

**Shram Sadhana Bombay Trust Sanchlit  
Arts, Commerce and Science College, Bambhori, Jalgaon  
Bachelor of Computer Application (B.C.A.)**



**Bachelor of Computer Applications**

**LAB MANUAL ON  
Web Development Technology-III 507(A)**

**Name:**

**Class:**

**Sem:**

**Roll No:**

**Seat No:**

**Name of faculty: Ms. H. S. Jadhvani**

**Academic Year: 20 - 20**

**Shram Sadhana Bombay Trust Sanchlit**  
**Arts, Commerce and Science College, Bambhori, Jalgaon**  
**Bachelor of Computer Application (B.C.A.)**

**Practical: 01**

**DOP:**

**DOC:**

---

---

**Title:** Create a user-friendly interface with a clean and proper design.

**Objective:** The objective of this practical is to create a user-friendly interface with a clean and proper design. The interface should be visually appealing and enhance the user experience. You will use HTML, CSS, and JavaScript (or AngularJS) to implement a responsive and consistent design that works well across all devices.

**Theory:**

1. Install Visual Studio Code and setting up the visual Studio Code
2. Install Nodejs (LTS)file and setting up the Nodejs

**Steps:**

1. Open visual studio code
2. Install some necessary Extensions that required to perform this practical
  - i. **Live Server by ritwick dey**
  - ii. **Angular Snippets by john papa**
  - iii. **Html-CSS-JavaScript Support by ecmel**
  - iv. **Prettier by prettier**
  - v. **Npm**
3. Install some necessary packages through npm (node package manager) to do this practical. This Packages can be installed using visual studio code terminal
  - i. **npm init -y**
  - ii. **npm install angular**
4. Now Create a Directory in visual studio code terminal  
**mkdir (Directory Name)**
5. After creating a Directory we need to change the directory  
**cd (Directory Name)**

6. After changing there is a link of your directory open it using (ctrl+click)

7. In created folder create a file with the extension of html

### **Index.html**

### **8. Program:**

```
<!DOCTYPE html>
<html ng-app="todoApp">
  <head>
    <meta charset="UTF-8">
    <title>To-Do List</title>
    <style>
      /* CSS */
      body { font-family: Arial; display: flex; justify-content: center; align-items:
center; height: 100vh; margin: 0; }

.container { width: 300px; padding: 20px; box-shadow: 0 0 10px rgba(0,0,0,0.1); }

      input, button { padding: 10px; margin: 5px; }
      .done { text-decoration: line-through; }

    </style>

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

  </script>
  <!-- javaScript -->

  <script>

    angular.module('todoApp', []).controller('TodoController', function() {
      this.todos = [{text: 'Learn AngularJS', done: false}, {text: 'Build a to-do
app', done: false}];

      this.addTodo = () => { if (this.newTodo) { this.todos.push({text:
this.newTodo, done: false}); this.newTodo = ""; } };
      this.removeTask = (index) => { this.todos.splice(index, 1); };
    });
```

```

</script>
</head>
<body ng-controller="TodoController as ctrl">
  <div class="container">

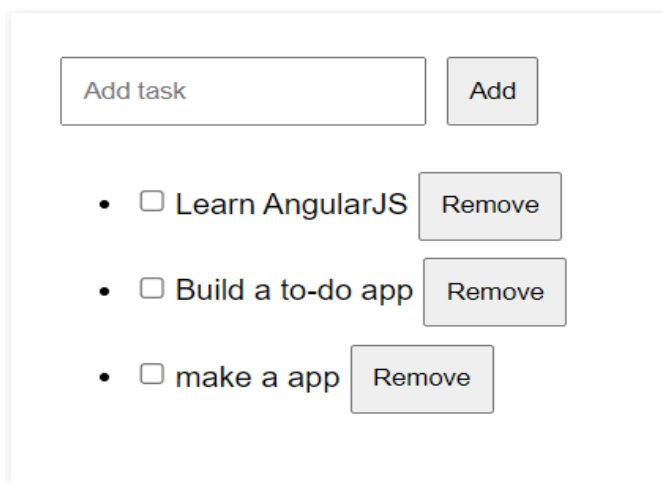
    <form ng-submit="ctrl.addTodo()"><input ng-model="ctrl.newTodo" placeholder="Add
    task"><button>Add</button></form>

    <ul><li ng-repeat="todo in ctrl.todos"><input type="checkbox" ng-
    model="todo.done"><span ng-class="{ 'done': todo.done }">{{ todo.text }}</span><button
    ng-click="ctrl.removeTask($index)">Remove</button></li></ul>

  </div>
</body>
</html>

```

## 9. Output:



Add task

- ☐ Learn AngularJS
- ☐ Build a to-do app
- ☐ make a app

## 10. Conclusion:

---

**Submitted By:**

**Checked By:**  
**Ms. H. S. Jadhvani**

**Sign:**

**Name:**

**Roll No:**

**Shram Sadhana Bombay Trust Sanchlit**  
**Arts, Commerce and science College, Bambhori, Jalgaon**  
**Bachelor of Computer Application (B.C.A.)**

**Practical: 02**

**DOP:**

**DOC:**

---

**Title: Develop AngularJS program to create a login form, with validation for the username and password fields.**

**Objective:** The objective of this practical is to develop a login form using AngularJS, incorporating validation for the username and password fields. The form will ensure that users provide valid input before submitting, enhancing security and user experience by preventing incomplete or incorrect data from being processed.

**Theory:**

1. Install Visual Studio Code and setting up the visual Studio Code
2. Install Nodejs (LTS)file and setting up the Nodejs

**Steps:**

1. Open visual studio code
2. Install some necessary Extensions that required to perform this practical
  - i. **Live Server by ritwick dey**
  - ii. **Angular Snippets by john papa**
  - iii. **Html-CSS-JavaScript Support by ecmel**
  - iv. **Prettier by prettier**
  - v. **Npm**
3. Install some necessary packages through npm (node package manager) to do this practical. This Packages can be installed using visual studio code terminal
  - i. **npm init -y**
  - ii. **npm install angular**
  - iii. **npm install -g angular@1.x**
4. Now Create a Directory in visual studio code terminal  
**mkdir (Directory Name)**

5. After creating a Directory we need to change the directory

**cd (Directory Name)**

6. After changing there is a link of your directory open it using (ctrl+click)

7. In created folder create a file with the extension of html

**Index.html**

## 8. Program:

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

  <style>

    body { background: burlywood; display: flex; justify-content: center; align-items: center;
height: 100vh; margin: 0; }

    .form-container { background: chocolate; padding: 70px; border-radius: 8px; box-
shadow: 0 0 10px black; text-align: center; }

    .error { color: red; }
    form.ng-pristine.ng-invalid.ng-invalid-required { display: flex; flex-direction: column; }
    button { color: red; padding: 9px; border-radius: 20px; }

  </style>
  </head>
  <body ng-controller="loginCtrl">
    <div class="form-container">
      <h2>Firewall Authentication</h2>
      <form name="loginForm" ng-submit="submitForm()" novalidate>

        <label>Username: <input type="text" name="username" ng-model="user.username"
required>

        <span class="error" ng-show="loginForm.username.$dirty &&
loginForm.username.$invalid">Required</span>
```

</label>

<label>Password: <input type="password" name="password" ng-model="user.password" required>

<span class="error" ng-show="loginForm.password.\$dirty && loginForm.password.\$invalid">Required</span>

</label>

<button type="submit" ng-disabled="loginForm.\$invalid">Login</button>

<div ng-show="loginSuccess" style="color: black;">Login Successful!</div>

</form>

</div>

<script>

angular.module('loginApp', []).controller('loginCtrl', function(\$scope) {

    \$scope.submitForm = function() {

        \$scope.loginSuccess = \$scope.user.username === 'student' && \$scope.user.password === 'icoet';

        if (!\$scope.loginSuccess) \$scope.user = {};

    };

});

</script>

</body>

</html>

## 9. Output:



**Firewall Authentication**

Username:  Password:

Login Successful!

**10. Conclusion:** \_\_\_\_\_

**Submitted By:**

**Checked By:**

**Ms. H. S. Jadhvani**

**Sign:**

**Name:**

**Roll No:**



**Shram Sadhana Bombay Trust Sanchlit  
Arts, Commerce and science College, Bambhori, Jalgaon  
Bachelor of Computer Application (B.C.A.)**

**Practical: 03**

**DOP:**

**DOC:**

---

**Title: To demonstrate AngularJS services and data bind.**

**Objective:** The objective is to demonstrate AngularJS services and data binding by creating a simple application that uses services to manage and share data across different components, while showcasing how data binding automatically updates the view with changes in the model.

**Theory:**

1. Install Visual Studio Code and setting up the visual Studio Code
2. Install Nodejs (LTS)file and setting up the Nodejs

**Steps:**

1. Open visual studio code
2. Install some necessary Extensions that required to perform this practical
  - i. **Live Server by ritwick dey**
  - ii. **Angular Snippets by john papa**
  - iii. **Html-CSS-JavaScript Support by ecmel**
  - iv. **Prettier by prettier**
  - v. **Npm**
3. Install some necessary packages through npm (node package manager) to do this practical. This Packages can be installed using visual studio code terminal
  - iv. **npm init -y**
  - v. **npm install angular**
  - vi. **npm install -g angular@1.x**
4. Now Create a Directory in visual studio code terminal  
**mkdir (Directory Name)**

5. After creating a Directory we need to change the directory

**cd (Directory Name)**

6. After changing there is a link of your directory open it using (ctrl+click)

7. In created folder create a file with the extension of html

**Index.html**

**8. Program:**

```
<!DOCTYPE html>
```

```
<html ng-app="myApp">
```

```
<head>
```

```
<title>Data Binding Example</title>
```

```
<script
```

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

```
<script>
```

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

```
.service('UserService', function() {
```

```
    this.getUserData = () => ({ name: 'SSBT COET', email: 'ssbt  
coer@example.com' });  
})
```

```
.controller('MainController', ['$scope', 'UserService', function($scope, UserService) {  
    $scope.title = 'AngularJS Example';  
    $scope.userData = UserService.getUserData();  
    $scope.user = { name: " " };  
}]);
```

```
</script>
```

```
</head>
```

```
<body ng-controller="MainController">
```

```
<h1>{{ title }}</h1>
```

```
<label>Enter your name:</label>
```

```
<input type="text" ng-model="user.name">
```

```
<h2>Welcome, {{ user.name }}!</h2>
```

```
<h3>Stored Data from Service:</h3>
```

```
<p>Name: {{ userData.name }}</p>
```

```
<p>Email: {{ userData.email }}</p>
```

```
</body>
```

```
</html>
```

## 9. Output:

### AngularJS Example

Enter your name:

**Welcome, AngularJS!**

**Stored Data from Service:**

Name: SSBT COET

Email: ssbt coer@example.com

10. Conclusion: \_\_\_\_\_

**Submitted By:**

**Checked By:**

**Ms. H. S. Jadhvani**

**Sign:**

**Name:**

**Roll No:**

**Shram Sadhana Bombay Trust Sanchlit**  
**Arts, Commerce and science College, Bambhori, Jalgaon**  
**Bachelor of Computer Application (B.C.A.)**

**Practical: 04**

**DOP:**

**DOC:**

---

**Title:** To display tasks in a list with checkboxes for marking completion as per user choice.

**Objective:** The objective is to create a task list application where users can mark tasks as completed by checking a checkbox, allowing for easy tracking of task progress.

**Theory:**

1. Install Visual Studio Code and setting up the visual Studio Code
2. Install Nodejs (LTS)file and setting up the Nodejs

**Steps:**

1. Open visual studio code
2. Install some necessary Extensions that required to perform this practical
  - i. **Live Server by ritwick dey**
  - ii. **Angular Snippets by john papa**
  - iii. **Html-CSS-JavaScript Support by ecmel**
  - iv. **Prettier by prettier**
  - v. **Npm**
3. Install some necessary packages through npm (node package manager) to do this practical. This Packages can be installed using visual studio code terminal
  - i. **npm init -y**
  - ii. **npm install angular**
  - iii. **npm install -g angular@1.x**
4. Now Create a Directory in visual studio code terminal  
**mkdir (Directory Name)**

5. After creating a Directory we need to change the directory

**cd (Directory Name)**

6. After changing there is a link of your directory open it using (ctrl+click)

7. In created folder create a file with the extension of html

**Index.html**

**8. Program:**

```
<!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>
    <style>
      body { font-family: Arial, sans-serif; }
      .task-list { list-style: none; padding: 0; }
      .task-item { margin: 5px 0; display: flex; align-items: center; }
      .task-item label { margin-left: 10px; }
      .completed label { text-decoration: line-through; color: gray; }
      .add-task, .subject-select { margin: 10px 0; }
    </style>
  </head>
  <body>
    <h1>Task List</h1>
    <div class="subject-select">
      <label for="subject-select">Select Subject:</label>
      <select id="subject-select">
        <option>AngularJS</option>
        <option>E-Commerce</option>
        <option>Cloud Computing</option>
      </select>
    </div>
    <input type="text" id="task-input" placeholder="New task">
    <button onclick="addTask()">Add Task</button>
    <h2>Tasks</h2>
    <ul id="task-list" class="task-list"></ul>
  </body>
</html>
<script>
function addTask() {
```

```

const taskText = document.getElementById('task-input').value.trim();
if (!taskText) return;

const subject = document.getElementById('subject-select').value;
const taskItem = document.createElement('li');
taskItem.className = 'task-item';
taskItem.innerHTML = `
  <input type="checkbox" onclick="toggleCompletion(this)">
  <label>${taskText} - <strong>${subject}</strong></label>
  <button onclick="removeTask(this)">Remove</button>
`; document.getElementById('task-list').appendChild(taskItem);
document.getElementById('task-input').value = "";
}

function toggleCompletion(checkbox) {
checkbox.parentElement.classList.toggle('completed', checkbox.checked);
}

function removeTask(button) {
button.parentElement.remove();
}

</script>
</body>
</html>

```

## 9. Output:

# Task List

Select Subject:

## Tasks

☐ assignment - **AngularJS**

**10.Conclusion:** \_\_\_\_\_

**Submitted By:**

**Checked By:**  
**Ms. H. S. Jadhvani**

**Sign:**

**Name:**

**Roll No:**

**Shram Sadhana Bombay Trust Sanchlit**  
**Arts, Commerce and science College, Bambhori, Jalgaon**  
**Bachelor of Computer Application (B.C.A.)**

**Practical: 05**

**DOP:**

**DOC:**

---

**Title:** To use ng-if directive to display tasks in the UI.

**Objective:** The objective is to use the ng-if directive in AngularJS to conditionally display tasks in the UI based on specific criteria, enhancing the user experience by showing or hiding elements dynamically.

**Theory:**

1. Install Visual Studio Code and setting up the visual Studio Code
2. Install Nodejs (LTS)file and setting up the Nodejs

**Steps:**

1. Open visual studio code
2. Install some necessary Extensions that required to perform this practical
  - i. **Live Server by ritwick dey**
  - ii. **Angular Snippets by john papa**
  - iii. **Html-CSS-JavaScript Support by ecmel**
  - iv. **Prettier by prettier**
  - v. **Npm**
3. Install some necessary packages through npm (node package manager) to do this practical. This Packages can be installed using visual studio code terminal
  - i. **npm init -y**
  - ii. **npm install angular**
  - iii. **npm install -g angular@1.x**
4. Now Create a Directory in visual studio code terminal  
**mkdir (Directory Name)**
5. After creating a Directory we need to change the directory  
**cd (Directory Name)**



6. After changing there is a link of your directory open it using (ctrl+click)
7. In created folder create a file with the extension of html

### **Index.html**

#### **8. Program**

```
<!DOCTYPE html>
<html ng-app="taskApp">
<head>
  <meta charset="UTF-8">
  <title>Task List</title>
<scriptsrc="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js">

</script>
  <script>

    // AngularJS application setup
    var app = angular.module('taskApp', []);

    app.controller('TaskController', function($scope) {
      $scope.tasks = [
        { name: 'Complete assignment', done: true },
        { name: 'Read a book', done: true },
        { name: 'Go for a walk', done: false }
      ];
    });
  </script>
</head>
<body ng-controller="TaskController">
  <h2>Tasks to Complete</h2>
  <ul>
    <li ng-repeat="task in tasks" ng-if="!task.done">
      {{ task.name }}
    </li>
  </ul>

  <h2>Completed Tasks</h2>
  <ul>
    <li ng-repeat="task in tasks" ng-if="task.done">
      {{ task.name }}
    </li>
  </ul>
</body>
</html>
```

```
</li>
</ul>
```

```
</body>
```

```
</html>
```

## 9. Output:

### Tasks to Complete

- Go for a walk

### Completed Tasks

- Complete assignment
- Read a book

10. Conclusion: \_\_\_\_\_

**Submitted By:**

**Checked By:**

**Ms. H. S. Jadhvani**

**Sign:**

**Name:**

**Roll No:**

**Shram Sadhana Bombay Trust Sanchlit**  
**Arts, Commerce and science College, Bambhori, Jalgaon**  
**Bachelor of Computer Application (B.C.A.)**

**Practical: 06**

**DOP:**

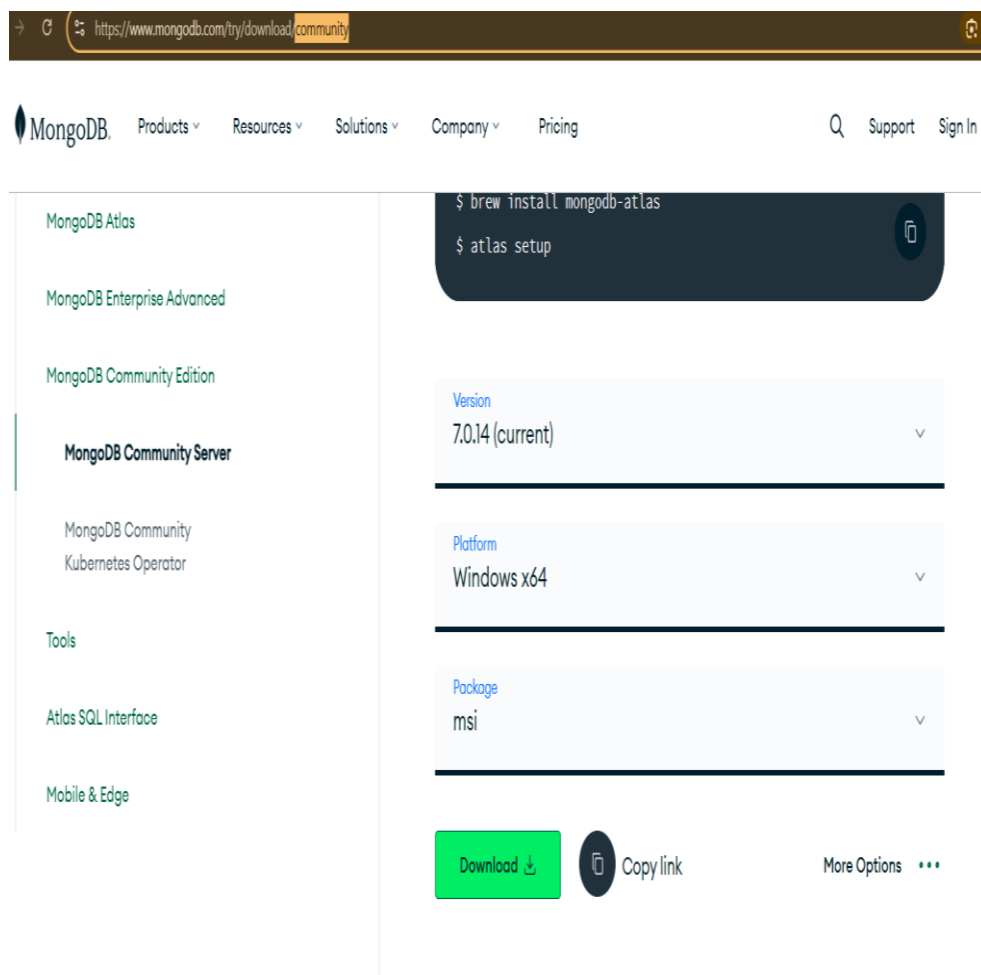
**DOC:**

**Title: To create database and structure in MongoDB.**

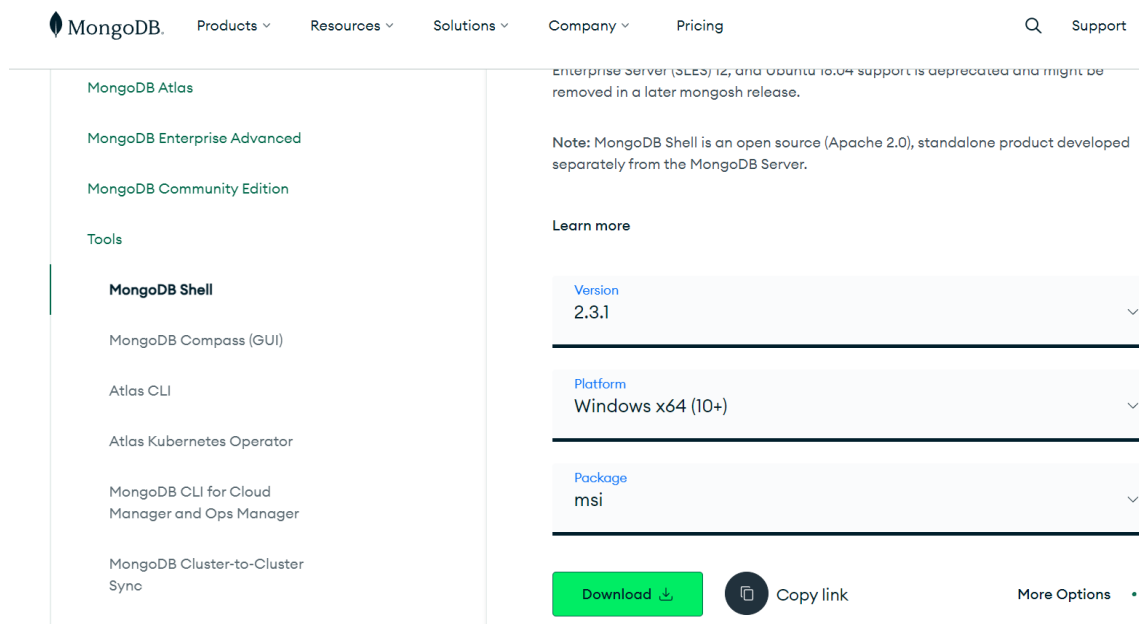
**Objective:** To set up a MongoDB database and define its structure by creating collections and specifying the schema for organizing and storing data efficiently.

**Theory:**

1. Install MongoDB (community server) MSI package



## 2. Install MongoDB Shell (Command line)



The screenshot shows the MongoDB website's navigation bar with links for Products, Resources, Solutions, Company, and Pricing. A search icon and 'Support' link are on the right. The left sidebar lists various MongoDB products and tools, with 'MongoDB Shell' highlighted under the 'Tools' section. The main content area for 'MongoDB Shell' includes a note about deprecated support for older versions, a 'Learn more' link, and three dropdown menus for 'Version' (2.3.1), 'Platform' (Windows x64 (10+)), and 'Package' (msi). At the bottom, there are 'Download' and 'Copy link' buttons, and a 'More Options' link.

## 3. After installing this we will check the version of MongoDB

**Mongod --version**

## 4. Open a MongoDB shell using CMD

**Mongosh**

## 5. Creating Database in MongoDB

**use bca**

**Output:**

```
test> use bca
switched to db bca
bca>
```

6. Creating a Collection in MongoDB

**Db.createCollection("mongodb")**

**Output:**

```
bca> db.createCollection("mongodb")
{ ok: 1 }
bca>
```

7. Conclusion: \_\_\_\_\_

**Submitted By:**

**Sign:**

**Name:**

**Roll No:**

**Checked By:**

**Ms. H. S. Jadhvani**

**Shram Sadhana Bombay Trust Sanchlit  
Arts, Commerce and science College, Bambhori, Jalgaon  
Bachelor of Computer Application (B.C.A.)**

**Practical: 07**

**DOP:**

**DOC:**

---

**Title:** To demonstrate insert, update, delete, select operations in MongoDB

**Objective:** To demonstrate the basic CRUD (Create, Read, Update, Delete) operations in MongoDB, showcasing how to insert, update, delete, and select data from a collection.

**Theory:** In this practical we will learn how to create update select delete the data

**1. Create A database:**

```
test> use bca  
switched to db bca  
bca>
```

**2. Create A Collection**

```
bca> db.createCollection("mongodb")  
{ ok: 1 }  
bca>
```

**3. Insert into the collection**

There are Two ways to insert the data into the collection

**i. insertOne**

```
bca> db.mongodb.insertOne([ { name: "john", age: 22, course: "BCA", city: "london" } ])
{
  acknowledged: true,
  insertedId: ObjectId('66df0c648a2e7225b12710bc')
}
bca>
```

**ii. insertMany**

```
bca> db.mongodb.insertMany([ { name: "jack", age: 26, course: "MCA", city: "pune" }, { name: "marry", age: 27, course: "MBA", city: "mumbai" }, { name: "ramesh", age: 28, course: "BSC", city: "delhi" } ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66df0e228a2e7225b12710bd'),
    '1': ObjectId('66df0e228a2e7225b12710be'),
    '2': ObjectId('66df0e228a2e7225b12710bf')
  }
}
bca>
```

#### 4. Update the data that we inserted

There are Two ways to update a data

##### i. updateOne

```
bca> db.mongodb.updateOne({ name: "jack" }, { $set: { age: 32 } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

##### ii. updateMany

```
bca> db.mongodb.updateMany({ age: 22 }, { $set: { age: 25 } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
bca>
```

#### 5. Select the data

```
bca> db.mongodb.find({ age: 32 })
[
  {
    _id: ObjectId('66df0e228a2e7225b12710bd'),
    name: 'jack',
    age: 32,
    course: 'MCA',
    city: 'pune'
  }
]
bca>
```



## 6. Delete the data

There are Two ways to delete the data

### i. deleteOne

```
bca> db.mongodb.deleteOne({ name: "ramesh" })
{ acknowledged: true, deletedCount: 1 }
bca>
```

### ii. deleteMany

```
bca> db.mongodb.deleteMany( {} )
{ acknowledged: true, deletedCount: 6 }
bca>
```

## 7. Conclusion:

---

**Submitted By:**

**Sign:**

**Name:**

**Roll No:**

**Checked By:**

**Ms. H. S. Jadhvani**

**Shram Sadhana Bombay Trust Sanchlit  
Arts, Commerce and science College, Bambhori, Jalgaon  
Bachelor of Computer Application (B.C.A.)**

**Practical: 08**

**DOP:**

**DOC:**

---

**Title:** To create collection and insert records in collections.

**Objective:** To create a collection in MongoDB and insert multiple records into it, demonstrating the process of data storage within a database.

**Theory:** In this practical we will learn how to create a database and collection and how to insert multiple data into the the collection

**1. Create A database:**

```
test> use bca  
switched to db bca  
bca>
```

**2. Create A Collection**

```
bca> db.createCollection("mongodb")  
{ ok: 1 }  
bca>
```

**3. Insert into the collection**

There are Two ways to insert the data into the collection

i. **insertOne**

```
bca> db.mongodb.insertOne([ { name: "john", age: 22, course: "BCA", city: "london" } ])
{
  acknowledged: true,
  insertedId: ObjectId('66df0c648a2e7225b12710bc')
}
bca>
```

## ii. insertMany

```
bca> db.mongodb.insertMany([ { name: "jack", age: 26, course: "MCA", city: "pune" }, { name: "marry", age: 27, course: "MBA", city: "mumbai" }, { name: "ramesh", age: 28, course: "BSC", city: "delhi" } ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66df0e228a2e7225b12710bd'),
    '1': ObjectId('66df0e228a2e7225b12710be'),
    '2': ObjectId('66df0e228a2e7225b12710bf')
  }
}
bca>
```

## 4. Conclusion:

---

**Submitted By:**

**Sign:**

**Name:**

**Roll No:**

**Checked By:**

**Ms. H. S. Jadhvani**

**Shram Sadhana Bombay Trust Sanchlit**  
**Arts, Commerce and science College, Bambhori, Jalgaon**  
**Bachelor of Computer Application (B.C.A.)**

**Practical: 09**

**DOP:**

**DOC:**

---

**Title: Develop a task manager application using AngularJS for the frontend and MongoDB for the backend**

**Objective:** To develop a task manager application using AngularJS for the frontend interface and MongoDB for the backend database, enabling users to manage tasks with functionalities like adding, updating, deleting, and viewing tasks.

**Theory:**

1. Install Visual Studio Code and setting up the visual Studio Code
2. Install Nodejs (LTS)file and setting up the Nodejs

**Steps:**

1. Open visual studio code
2. Install some necessary Extensions that required to perform this practical
  - i. **Live Server by ritwick dey**
  - ii. **Angular Snippets by john papa**
  - iii. **Html-CSS-JavaScript Support by ecmel**
  - iv. **Prettier by prettier**
  - v. **Npm**
  - vi. **Mongodb by mongodb**
3. Install some necessary packages through npm (node package manager) to do this practical. This Packages can be installed using visual studio code terminal
  - i. **npm init -y**
  - ii. **npm install angular**
  - iii. **npm install express mongoose body-parse cors**
  - iv. **npm install mongoose**
  - v. **npm install cors body-parser**

4. Now Create a Directory in visual studio code terminal

**mkdir (Directory Name)**

5. After creating a Directory we need to change the directory

**cd (Directory Name)**

6. After changing there is a link of your directory open it using (ctrl+click)

7. After downloading the mongodb extension → click on the extension

→ Connect the server with the visual studio code using a connection string → **mongodb://localhost:27017** → connect

8. After connecting the localhost we can create a database where we insert a data that shown in our application

## 9. Creating database and inserting a data

i. **Creating database**

```
test> use try
switched to db try
```

ii. **Create collection**

```
try> db.createCollection("prac12")
{ ok: 1 }
```

iii. **Insert the data into the collection**

```
try> db.prac12.insertMany([ { task: "Assignment1", description: "complete unit 1", status: "pending" }, { task: "Assignment2", description: "complete unit 2", status: "pending" }, { task: "Assignment3", description: "complete unit 3", status: "pending" }, { task: "Assignment4", description: "complete unit 4", status: "pending" } ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66e129616eb6dce6002710bc'),
    '1': ObjectId('66e129616eb6dce6002710bd'),
    '2': ObjectId('66e129616eb6dce6002710be'),
    '3': ObjectId('66e129616eb6dce6002710bf')
  }
}
```

10. Open visual studio code → open your folder → create a files

i. **Server.js**

ii. **Index.html**

### i. Server .js

```
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;

app.use(cors());
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, 'public')));

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/try')
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.error('Failed to connect to MongoDB', err));

const taskSchema = new mongoose.Schema({
  task: String,
  description: String,
  status: String
}, { collection: 'prac12' }); // Use the collection name `prac12`

const Task = mongoose.model('Task', taskSchema);

// Get all tasks
app.get('/tasks', async (req, res) => {
  try {
    const tasks = await Task.find();
    res.json(tasks);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch tasks' });
  }
});
```

```

// Add a new task
app.post('/tasks', async (req, res) => {
  try {
    const task = new Task(req.body);
    await task.save();
    res.json(task);
  } catch (error) {
    res.status(500).json({ error: 'Failed to add task' });
  }
});

// Update an existing task
app.put('/tasks/:id', async (req, res) => {
  try {
    const task = await Task.findByIdAndUpdate(req.params.id, req.body, { new: true });
    res.json(task);
  } catch (error) {
    res.status(500).json({ error: 'Failed to update task' });
  }
});

// Delete a task
app.delete('/tasks/:id', async (req, res) => {
  try {
    await Task.findByIdAndDelete(req.params.id);
    res.json({ message: 'Task deleted' });
  } catch (error) {
    res.status(500).json({ error: 'Failed to delete task' });
  }
});

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

```

11.Run this program in visual studio using a command  
**node server.js**

## ii. Index.html

```
<!DOCTYPE html>
<html ng-app="taskManagerApp">
<head>
<title>Task Manager</title>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js">
</script>
    <script>
        angular.module('taskManagerApp', [])
            .controller('TaskController', ['$scope', '$http', function($scope, $http) {
                const apiUrl = 'http://localhost:3000/tasks';

                function loadTasks() {
                    $http.get(apiUrl).then(response => {
                        $scope.tasks = response.data;
                    });
                }

                loadTasks();

                $scope.addTask = function() {
                    $http.post(apiUrl, $scope.newTask).then(response => {
                        $scope.tasks.push(response.data);
                        $scope.newTask = { };
                    });
                };

                $scope.deleteTask = function(id) {
                    $http.delete(`${apiUrl}/${id}`).then(() => {
                        $scope.tasks = $scope.tasks.filter(task => task._id !== id);
                    });
                };

                $scope.editTask = function(task) {
                    $scope.newTask = angular.copy(task);
                    $scope.newTask._id = task._id;
                };
            }]);
    </script>
</html>
```



```

$scope.updateTask = function() {
$http.put(`${apiUrl}/${$scope.newTask._id}`, $scope.newTask).then(response
=> {
const index = $scope.tasks.findIndex(task => task._id === response.data._id);
$scope.tasks[index] = response.data;
$scope.newTask = {};
});
};
}]);

</script>
<style>
body {
font-family: Arial, sans-serif;
margin: 20px;
}
form {
margin-bottom: 20px;
}
input, select {
margin-right: 10px;
}
ul {
list-style: none;
padding: 0;
}
li {
margin-bottom: 10px;
}
button {
margin-left: 5px;
}
</style>
</head>
<body ng-controller="TaskController">
<h1>Task Manager</h1>
<form ng-submit="newTask._id ? updateTask() : addTask()">
<input type="text" ng-model="newTask.task" placeholder="Task" required>
<input type="text" ng-model="newTask.description" placeholder="Description">
<select ng-model="newTask.status">

```

```

<option value="pending">Pending</option>
<option value="complete">Complete</option>
<option value="incomplete">Incomplete</option>
</select>
<button type="submit">{{ newTask._id ? 'Update Task' : 'Add Task' }}</button>
</form>
<ul>
<li ng-repeat="task in tasks">
<span>{{ task.task }} - {{ task.description }} - {{ task.status }}</span>
<button ng-click="editTask(task)">Edit</button>
<button ng-click="deleteTask(task._id)">Delete</button>
</li>
</ul>
</body>
</html>

```

## 12.Output:

# Task Manager

Task	Description	▼	Add Task
------	-------------	---	----------

Assignment1 - complete unit 1 - pending	Edit	Delete
Assignment2 - complete unit 2 - pending	Edit	Delete
Assignment3 - complete unit 3 - pending	Edit	Delete
Assignment4 - complete unit 4 - pending	Edit	Delete
assignment5 - complete unit 5 - complete	Edit	Delete

❖ Now we insert a data

## Task Manager

<input type="text" value="Task"/>	<input type="text" value="Description"/>	<input type="text" value="v"/>	<input type="button" value="Add Task"/>
-----------------------------------	--	--------------------------------	---

Assignment1 - complete unit 1 - pending

Assignment2 - complete unit 2 - pending

Assignment3 - complete unit 3 - pending

Assignment4 - complete unit 4 - pending

assignment5 - complete unit 5 - complete

Assignment6 - complete unit 6 - complete

**13. Conclusion:** \_\_\_\_\_

**Submitted By:**

**Sign:**

**Name:**

**Roll No:**

**Checked By:**

**Ms. H. S. Jadhvani**

**Shram Sadhana Bombay Trust Sanchlit**  
**Arts, Commerce and science College, Bambhori, Jalgaon**  
**Bachelor of Computer Application (B.C.A.)**

**Practical: 10**

**DOP:**

**DOC:**

---

**Title: To Create Student interface to stored and update the information.**

**Objective:** To design and implement a student interface using MongoDB and Visual Studio Code that allows users to efficiently store, update, and manage student information. This interface will provide functionalities for adding new student records, updating existing data, and ensuring seamless integration with the MongoDB database for real-time data management.

**Theory:**

1. Install Visual Studio Code and setting up the visual Studio Code
2. Install Nodejs (LTS)file and setting up the Nodejs

**Steps:**

1. Open visual studio code
2. Install some necessary Extensions that required to perform this practical
  - i. **Live Server by ritwick dey**
  - ii. **Angular Snippets by john papa**
  - iii. **Html-CSS-JavaScript Support by ecmel**
  - iv. **Prettier by prettier**
  - v. **Npm**
  - vi. **Mongodb by mongodb**
3. Install some necessary packages through npm (node package manager) to do this practical. This Packages can be installed using visual studio code terminal
  - i. **npm init -y**
  - ii. **npm install angular**
  - iii. **npm install express mongoose body-parse cors**

iv. **npm install mongoose**

v. **npm install cors body-parser**

4. Now Create a Directory in visual studio code terminal

**mkdir (Directory Name)**

5. After creating a Directory we need to change the directory

**cd (Directory Name)**

6. After changing there is a link of your directory open it using  
(ctrl+click)

7. After downloading the mongodb extension → click on the extension  
→ Connect the server with the visual studio code using a connection  
string → **mongodb://localhost:27017** → connect

8. After connecting the localhost we can create a database where we  
insert a data that shown in our application

## 9. Creating database and inserting a data

### i. Creating a database

```
test> use student
switched to db student
student>
```

### ii. Creating a collection

```
student> db.createCollection("stu")
{ ok: 1 }
student>
```

### iii. Insert the data into the collection

```
test> db.stu.insertMany([ { name: "jack", age: 23, course: "Bca" }, { name: "john",
age: 24, course: "Mba" }, { name: "marry", age: 25, course: "Bsc" }, { name: "
ramesh", age: 26, course: "engg" }, { name: "suresh", age: 29, course: "phd" } ] )
{ }
```

## 10. Open visual studio code → open your folder → create a files

i. **Server.js**

ii. **Index.html**

i. **Server.js**

```
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;
app.use(cors());
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, 'public')));
// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/student')
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.error('Failed to connect to MongoDB', err));
const studentSchema = new mongoose.Schema({
  name: String,
  age: Number,
  course: String,
  year: String
}, { collection: 'stu' }); // Collection name is 'stu'
const Student = mongoose.model('Student', studentSchema);
// Get all students
app.get('/students', async (req, res) => {
  try {
```

```

const students = await Student.find();
    res.json(students);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch students' });
  }
});

// Add a new student
app.post('/students', async (req, res) => {
  try {
    const student = new Student(req.body);
    await student.save();
    res.json(student);
  } catch (error) {
    res.status(500).json({ error: 'Failed to add student' });
  }
});

// Update an existing student
app.put('/students/:id', async (req, res) => {
  try {
    const student = await Student.findByIdAndUpdate(req.params.id,
req.body, { new: true });
    res.json(student);
  } catch (error) {
    res.status(500).json({ error: 'Failed to update student' });
  }
});

// Delete a student
app.delete('/students/:id', async (req, res) => {
  try {

```

```

        await Student.findByIdAndDelete(req.params.id);
        res.json({ message: 'Student deleted' });
    } catch (error) {
        res.status(500).json({ error: 'Failed to delete student' });
    }
});

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

```

11.Run this program using command

**Node server.js**

## ii. **Index.html**

```

<!DOCTYPE html>
<html lang="en" ng-app="studentApp">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Student Manager</title>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js">
</script>
    <script>
        const app = angular.module('studentApp', []);
        app.controller('studentController', function($scope, $http) {
            $scope.students = [];
            $scope.newStudent = {};

```



```

$scope.editStudent = null;
// Load students
$scope.loadStudents = function() {
    $http.get('http://localhost:3000/students')
        .then(function(response) {
            $scope.students = response.data;
        });
};

// Add student
$scope.addStudent = function() {
    $http.post('http://localhost:3000/students', $scope.newStudent)
        .then(function(response) {
            $scope.loadStudents();
            $scope.newStudent = {}; // Clear the form
        });
};

//
Edit student
$scope.editStudentData = function(student) {
    $scope.editStudent = angular.copy(student);
};

// Update student
$scope.updateStudent = function() {

$http.put(`http://localhost:3000/students/${$scope.editStudent._id}`,
$scope.editStudent)

        .then(function(response) {
            $scope.loadStudents();
            $scope.editStudent = null; // Clear the edit form
        });
};

```

```

    };

    // Delete student
    $scope.deleteStudent = function(id) {
        $http.delete(`http://localhost:3000/students/${id}`)
            .then(function(response) {
                $scope.loadStudents();
            });
    };

    // Initialize
    $scope.loadStudents();
});
</script>
</head>
<body ng-controller="studentController">
    <h1>Student Manager</h1>
    <div>
        <h2>Add New Student</h2>
        <form ng-submit="addStudent()">
            <label>Name:</label>
            <input type="text" ng-model="newStudent.name" required><br>
            <label>Age:</label>
            <input type="number" ng-model="newStudent.age" required><br>
            <label>Course:</label>
            <input type="text" ng-model="newStudent.course" required><br>
            <button type="submit">Add Student</button>
        </form>
    </div>

```



## 12.Output

# Student Manager

## Add New Student

Name:

Age:

Course:

## Students

- **john** - Age: 25 - Course: BCA
- **jack** - Age: 26 - Course: MCA
- **Marry** - Age: 27 - Course: Engg
- **ramesh** - Age: 28 - Course: Engg
- **suresh** - Age: 29 - Course: phd

## 13.Conclusion: \_\_\_\_\_

**Submitted By:**

**Checked By:**

**Ms. H. S. Jadhvani**

**Sign:**

**Name:**

**Roll No:**

**Shram Sadhana Bombay Trust Sanchlit**  
**Arts, Commerce and science College, Bambhori, Jalgaon**  
**Bachelor of Computer Application (B.C.A.)**

**Practical: 11**

**DOP:**

**DOC:**

---

**Title:** To display students information reports (All, Parametized)

**Objective:** To display students' information reports from a MongoDB database, including all records and parameterized results using filtering and sorting by age.

**Theory:**

1. Install Visual Studio Code and setting up the visual Studio Code
2. Install Nodejs (LTS)file and setting up the Nodejs

**Steps:**

1. Open visual studio code
2. Install some necessary Extensions that required to perform this practical
  - i. **Live Server by ritwick dey**
  - ii. **Angular Snippets by john papa**
  - iii. **Html-CSS-JavaScript Support by ecmel**
  - iv. **Prettier by prettier**
  - v. **Npm**
  - vi. **Mongodb by mongodb**
3. Install some necessary packages through npm (node package manager) to do this practical. This Packages can be installed using visual studio code terminal
  - i. **npm init -y**
  - ii. **npm install angular**
  - iii. **npm install express mongoose body-parse cors**
  - iv. **npm install mongoose**
  - v. **npm install cors body-parser**

4. Now Create a Directory in visual studio code terminal  
**mkdir (Directory Name)**
5. After creating a Directory we need to change the directory  
**cd (Directory Name)**
6. After changing there is a link of your directory open it using  
(ctrl+click)
7. After downloading the mongodb extension → click on the extension  
→ Connect the server with the visual studio code using a connection  
string → **mongodb://localhost:27017** → connect
8. After connecting the localhost we can create a database where we  
insert a data that shown in our application
9. **Creating database and inserting a data**

**iv. Creating a database**

```
test> use student
switched to db student
student>
```

**v. Creating a collection**

```
student> db.createCollection("stu")
{ ok: 1 }
student>
```

**vi. Insert the data into the collection**

```
test> db.stu.insertMany([ { name: "jack", age: 23, course: "Bca" }, { name: "john",
, age: 24, course: "Mba" }, { name: "marry", age: 25, course: "Bsc" }, { name: "
ramesh", age: 26, course: "engg" }, { name: "suresh", age: 29, course: "phd" } ] )
{ }
```

**vii. Sort data in parameterized using commands**

```
student> db.stu.find().sort({ age: 1 })
```

- 2 Way using AngularJS as frontend and MongoDB as backend

**10. Open visual studio code →open your folder→create a files**

- i. Server.js**
- ii. Index.html**

**i. Server.js**

```
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;
app.use(cors());
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, 'public')));
// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/student')
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.error('Failed to connect to MongoDB', err));
const studentSchema = new mongoose.Schema({
  name: String,
  age: Number,
  course: String,
  year: String
}, { collection: 'stu' }); // Collection name is 'stu'
const Student = mongoose.model('Student', studentSchema);
```

```

// Get all students with sorting
app.get('/students', async (req, res) => {
  try {
    // Default to ascending if no sortOrder provided
    const sortOrder = req.query.sortOrder === 'desc' ? -1 : 1;
    const students = await Student.find().sort({ age: sortOrder });
    res.json(students);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch students' });
  }
});

// Add a new student
app.post('/students', async (req, res) => {
  try {
    const student = new Student(req.body);
    await student.save();
    res.json(student);
  } catch (error) {
    res.status(500).json({ error: 'Failed to add student' });
  }
});

// Update an existing student
app.put('/students/:id', async (req, res) => {
  try {
    const student = await Student.findByIdAndUpdate(req.params.id, req.body, {
      new: true });
    res.json(student);
  } catch (error) {
    res.status(500).json({ error: 'Failed to update student' });
  }
});

```



```

    // Delete a student
    app.delete('/students/:id', async (req, res) => {
      try {
        await Student.findByIdAndDelete(req.params.id);
        res.json({ message: 'Student deleted' });
      } catch (error) {
        res.status(500).json({ error: 'Failed to delete student' });
      }
    });

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

```

11.Run this code using command

**Node server.js**

## ii. Index.html

```

<!DOCTYPE html>
<html lang="en" ng-app="studentApp">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Student Manager</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <script>
    const app = angular.module('studentApp', []);

    app.controller('studentController', function($scope, $http) {
      $scope.students = [];
      $scope.sortOrder = 'asc'; // Default sort order

```

```

// Load students with the specified sorting order
$scope.loadStudents = function() {
    $http.get(`http://localhost:3000/students?sortOrder=${$scope.sortOrder}`)
        .then(function(response) {
            $scope.students = response.data;
        });
};

// Change sort order and reload students
$scope.changeSortOrder = function(order) {
    $scope.sortOrder = order;
    $scope.loadStudents();
};

// Initialize
$scope.loadStudents();
});
</script>
</head>
<body ng-controller="studentController">
    <h1>Student Manager</h1>

    <div>
        <h2>Sort By Age:</h2>
        <button ng-click="changeSortOrder('asc')">Ascending</button>
        <button ng-click="changeSortOrder('desc')">Descending</button>
    </div>

    <div>
        <h2>Students</h2>
        <ul>
<li ng-repeat="student in students">

```

```
<strong>{{ student.name }}</strong> - {{ student.age }} - {{ student.course }} - {{
student.year }}
</li>
</ul>
</div>
</body>
</html>
```

## 12.Output

# Student Manager

## Sort By Age:

## Students

- **john** - 25 - BCA - TY
- **jack** - 26 - MCA - SY
- **Marry** - 27 - Engg - FY
- **ramesh** - 28 - Engg - BE
- **suresh** - 29 - phd -

## 13.Conclusion:

---

**Submitted By:**

**Sign:**

**Name:**

**Roll No:**

**Checked By:**

**Ms. H. S. Jadhvani**

**Shram Sadhana Bombay Trust Sanchlit**  
**Arts, Commerce and science College, Bambhori, Jalgaon**  
**Bachelor of Computer Application (B.C.A.)**

**Practical: 12**

**DOP:**

**DOC:**

---

**Title: Implement a user interface where users can view, add, update, and delete tasks with MongoDB Database.**

**Objective:** The objective of this practical is to create a user interface using AngularJS and MongoDB where users can view, add, update, and delete tasks. The task management system will allow users to interact with a MongoDB database, perform CRUD (Create, Read, Update, Delete) operations, and ensure real-time data updates.

**Theory:**

1. Install Visual Studio Code and setting up the visual Studio Code
2. Install Nodejs (LTS)file and setting up the Nodejs

**Steps:**

1. Open visual studio code
2. Install some necessary Extensions that required to perform this practical
  - i. **Live Server by ritwick dey**
  - ii. **Angular Snippets by john papa**
  - iii. **Html-CSS-JavaScript Support by ecmel**
  - iv. **Prettier by prettier**
  - v. **Npm**
  - vi. **Mongodb by mongodb**
3. Install some necessary packages through npm (node package manager) to do this practical. This Packages can be installed using visual studio code terminal
  - i. **npm init -y**
  - ii. **npm install angular**
  - iii. **npm install express mongoose body-parse cors**
  - iv. **npm install mongoose**

**v. npm install cors body-parser**

4. Now Create a Directory in visual studio code terminal

**mkdir (Directory Name)**

5. After creating a Directory we need to change the directory

**cd (Directory Name)**

6. After changing there is a link of your directory open it using (ctrl+click)

7. After downloading the mongodb extension → click on the extension → Connect the server with the visual studio code using a connection string → **mongodb://localhost:27017** → connect

8. After connecting the localhost we can create a database where we insert a data that shown in our application

**9. Creating database and inserting a data**

**i. Creating database**

```
test> use try
switched to db try
```

**ii. Create collection**

```
try> db.createCollection("prac12")
{ ok: 1 }
```

**iii. Insert the data into the collection**

```
try> db.prac12.insertMany([ { task: "Assignment1", description: "complete unit 1", status: "pending" }, { task: "Assignment2", description: "complete unit 2", status: "pending" }, { task: "Assignment3", description: "complete unit 3", status: "pending" }, { task: "Assignment4", description: "complete unit 4", status: "pending" } ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66e129616eb6dce6002710bc'),
    '1': ObjectId('66e129616eb6dce6002710bd'),
    '2': ObjectId('66e129616eb6dce6002710be'),
    '3': ObjectId('66e129616eb6dce6002710bf')
  }
}
```

**iv. Open visual studio code → open your folder → create a files**

**i. Server.js**

**ii. Index.html**

## i. **Server.js**

```
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;

app.use(cors());
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, 'public')));

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/student')
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.error('Failed to connect to MongoDB', err));

const studentSchema = new mongoose.Schema({
  name: String,
  age: Number,
  course: String,
  year: String
}, { collection: 'stu' }); // Collection name is 'stu'

const Student = mongoose.model('Student', studentSchema);

// Get all students with sorting
app.get('/students', async (req, res) => {
  try {
    // Default to ascending if no sortOrder provided
    const sortOrder = req.query.sortOrder === 'desc' ? -1 : 1;
    const students = await Student.find().sort({ age: sortOrder });
    res.json(students);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch students' });
  }
});

// Add a new student
app.post('/students', async (req, res) => {
  try {
```

```

        const student = new Student(req.body);
        await student.save();
        res.json(student);
    } catch (error) {
        res.status(500).json({ error: 'Failed to add student' });
    }
});

// Update an existing student
app.put('/students/:id', async (req, res) => {
    try {
        const student = await Student.findByIdAndUpdate(req.params.id, req.body,
        { new: true });
        res.json(student);
    } catch (error) {
        res.status(500).json({ error: 'Failed to update student' });
    }
});

// Delete a student
app.delete('/students/:id', async (req, res) => {
    try {
        await Student.findByIdAndDelete(req.params.id);
        res.json({ message: 'Student deleted' });
    } catch (error) {
        res.status(500).json({ error: 'Failed to delete student' });
    }
});

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

```

## ii. Index.html

```

<!DOCTYPE html>
<html lang="en" ng-app="studentApp">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Student Manager</title>

```

```

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
<script>
    const app = angular.module('studentApp', []);
    app.controller('studentController', function($scope, $http) {
        $scope.students = [];
        $scope.sortOrder = 'asc'; // Default sort order
        // Load students with the specified sorting order
        $scope.loadStudents = function() {
            $http.get(`http://localhost:3000/students?sortOrder=${$scope.sortOrder}`)
                .then(function(response) {
                    $scope.students = response.data;
                });
        };
        // Change sort order and reload students
        $scope.changeSortOrder = function(order) {
            $scope.sortOrder = order;
            $scope.loadStudents();
        };
        // Initialize
        $scope.loadStudents();
    });
</script>
</head>
<body ng-controller="studentController">
    <h1>Student Manager</h1>
    <div>
        <h2>Sort By Age:</h2>
        <button ng-click="changeSortOrder('asc')">Ascending</button>
        <button ng-click="changeSortOrder('desc')">Descending</button>
    </div>
    <div>
        <h2>Students</h2>
        <ul>
            <li ng-repeat="student in students">
                <strong>{{ student.name }}</strong> - {{ student.age }} - {{ student.course }} -
                {{ student.year }}
            </li>
        </ul>
    </div>
</body>
</html>

```



**Output:**

# Task Manager

<input type="text" value="Task"/>	<input type="text" value="Description"/>	<input type="button" value="v"/>	<input type="button" value="Add Task"/>
-----------------------------------	--	----------------------------------	---

Assignment1 - complete unit 1 - pending

Assignment2 - complete unit 2 - pending

Assignment3 - complete unit 3 - pending

Assignment4 - complete unit 4 - pending

assignment5 - complete unit 5 - complete

**11.Conclusion:**

---

**Submitted By:**

**Sign:**

**Name:**

**Roll No:**

**Checked By:**

**Ms. H. S. Jadhvani**