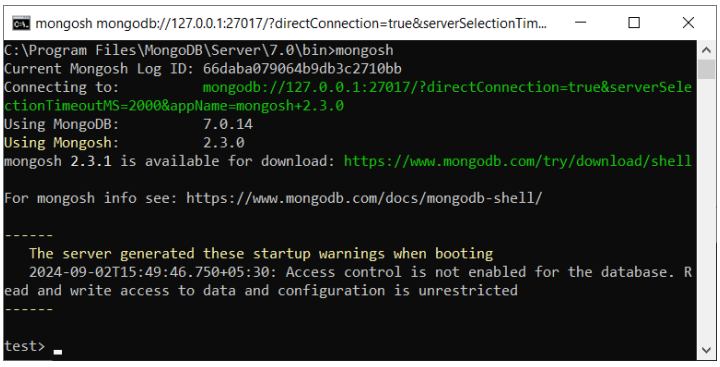
Practical 7:- To create collection and insert records in collections.

Run MongoDB

Step-1 To run MongoDb open command prompt and go to C:\Program Files\MongoDB\Server\7.0\bin>



Step-2 After that go to mongosh shell

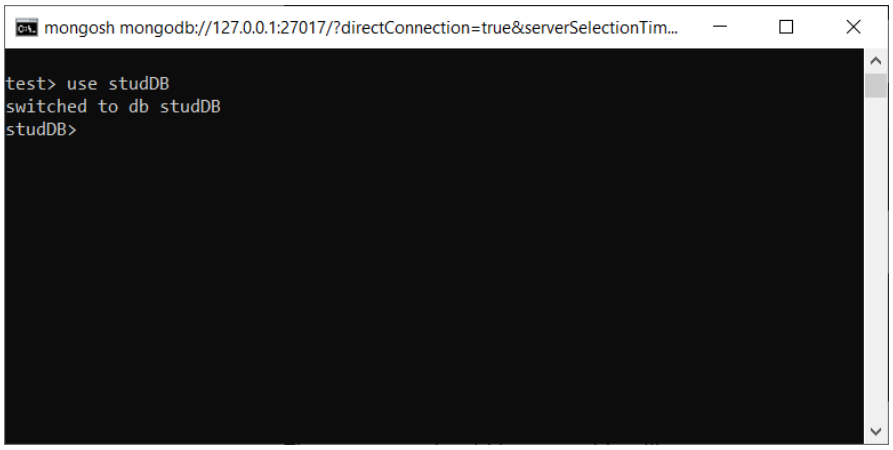


Step-3 Now you can create a new database, collections, and documents in your mongosh shell.

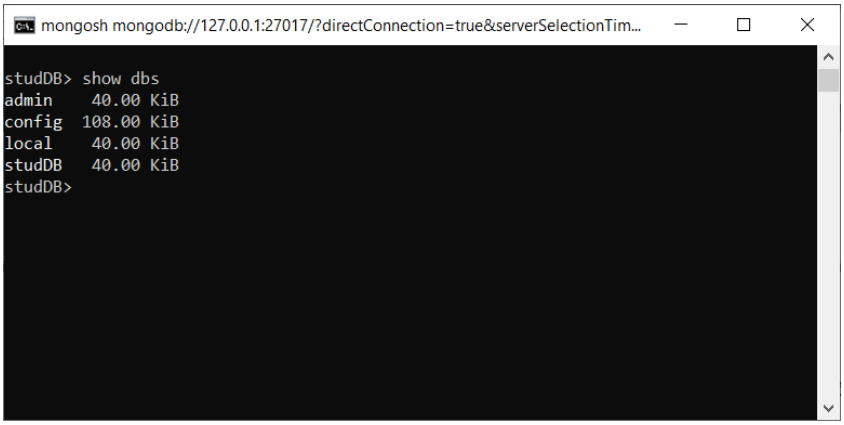
- The **use <<Database_name>>** command to create a new database in the system if it does not exist, if the database exists it uses that database:

use studDB

Now your database is ready of name **studDB**.

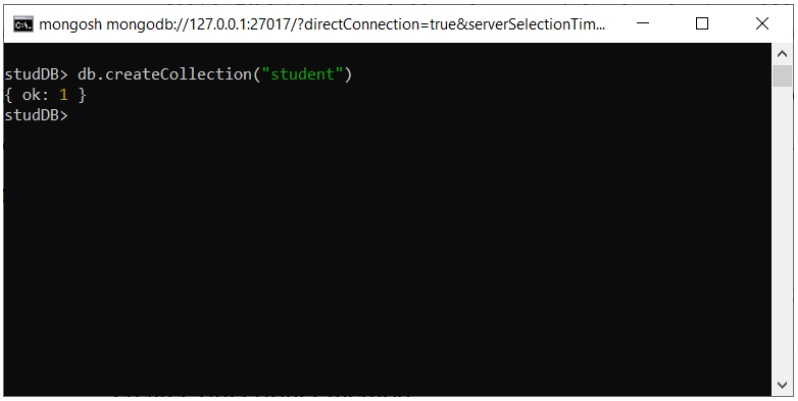


- To display all database use **show dbs** command

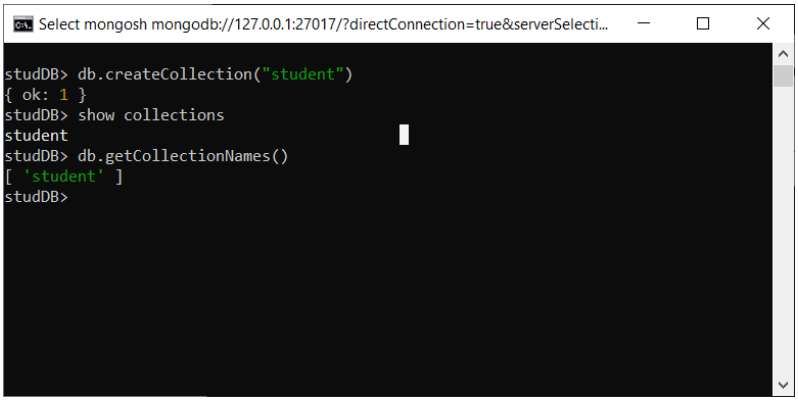


- Collections are created automatically when you insert the first document. Alternatively, you can create a collection explicitly using the createCollection() method.

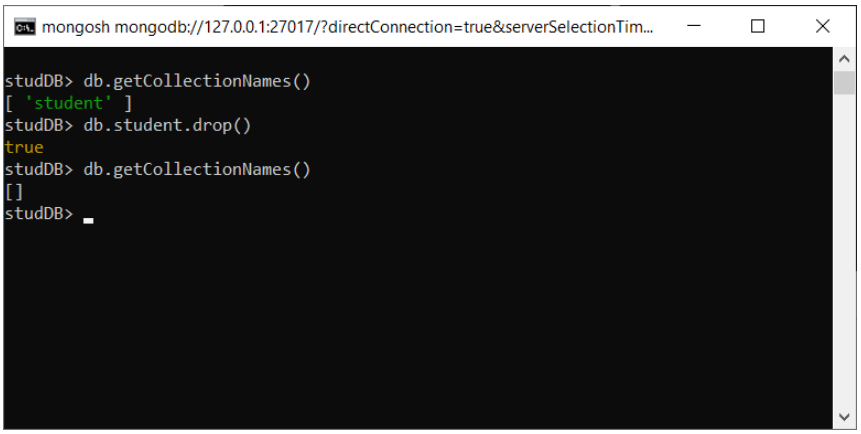
db.createCollection("myCollection")



- Display all collection in Database used **show collections command or db.getCollectionNames() method**.



- To delete collection use : Use the **drop() method** on the collection



- The **insertOne()** method inserts the document in the student collection:

db.student.insertOne({name: "Purva", age: 08 **})**

- **Insert Multiple Records:** Use the insertMany() method to insert multiple documents at once. Here's an example:

```
db.student.insertMany([
{ name: "Harshal", age: 38 },
{ name: "Bhakti", age: 08 },
{ name: "Sujata", age: 35 }
])
```

This will insert three documents into the student collection.

Practical 8:- To demonstrate insert, update, delete, select operations in MongoDB.

How to insert, update, display and remove documents in MongoDB:

- **Inserting a Document:** To insert a document into a collection, use the insertOne() or insertMany() methods.

  Example - 1

```
db. student.insertOne({
   name: "Purva Harshal Patil",
   age: 08,
   city: "Talvel"
});
```

  Example - 2

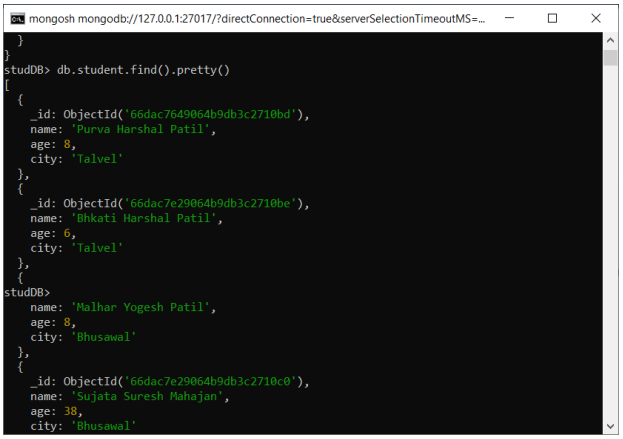
```
db. student.insertMany([
   { name: "Bhkati Harshal Patil", age: 06, city: "Talvel" },
   { name: "Malhar Yogesh Patil", age: 08, city: "Bhusawal" },
   { name: "Sujata Suresh Mahajan", age: 38, city: "Bhusawal" }
]);
```

- **Retrieving Documents:** Show the Records: Use the find method to retrieve and display records from the student collection:

  db.student.find().pretty()

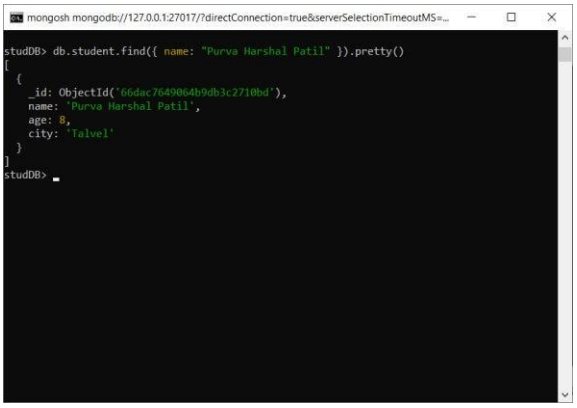
  - find() retrieves all documents from the student collection.

- pretty() formats the output to be more readable



If you want to find a specific record, you can pass a query to the find method. For example, to find the record where name equals Purva Harshal Patil, you can use:

db.student.find({ name: "Purva Harshal Patil" }).pretty()



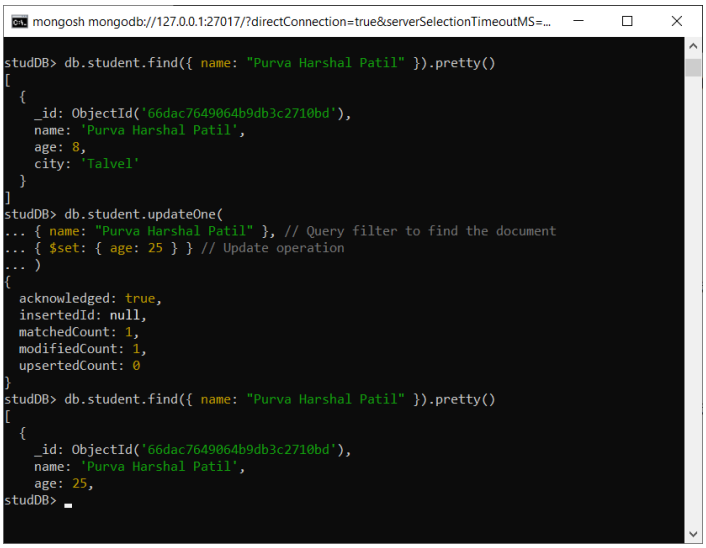
- **Updating Documents:** Use the updateOne() or updateMany() methods to update documents.

Example – 1

Use updateOne with a query filter and update operations:

```
db.student.updateOne(
{ name: "Purva Harshal Patil" }, // Query filter to find the document
{ $set: { age: 25 } } // Update operation
)
```

This will update the first document where the name field is "**Purva Harshal Patil**", setting the age field to 25.



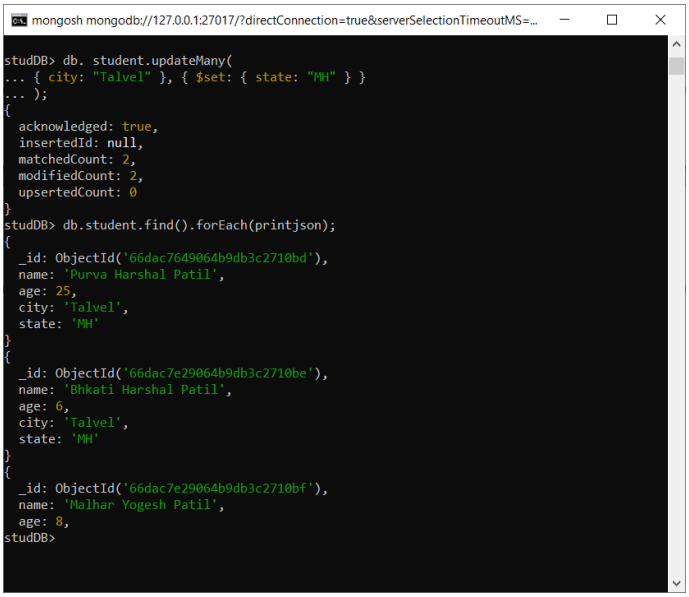
Example – 2

Use updateMany with a query filter and update operations:

db. student.updateMany(

{ city: "Talvel" }, { $set: { state: "MH" } }

);

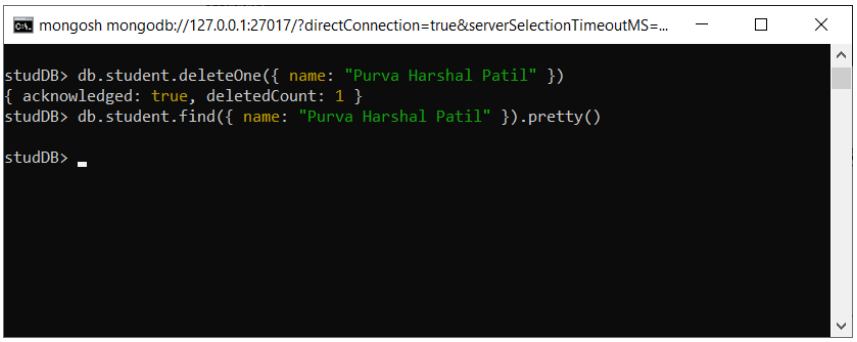


- **Deleting Documents:** To delete documents, use the deleteOne() or deleteMany() methods.
  - **Delete a Single Document**

    If you want to delete a specific document, use deleteOne with a query filter:

    db.student.deleteOne({ name: "Purva Harshal Patil" })

    This will delete the first document that matches the filter.



- **Delete Multiple Documents**

  If you want to delete multiple documents that match a certain condition, use deleteMany:

  db.student.deleteMany({ age: { $lt: 18 } })

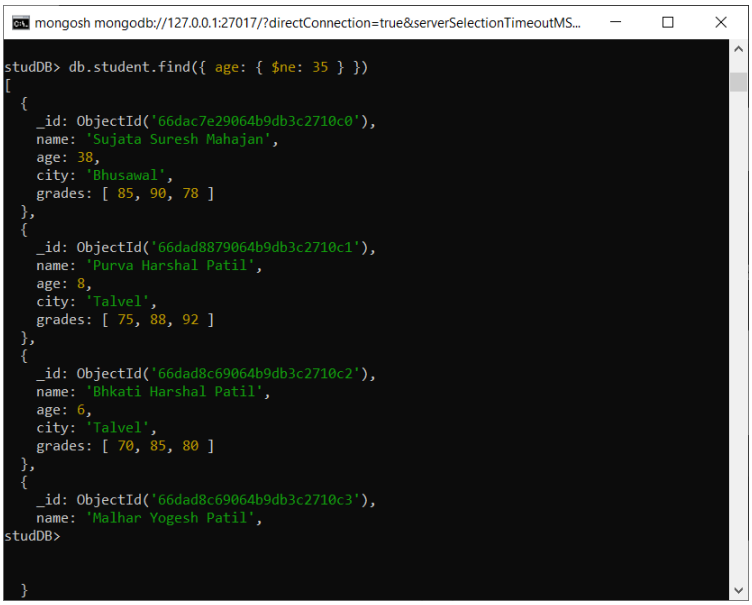
  This example deletes all documents where the age field is less than 18.

Example – 1 Use Comparison $eq - Equal Query Operators:

db.student.find({ age: { $eq: 35 } })

Example – 2 Use Comparison $ne - Not Equal Query Operators:

db.student.find({ age: { $ne: 35 } })



Example – 3 Use Comparison $gt - Greater Than Query Operators:

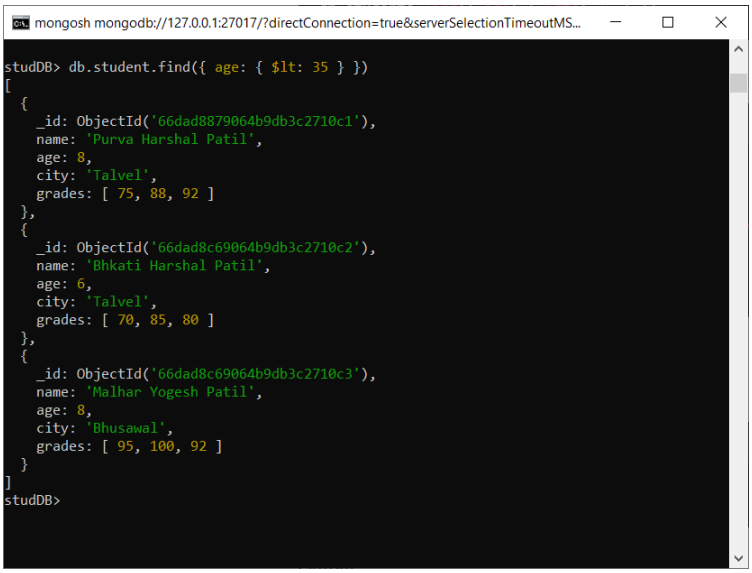
db.student.find({ age: { $gt: 35 } })

Example – 4 Use Comparison $gte - Greater Than or Equal Query Operators:

db.student.find({ age: { $gte: 35 } })

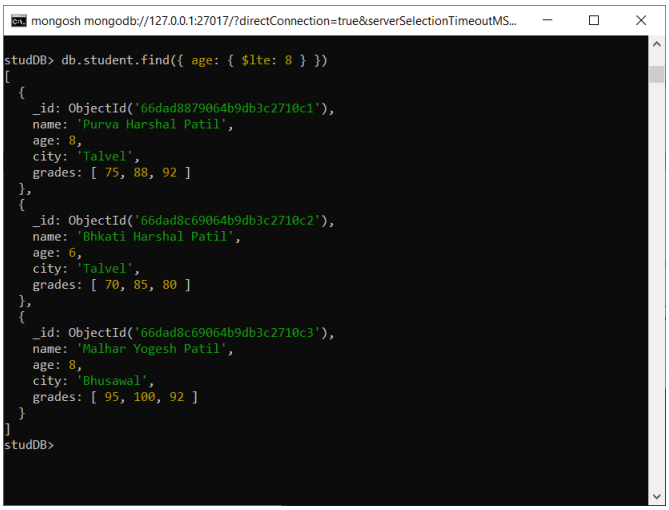
Example – 5 Use Comparison $lt - Less Than Query Operators:

db.student.find({ age: { $lt: 35 } })



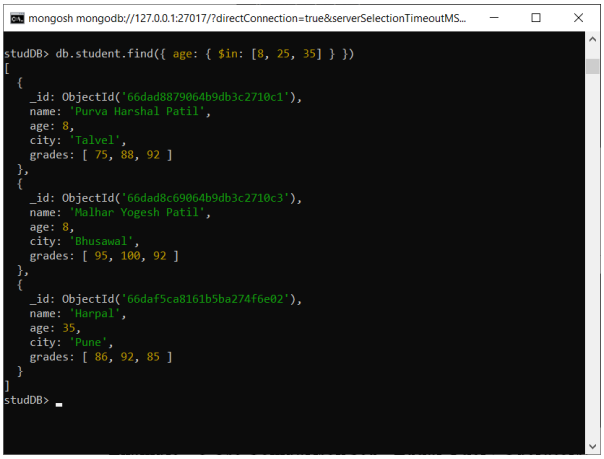
Example – 6 Use Comparison $lte - Less Than or Equal Query Operators:

db.student.find({ age: { $lte: 8 } })



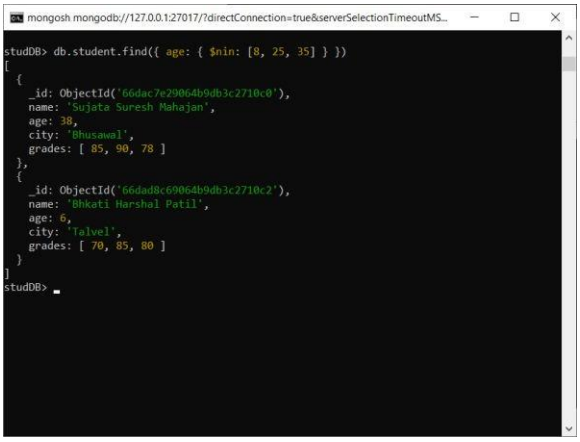
Example – 7 Use Comparison $in – In Query Operators:

db.student.find({ age: { $in: [8, 25, 35] } })



Example – 8 Use Comparison $nin - Not In Query Operators:

db.student.find({ age: { $nin: [8, 25, 35] } })

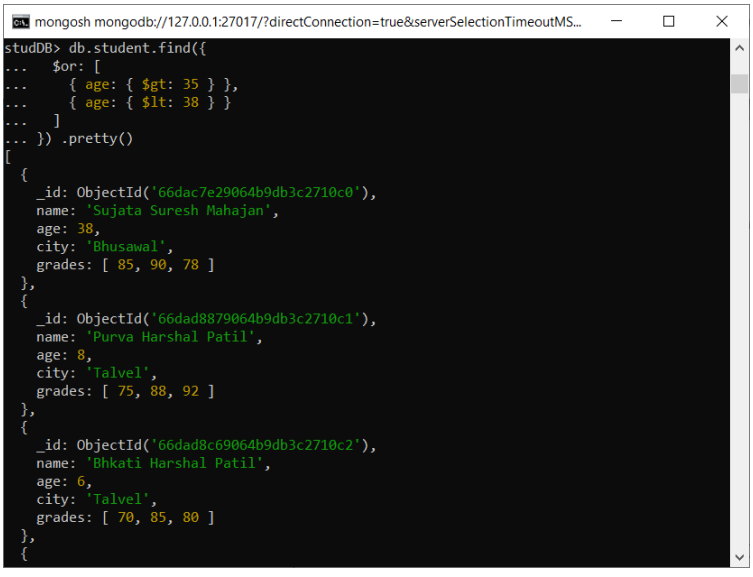


Example – 1 Use $and - Logical AND Query Operators:

```
db.student.find({
  $and: [
    { age: { $gt: 6 } },
    { age: { $lt: 35 } }
  ]
}) .pretty()
```

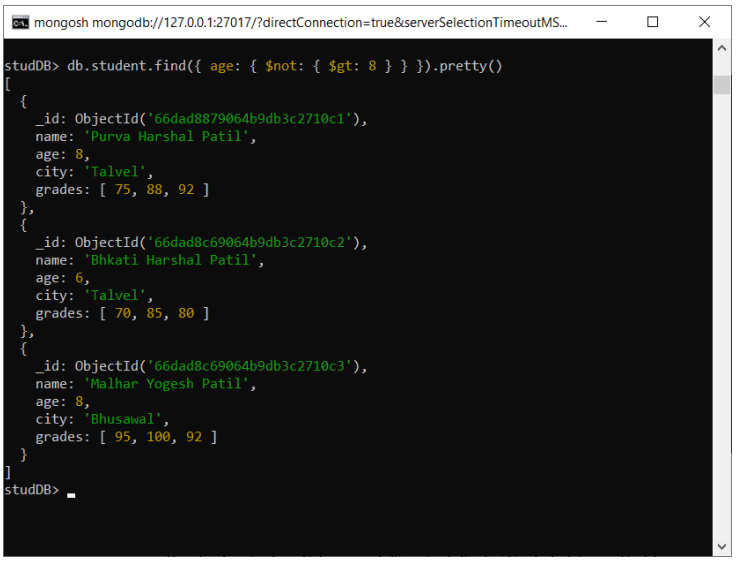
Example – 2 Use $or - Logical OR Query Operators:

```
db.student.find({
  $or: [
    { age: { $gt: 35 } },
    { age: { $lt: 38 } }
  ]
}).pretty()
```



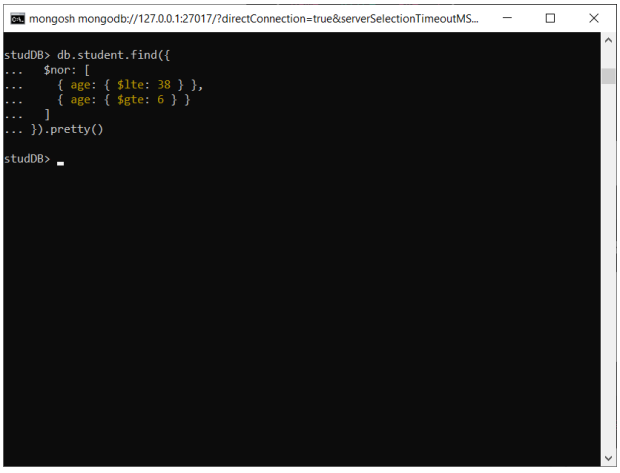
Example – 3 Use $not - Logical NOT Query Operators:

```
db.student.find({ age: { $not: { $gt: 8 } } }).pretty()
```



Example – 4 Use $nor - Logical NOR Query Operators:

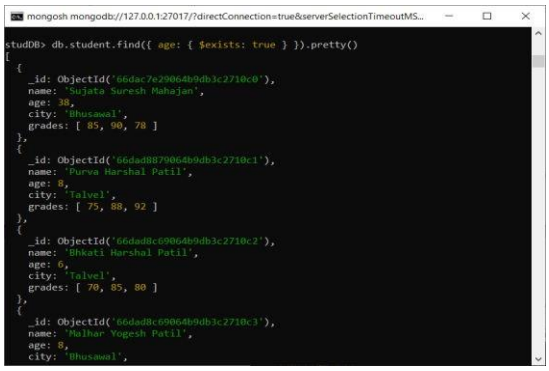
```
db.student.find({
  $nor: [
    { age: { $lte: 38 } },
    { age: { $gte: 6 } }
  ]
}).pretty()
```



This will return no documents because all age values will satisfy either age <= 38 or age >= 6.

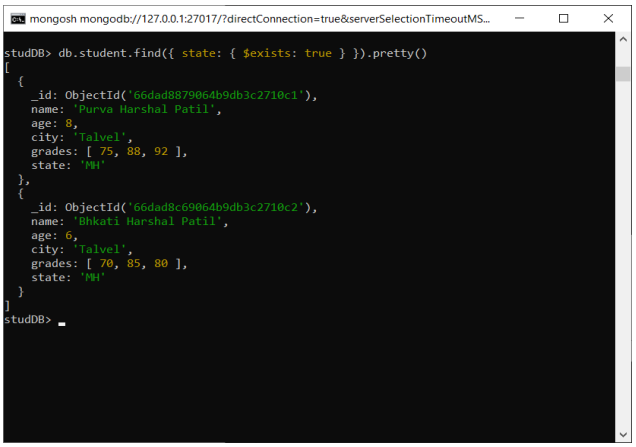
Example – 1 Use $exists Query Operators:

db.student.find({ age: { $exists: true } }).pretty()



Example – 2 Use $exists Query Operators:

db.student.find({ state: { $exists: true } }).pretty()

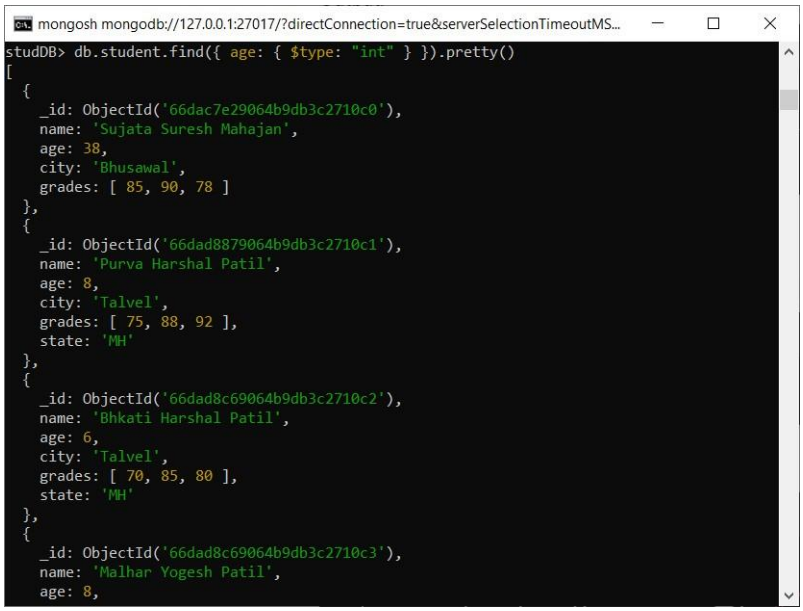


Example – 3 Use $exists Query Operators:

db.student.find({ state: { $exists: false } }).pretty()

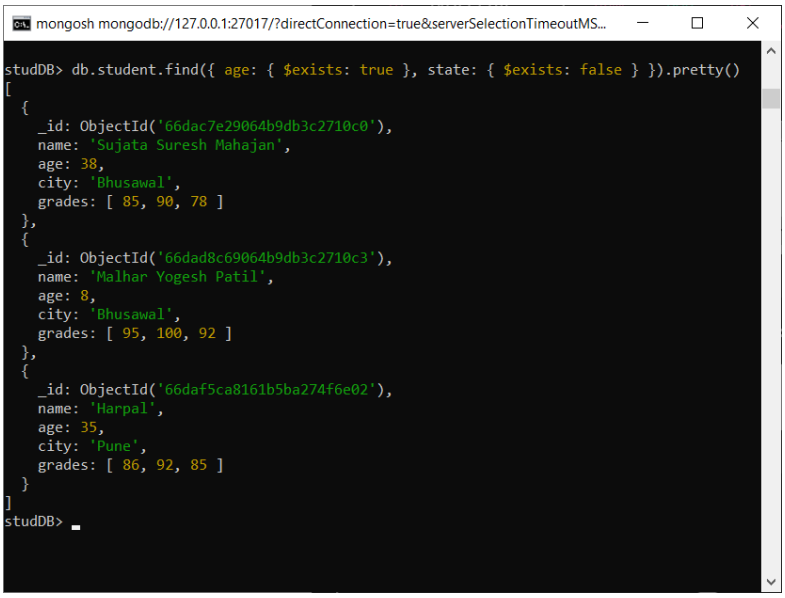
Example – 4 Use $type Query Operators:

db.student.find({ age: { $type: "int" } }).pretty()



Example – 5 Find students who have an age field but do not have a state field.:

db.student.find({ age: { $exists: true }, state: { $exists: false } }).pretty()



Example – 1 Use $expr Query Operators:

Find students whose age are greater than or equal to 35.

```
db.student.find({ $expr: { $gte: ["$age", 35] } }).pretty()
```

Example – 2 Use $regex Query Operators:

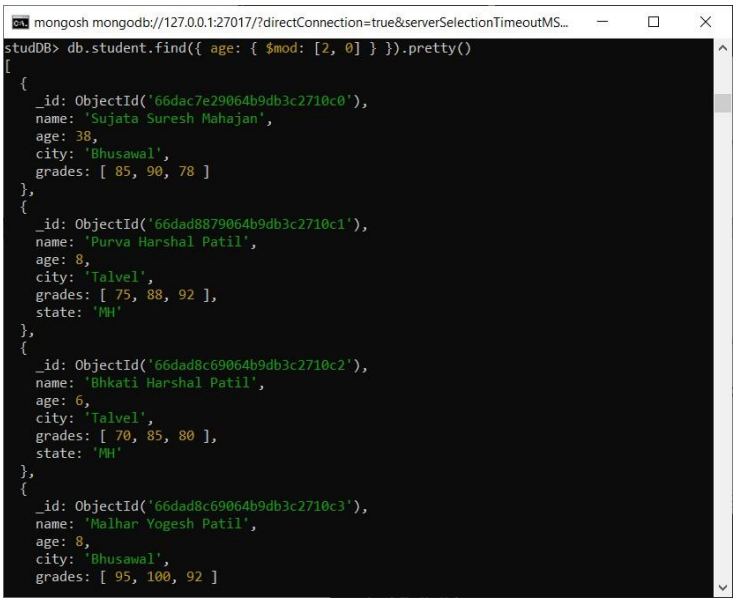
Find students whose names start with the letter "H".

```
db.student.find({ name: { $regex: /^H/ } }).pretty()
```

Example – 3 Use $mod (Module Operation) Query Operators:

Find students whose age is an even number.

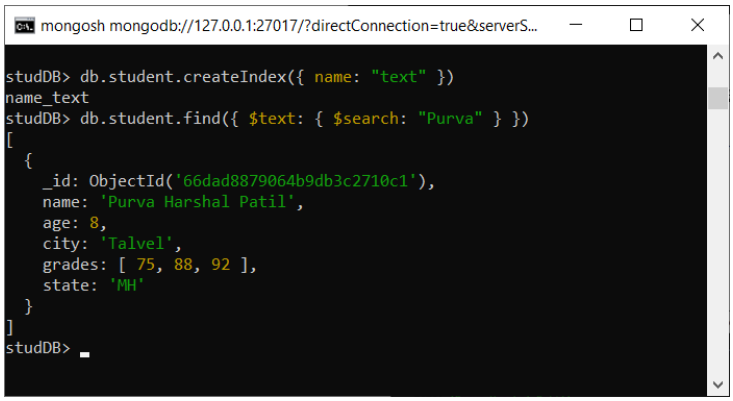
```
db.student.find({ age: { $mod: [2, 0] } }).pretty()
```



Example – 4 Use $text (Perform Text Search) Query Operators:

Create a text index on the name field and find students whose names contain the word "Purva".

```
db.student.createIndex({ name: "text" })
db.student.find({ $text: { $search: "Purva" } })
```



Example – 5 Use $where Query Operators:

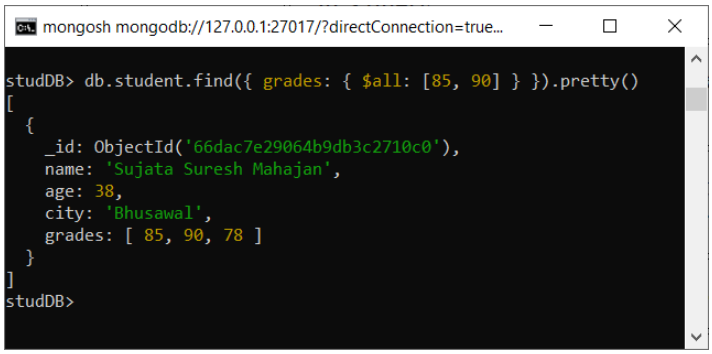
Find students whose age is greater than 25.

```
db.student.find({ $where: function() { return this.age > 25} }).pretty()
```

Example – 1 Use $all Query Operators:

Find students who have grades of 85 and 90.

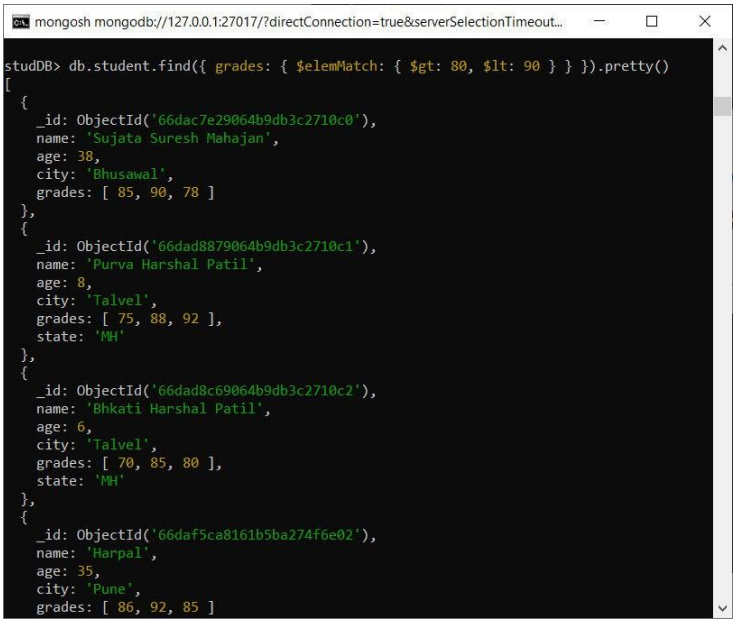
```
db.student.find({ grades: { $all: [85, 90] } }).pretty()
```



Example – 2 Use $elemMatch Query Operators:

Find students who have a grade greater than 80 and less than 90.

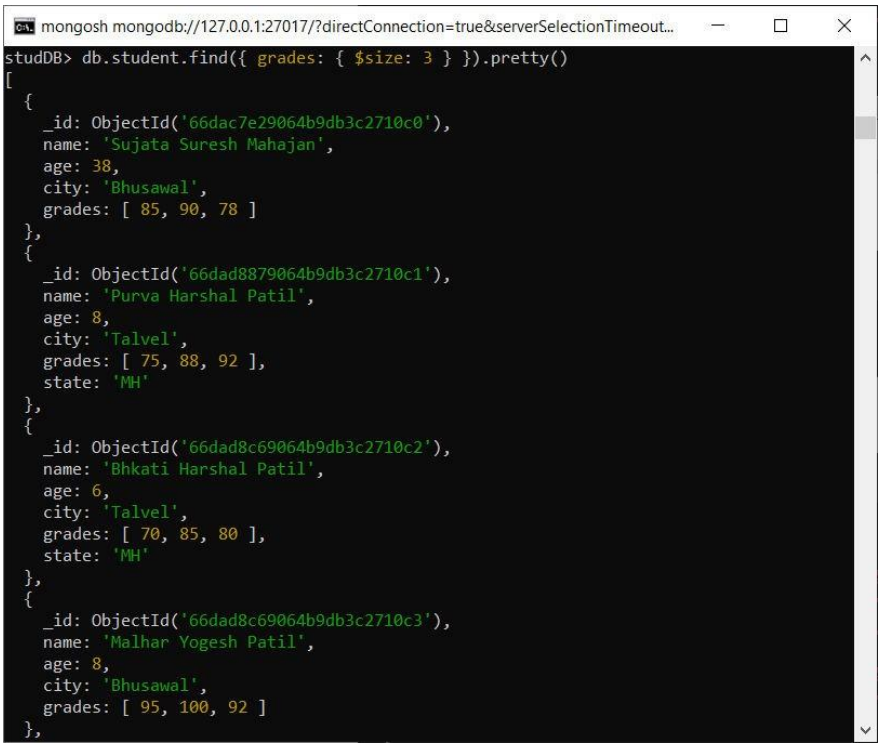
```
db.student.find({ grades: { $elemMatch: { $gt: 80, $lt: 90 } } }).pretty()
```



Example – 3 Use $size Query Operators:

Find students who have exactly 3 grades.

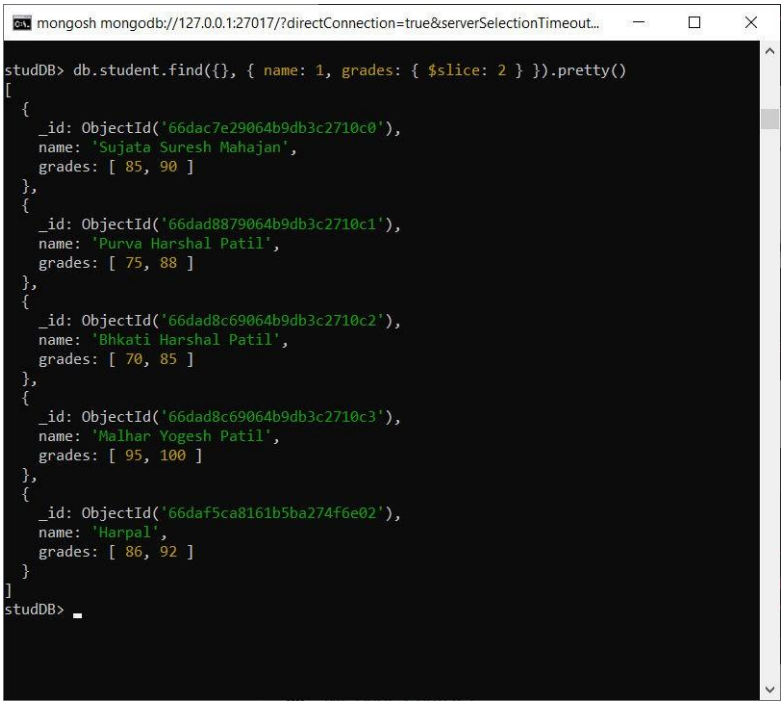
```
db.student.find({ grades: { $size: 3 } }).pretty()
```



Example – 4 Use $slice Query Operators:

Find students and return only the first 2 grades from the grades array.

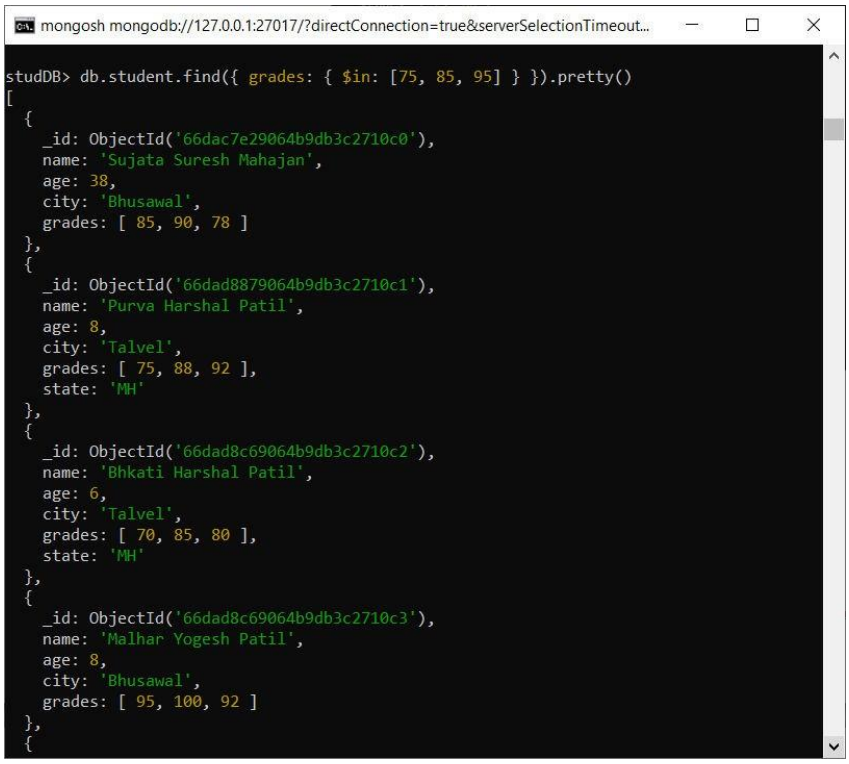
db.student.find({}, { name: 1, grades: { $slice: 2 } }).pretty()



Example – 5 Use $in Query Operators:

Find students who have any of the grades 75, 85, or 95.

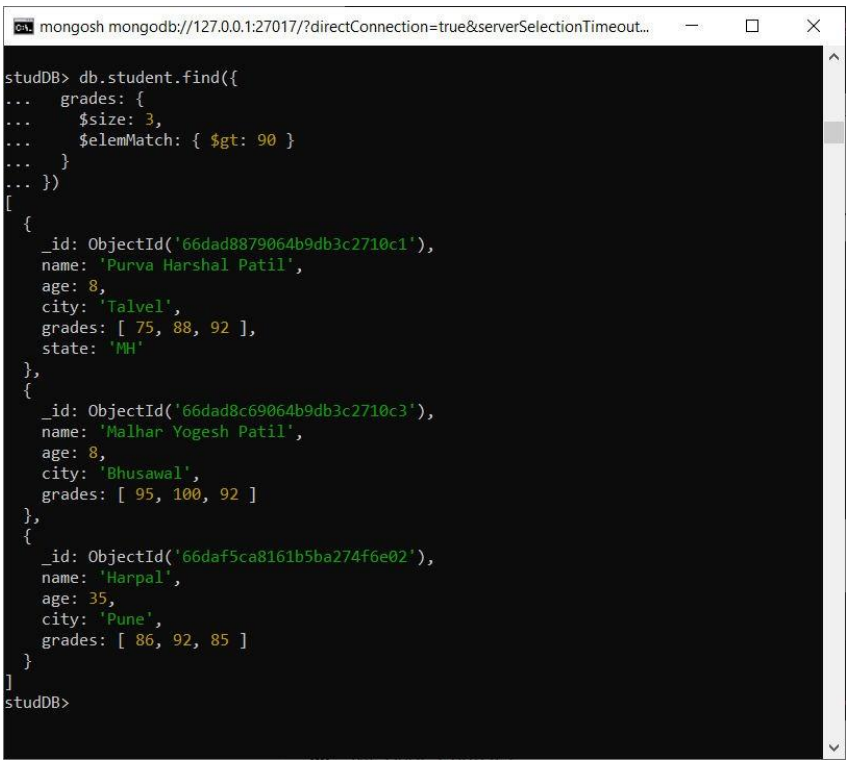
db.student.find({ grades: { $in: [75, 85, 95] } }).pretty()



Example – 6 Use Array Query Operators:

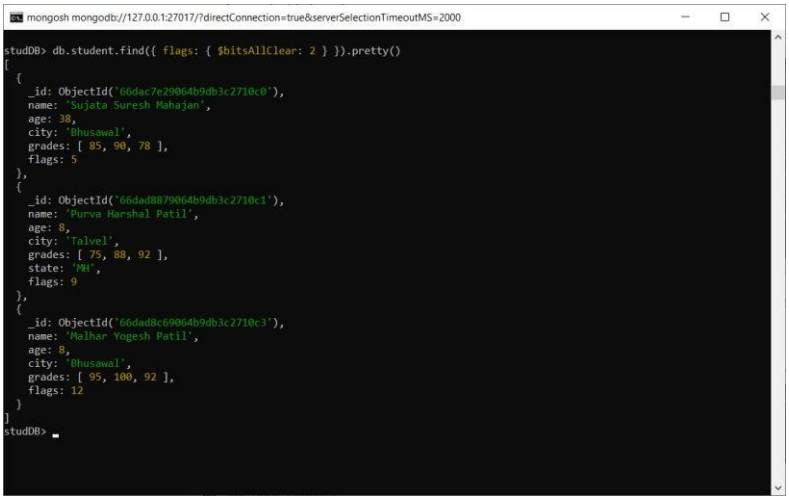
Find students who have exactly 3 grades, and at least one grade is greater than 90.

db.student.find({ grades: { $size: 3, $elemMatch: { $gt: 90 }}})



Example – 1 Use $ bitsAllClear Query Operators:

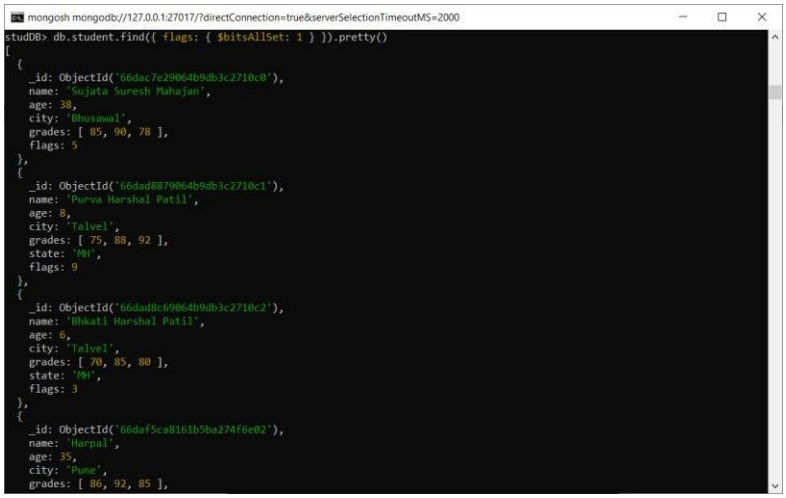
db.student.find({ flags: { $bitsAllClear: 2 } }).pretty()



This query matches students whose flags field has the second bit clear (0).

Example – 2 Use $ bitsAllSet Query Operators:

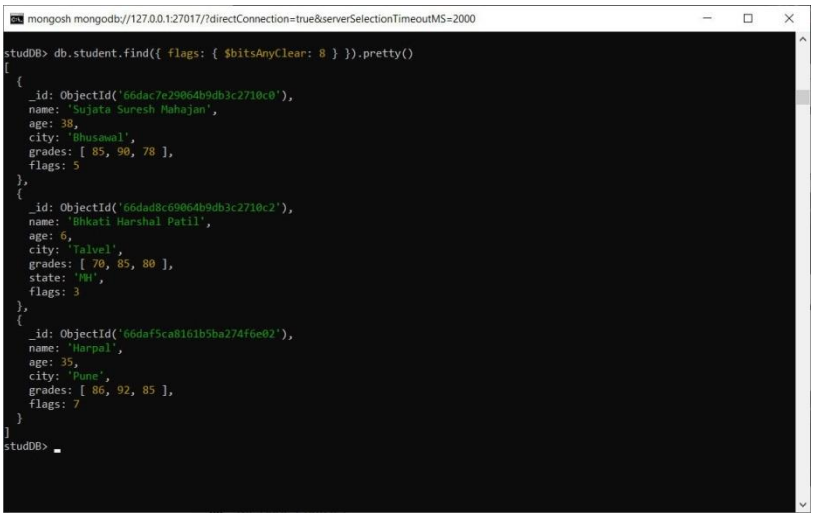
db.student.find({ flags: { $bitsAllSet: 1 } }).pretty()



This query matches students whose flags field has the first bit set (1).

Example – 3 Use $ bitsAnyClear Query Operators:

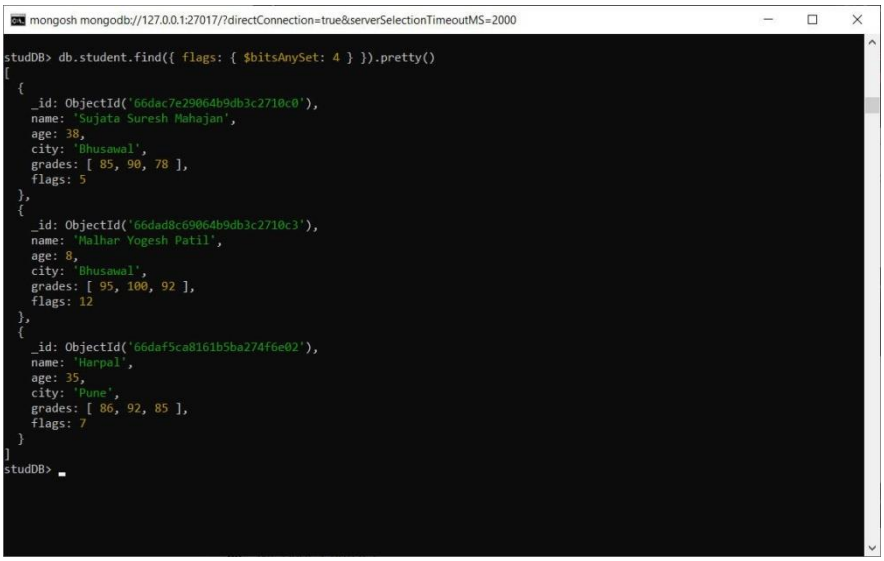
db.student.find({ flags: { $bitsAnyClear: 8 } }).pretty()



This query matches students whose flags field has at least one bit clear (0) in the fourth bit position.

Example – 4 Use $ bitsAnySet Query Operators:

db.student.find({ flags: { $bitsAnySet: 4 } }).pretty()



This query matches students whose flags field has at least one bit set (1) in the third bit position.

Example – 5 Use Bitwise Query Operators:

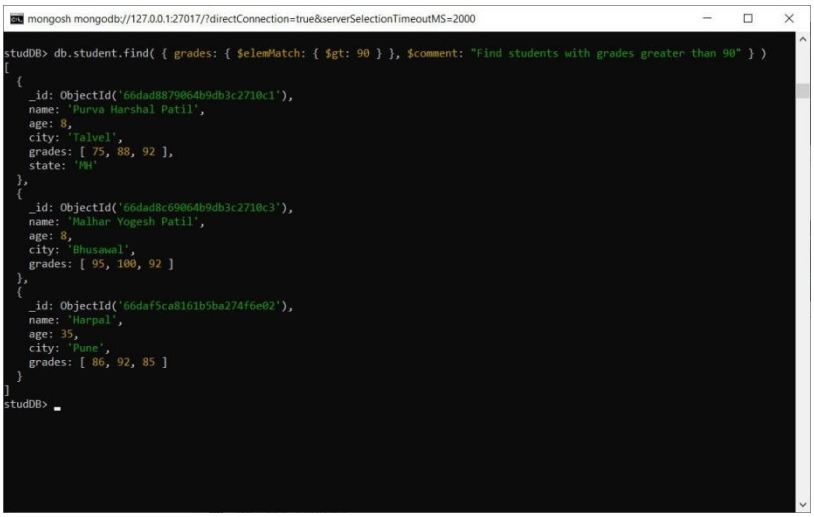
db.student.find({ flags: { $bitsAllSet: 1, $bitsAnySet: 4 }}).pretty()

This query matches students whose flags field has the first bit set (1) and at least one bit set (1) in the third bit position.

Example – 1 Use $comment Query Operators:

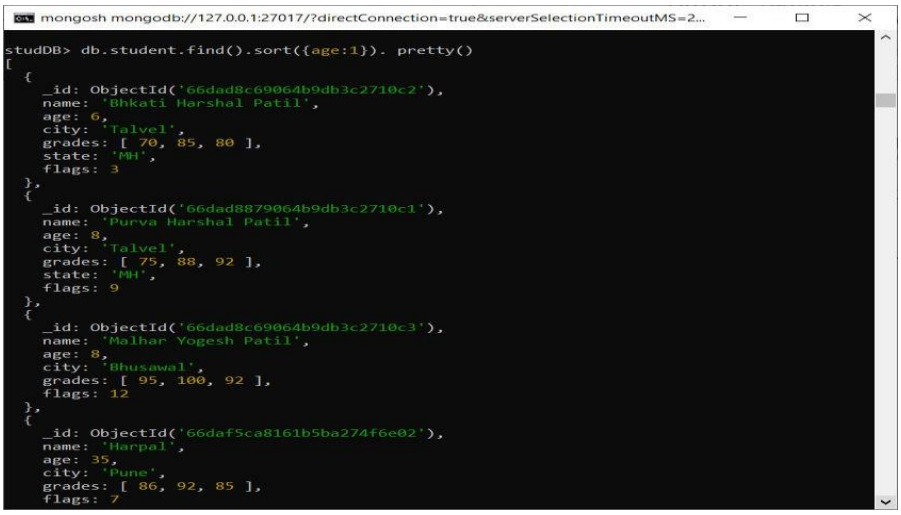
Find students who have a grade greater than 90 and include a comment..

db.student.find({ grades: { $elemMatch: { $gt: 90 } }, $comment: "Find students with grades greater than 90" }) .pretty()



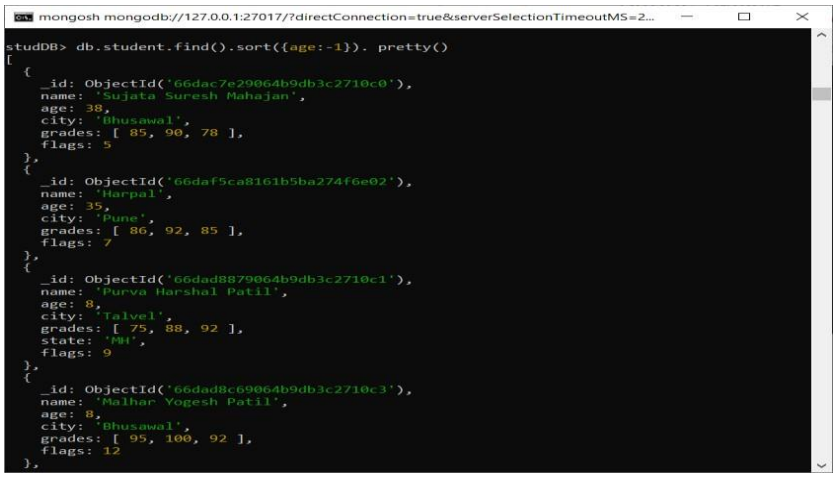
Example 1: Ascending Order orting

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



Example 2: Descending Order Sorting

db.student.find().sort({age:-1}). pretty()

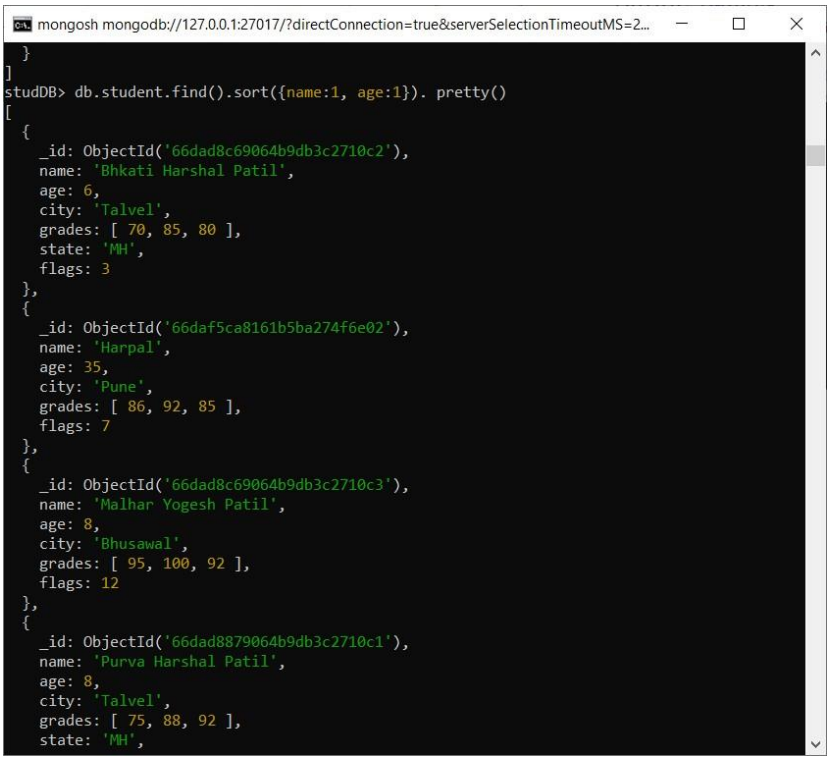


Example 3: Ascending Order by Student Name

db.student.find().sort({name:1}). pretty()

Example 4: Ascending Order by Student Name and age

db.student.find().sort({name:1, age:1}). pretty()



Practical 9:- Develop a task manager application using AngularJS for the frontend and MongoDB for the backend.

Step 1: Set Up the Backend (Node.js + Express.js + MongoDB)

1. Install Node.js and MongoDB:

- Download and install Node.js from nodejs.org.
- Install MongoDB by following the instructions on the [MongoDB installation page](https://www.mongodb.com/docs/manual/installation/).

2. Initialize a Node.js Project:

```
mkdir  mean-simple-app
cd mean-simple-app
npm init -y
```

3. Install Required Dependencies:

```
npm install express mongoose body-parser cors
```

4. Create the Server:

- Create a file named `**server.js**` and set up a basic Express server:

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');
const path = require('path');
const app = express();
const port = 3000;
// Middleware
app.use(bodyParser.json());
app.use(cors());
// Serve static files from the 'public' directory
app.use(express.static(path.join(_dirname, 'public')));
// MongoDB connection
mongoose.connect('mongodb://localhost:27017/mean_db', { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.error(err));

// Define a simple schema and model
const ItemSchema = new mongoose.Schema({
```

```
  name: String
});
const Item = mongoose.model('Item', ItemSchema);
// Routes
app.get('/items', async (req, res) => {
  try {
    const items = await Item.find();
    res.json(items);
  } catch (err) {
    res.status(500).send(err);
  }
});
// Serve the index.html for any other routes not handled
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});
// Start server
app.listen(port, () => {
  console.log(`Server running on http://localhost:${port}`);
});
```

Step 2: Set Up the Frontend (AngularJS)

mean-simple-app/
├── node_modules/
├── public/
│ ├── index.html
│ ├── app.js
│ ├── services.js
│ ├── controllers.js
│ └── views/
│ └── home.html
├── server.js
├── package.json
└── package-lock.json

1. Create an AngularJS Project:

- Create a basic HTML file (`**index.html**`) for your AngularJS application:

```
<!DOCTYPE html>
<html ng-app="simpleApp">
<head>
 <title>Simple AngularJS App</title>
 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular-
route.min.js"></script>
 <script src="app.js"></script>
 <script src="services.js"></script>
 <script src="controllers.js"></script>
</head>
<body>
 <div ng-view></div>
</body>
</html>
```

2. Set Up AngularJS Application:

- Create `**app.js**` for defining the AngularJS module and routes:

```
angular.module('simpleApp', ['ngRoute'])
.config(function($routeProvider) {
 $routeProvider
  .when('/', {
   templateUrl: 'views/home.html',
   controller: 'MainController'
  })
  .otherwise({
   redirectTo: '/'
  });
});
```

3. Create AngularJS Services:

- Create `**services.js**` to handle HTTP requests to the backend:

```
angular.module('simpleApp')
.service('ItemService', function($http) {
 this.getItems = function() {
  return $http.get('http://localhost:3000/items');
```

```
  };
});
```

4. Create Controllers:

- Create `**controllers.js**` to handle the interaction between the view and the backend:

```
angular.module('simpleApp')
.controller('MainController', function($scope, ItemService) {
  $scope.items = [];
  $scope.getItems = function() {
    ItemService.getItems().then(function(response) {
    console.log('Items fetched:', response.data);
      $scope.items = response.data;
    }).catch(function(error) {
      console.error('Error fetching items:', error);
    });
  };
  // Initial load
  $scope.getItems();
});
```

5. Create Views:

- Create a folder named `**views**` and add `**home.html**` for the main view:

```
<div>
  <h1>Creating AngularJS services to interact with MongoDB</h1>
  <ul>
    <li ng-repeat="item in items">{{ item.name }}</li>
  </ul>
</div>
```

Step 3: Run and Test the Application

1. Start MongoDB:

- Ensure MongoDB is running. You can start MongoDB by running `mongod` in your terminal.

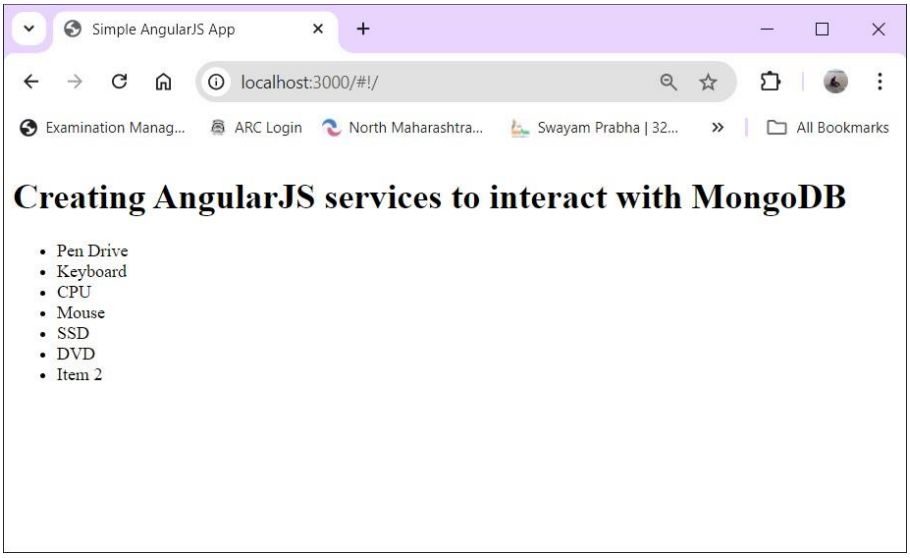
2. Start the Backend Server:

- In the root directory of your project, run the following command:

```
node server.js
```

3. Open the Frontend in a Browser:

Open ` **http://localhost:3000/**` in your browser. You should see your AngularJS application running, displaying the list of items fetched from MongoDB.



Practical 10:- To Create Student interface to stored and update the information.

Step 1: Set Up the Backend (Node.js + Express.js + MongoDB)

1. **Install Node.js and MongoDB:**

- Download and install Node.js from nodejs.org.
- Install MongoDB by following the instructions on the [MongoDB installation page](https://www.mongodb.com/docs/manual/installation/).

Step 2: Set up a new Node.js project:

- Create a new directory for your project and navigate into it:

  mkdir Interation-angularjs-mongodb

  cd Interation-angularjs-mongodb

- Initialize a new Node.js project: **npm init –y**
- Install Required Dependencies:

  npm install express mongoose body-parser cors

Step 3: Create the Express server:

- Create a file named server.js in the project root::

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();
// Middleware
app.use(bodyParser.json());
app.use(cors());
// MongoDB connection
mongoose.connect('mongodb://localhost:27017/mean_db')
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.log(err));
// Define a simple schema and model
const ItemSchema = new mongoose.Schema({
  name: String,
  quantity: Number
});
const Item = mongoose.model('Item', ItemSchema);
```

```
// Define API endpoints
app.get('/api/items', async (req, res) => {
try {
    const items = await Item.find();
    res.json(items);
  } catch (err) {
    res.status(500).send(err);
  }
});

app.post('/api/items', async (req, res) => {
  try {
    const newItem = new Item(req.body);
    const savedItem = await newItem.save();
    res.json(savedItem);
  } catch (err) {
    res.status(500).send(err);
  }
});
// Serve static files from the 'public' directory
app.use(express.static('public'));
// Fallback to index.html for single-page applications
// Ensure the path is correct and the file exists
app.get('*', (req, res) => {
console.log('Serving index.html');
  res.sendFile(_dirname + '/public/index.html', function(err) {
    if (err) {
      res.status(500).send(err);
    }
  });
});
const port = process.env.PORT || 3000;
app.listen(port, () => console.log(`Server running on port ${port}`));
```

Step 4: Run the server

node server.js

Step 5: Set Up the Frontend (AngularJS)

1. Create an AngularJS Project:

- If you haven't already, create an AngularJS project structure inside your **Interation-angularjs-mongodb** directory:
    - mkdir public
    - cd public
    - npm init -y
    - npm install angular
- Create the necessary files and folders:

public
├── index.html
├── app
│ ├── app.module.js
│ ├── app.config.js
│ └── app.controller.js Fff

2. Set Up AngularJS Application:

- Create the **index.html** file:

```
<html ng-app="myApp">
<head>
 <title>AngularJS and MongoDB Integration</title>
 <script src="node_modules/angular/angular.min.js"></script>
 <script src="app/app.module.js"></script>
 <script src="app/app.config.js"></script>
 <script src="app/app.controller.js"></script>
</head>
<body>
 <div ng-controller="MainController as ctrl">
  <form ng-submit="ctrl.addItem()">
   <input type="text" ng-model="ctrl.newItem.name" placeholder="Item Name" required>
   <input type="number" ng-model="ctrl.newItem.quantity" placeholder="Quantity" required>
   <button type="submit">Add Item</button>
```

```
      </form>
      <ul>
        <li ng-repeat="item in ctrl.items">{{ item.name }} - {{ item.quantity }}</li>
      </ul>
    </div>
  </body>
</html>
```

3. Create the AngularJS application files:

- app/app.module.js:

```
angular.module('myApp', []);
```

- app/app.config.js:

```
angular.module('myApp').config(function($httpProvider) {
  // Configuration if needed
});
```

- app/app.controller.js:

```
angular.module('myApp').controller('MainController', function($http) {
  var vm = this;
  vm.items = [];
  vm.newItem = {};
  // Fetch items from the server
  vm.getItems = function() {
    $http.get('/api/items').then(function(response) {
      vm.items = response.data;
    }).catch(function(error) {
      console.error('Error fetching items:', error);
    });
  };
  // Add a new item
  vm.addItem = function() {
    $http.post('/api/items', vm.newItem).then(function(response) {
      vm.items.push(response.data);
      vm.newItem = {}; // Clear the form
    }).catch(function(error) {
      console.error('Error adding item:', error);
```

```
  });
 };
  // Initial fetch
 vm.getItems();
});
```

Step 6: Run and Test the Application

1. Start MongoDB:

- Ensure MongoDB is running. You can start MongoDB by running `mongod` in your terminal.

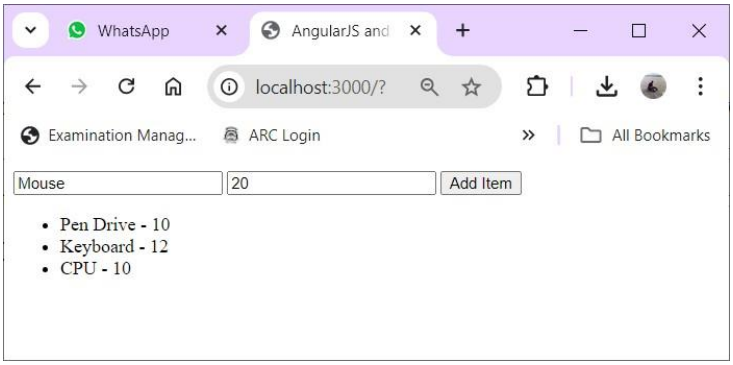
2. Start the Backend Server:

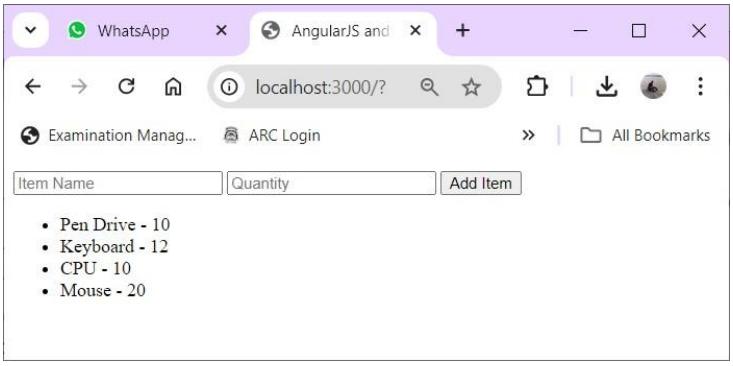
- In the root directory of your project, run the following command:

  node server.js

3. Open the Frontend in a Browser:

- Open your browser and navigate to http://localhost:3000. You should see the AngularJS application, and you can add and view items that are stored in the MongoDB database.





Practical 12:- Implement a user interface where users can view, add, update, and delete tasks with MongoDB Database.

Step 1: Set Up the Backend (Node.js + Express.js + MongoDB)

1. Install Node.js and MongoDB:

- Download and install Node.js from nodejs.org.
- Install MongoDB by following the instructions on the [MongoDB installation page](https://www.mongodb.com/docs/manual/installation/).

2. Initialize a Node.js Project:

```
mkdir mean-stack-app
cd mean-stack-app
npm init -y
```

3. Install Required Dependencies:

```
npm install express mongoose body-parser cors
```

4. Create the Server:

- Create a file named `**server.js**` and set up a basic Express server:

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');
const path = require('path');
const app = express();
const port = 3000;
// Middleware
app.use(bodyParser.json());
app.use(cors());
// Serve static files from the 'public' directory
app.use(express.static(path.join(_dirname, 'public')));
// MongoDB connection
mongoose.connect('mongodb://localhost:27017/mean_db', { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.error(err));
// Define a simple schema and model
const ItemSchema = new mongoose.Schema({
  name: String,
```

```
  quantity: Number
});
const Item = mongoose.model('Item', ItemSchema);
// Routes
app.get('/items', async (req, res) => {
  try {
    const items = await Item.find();
    res.json(items);
  } catch (err) {
    res.status(500).send(err);
  }
});
app.post('/items', async (req, res) => {
  try {
    const newItem = new Item(req.body);
    const savedItem = await newItem.save();
    res.json(savedItem);
  } catch (err) {
    res.status(500).send(err);
  }
});
app.put('/items/:id', async (req, res) => {
  try {
    const updatedItem = await Item.findByIdAndUpdate(req.params.id, req.body, {
new: true });
    res.json(updatedItem);
  } catch (err) {
    res.status(500).send(err);
  }
});
app.delete('/items/:id', async (req, res) => {
  try {
    await Item.findByIdAndDelete(req.params.id);
    res.json({ message: 'Item deleted' });
  } catch (err) {
```

```
    res.status(500).send(err);
  }
});
// Serve the index.html for any other routes not handled
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});
// Start server
app.listen(port, () => {
  console.log(`Server running on http://localhost:${port}`);
});
```

Step 2: Set Up the Frontend (AngularJS)

mean-stack-app/
├── node_modules/
├── public/
│ ├── index.html
│ ├── app.js
│ ├── services.js
│ ├── controllers.js
│ └── views/
│ └── home.html
├── server.js
├── package.json
└── package-lock.json

1. Create an AngularJS Project:

Create a basic HTML file (`**index.html**`) for your AngularJS application:

```
<!DOCTYPE html>
<html ng-app="meanApp">
<head>
  <title>MEAN Stack App</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular-
route.min.js"></script>
  <script src="app.js"></script>
```

```
  <script src="services.js"></script>
  <script src="controllers.js"></script>
</head>
<body>
  <div ng-view></div>
</body>
</html>
```

2. Set Up AngularJS Application:

Create `**app.js**` for defining the AngularJS module and routes:

```
angular.module('meanApp', ['ngRoute'])
.config(function($routeProvider) {
  $routeProvider
    .when('/', {
      templateUrl: 'views/home.html',
      controller: 'MainController'
    })
    .otherwise({
      redirectTo: '/'
    });
});
```

3. Create AngularJS Services:

Create `**services.js**` to handle HTTP requests to the backend:

```
angular.module('meanApp')
.service('ItemService', function($http) {
  this.getItems = function() {
    return $http.get('http://localhost:3000/items');
  };
  this.addItem = function(item) {
    return $http.post('http://localhost:3000/items', item);
  };
  this.updateItem = function(id, item) {
    return $http.put('http://localhost:3000/items/' + id, item);
  };
  this.deleteItem = function(id) {
    return $http.delete('http://localhost:3000/items/' + id);
```

```
  };
});
```

4. Create Controllers:

Create `**controllers.js**` to handle the interaction between the view and the backend:

```
angular.module('meanApp')
.controller('MainController', function($scope, ItemService) {
  $scope.items = [];
  $scope.newItem = {};
  $scope.editingItem = null;
  $scope.getItems = function() {
    ItemService.getItems().then(function(response) {
      $scope.items = response.data;
    });
  };
  $scope.addItem = function() {
    ItemService.addItem($scope.newItem).then(function(response) {
      $scope.items.push(response.data);
      $scope.newItem = {};
    });
  };
  $scope.updateItem = function(item) {
    ItemService.updateItem(item._id, item).then(function(response) {
      $scope.editingItem = null;
      $scope.getItems();
    });
  };
  $scope.deleteItem = function(id) {
    ItemService.deleteItem(id).then(function(response) {
      $scope.getItems();
    });
  };
  $scope.startEditing = function(item) {
    $scope.editingItem = angular.copy(item);
  };
  $scope.cancelEditing = function() {
```

```
    $scope.editingItem = null;
  };
  // Initial load
  $scope.getItems();
 });
```

5. Create Views:

Create a folder named **`views`** and add **`home.html`** for the main view:

```
<div>
 <h1>MEAN Stack App</h1>
 <form ng-submit="addItem()">
  <input type="text" ng-model="newItem.name" placeholder="Item Name" required>
  <input type="number" ng-model="newItem.quantity" placeholder="Quantity" required>
  <button type="submit">Add Item</button>
 </form>
 <ul>
  <li ng-repeat="item in items">
   <span ng-if="editingItem && editingItem._id === item._id">
    <input type="text" ng-model="editingItem.name" required>
    <input type="number" ng-model="editingItem.quantity" required>
    <button ng-click="updateItem(editingItem)">Save</button>
    <button ng-click="cancelEditing()">Cancel</button>
   </span>
   <span ng-if="!editingItem || editingItem._id !== item._id">
    {{ item.name }} - {{ item.quantity }}
    <button ng-click="startEditing(item)">Edit</button>
    <button ng-click="deleteItem(item._id)">Delete</button>
   </span>
  </li>
 </ul>
</div>
```

Step 3: Run and Test the Application

1. Start MongoDB:

Ensure MongoDB is running. You can start MongoDB by running `mongod` in your terminal.

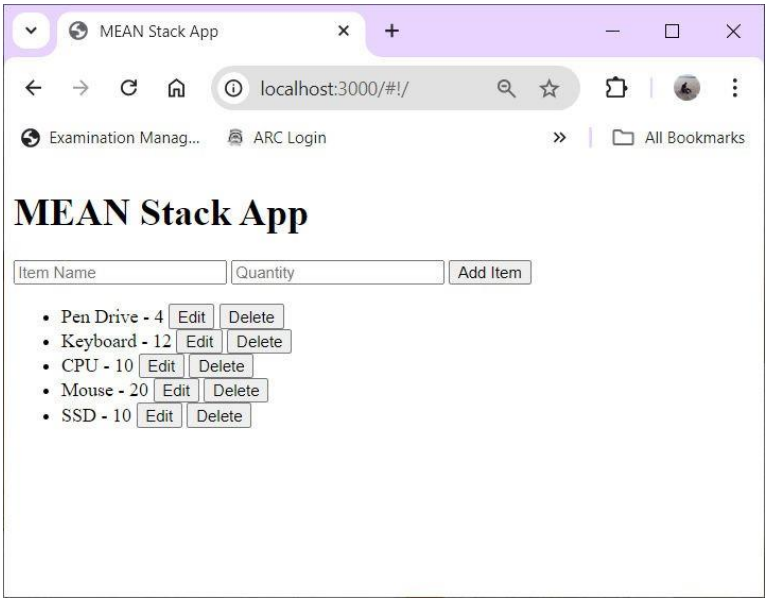
2. Start the Backend Server:

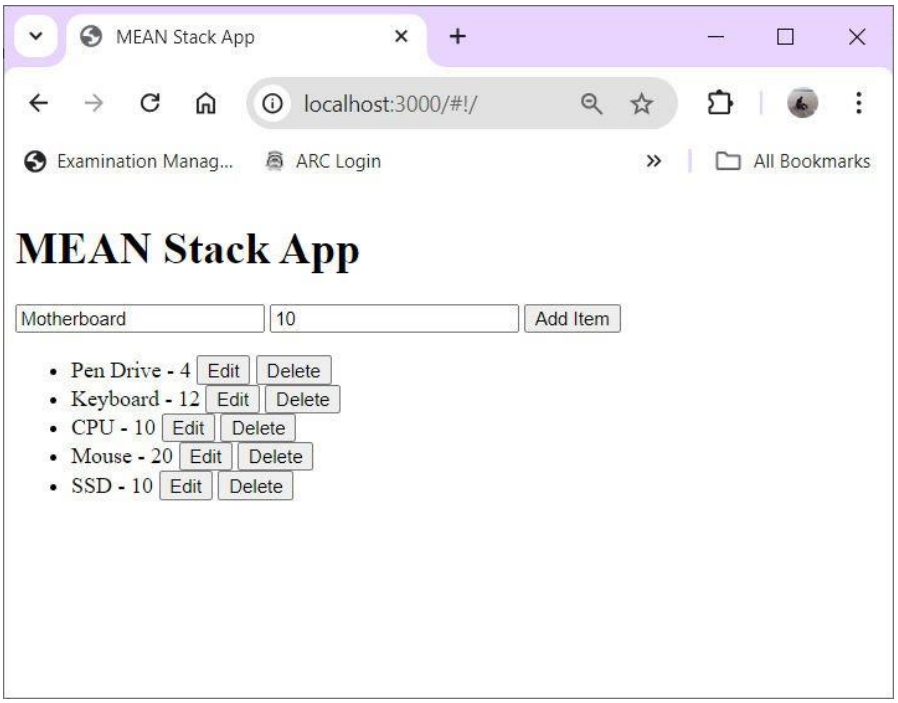
In the root directory of your project, run the following command:

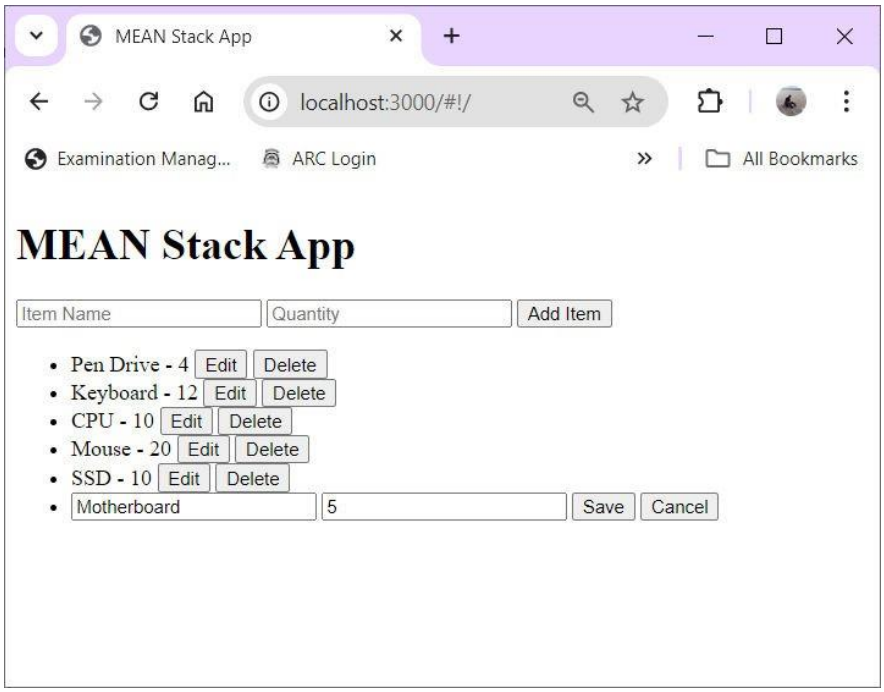
node server.js

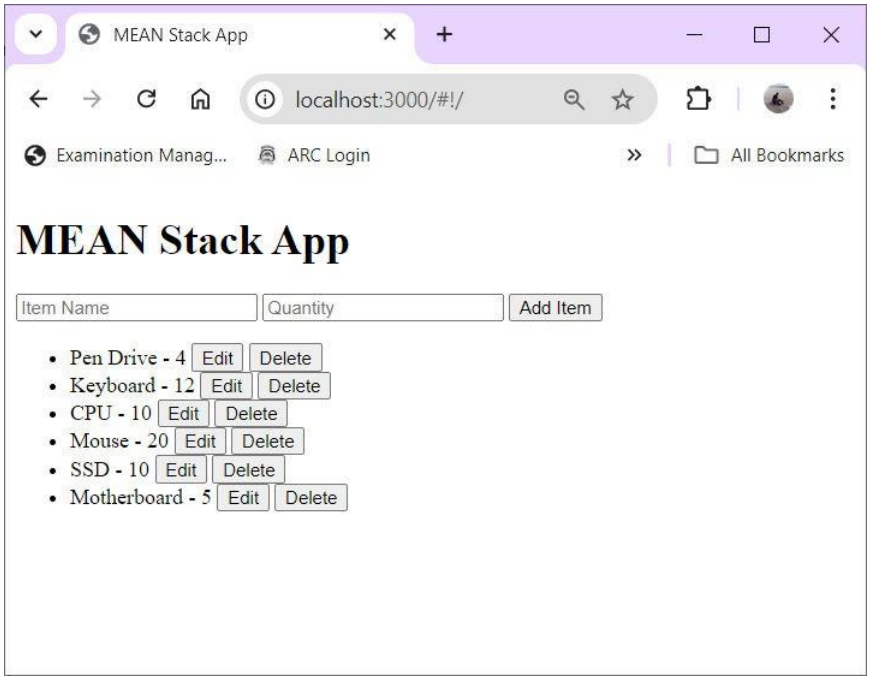
3. Open the Frontend in a Browser:

Open ` **http://localhost:3000/**` in your browser. You should see the MEAN stack application running, where you can add items and view the list of items stored in MongoDB.






MEAN Stack App
localhost:3000/#!/
Examination Manag...
ARC Login
All Bookmarks
MEAN Stack App
Item Name
Quantity
Add Item
Pen Drive - 4 Edit Delete
Keyboard - 12 Edit Delete
CPU - 10 Edit Delete
Mouse - 20 Edit Delete
SSD - 10 Edit Delete
Motherboard
5
Save
Cancel


MEAN Stack App
localhost:3000/#!/
Examination Manag...
ARC Login
All Bookmarks
MEAN Stack App
Item Name
Quantity
Add Item
Pen Drive - 4 Edit Delete
Keyboard - 12 Edit Delete
CPU - 10 Edit Delete
Mouse - 20 Edit Delete
SSD - 10 Edit Delete
Motherboard - 5 Edit Delete