ADHIYAMAAN COLLEGE OF ENGINEERING
(Autonomous), Hosur

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(Accredited by NBA)

622CIP07 – FULL STACK DEVELOPMENT LABORATORY

(REGULATION - 2022)

STAFF INCHARGE **HOD**

Vision of the Institute

To foster ACE as a centre for nurturing and developing world class Engineers and Managers who
convert global challenges into opportunities through value-based quality education.

Mission of the Institute

M 1 : To impart value-based quality education through effective teaching-learning processes

M 2 : To nurture creativity, excellence and critical thinking by applying global competencyfactors to contribute and excel in the rapidly growing technological world.

M 3 : To continuously develop and improve holistic and innovative personality for globalMobility.

M 4 : To make ACE a centre for excellence.

Vision of the Department

To empower young minds to become resilient professionals, instilled with ethical principles andequipped with cutting-edge technologies to meet the evolving demands of the world

Mission of the Department

M 1 : To empower individuals with a comprehensive understanding of computer engineering principles and its applications through effective teaching and learning practices.

M 2 : To cultivate excellence and critical thinking, while leveraging global competency, thus enabling significant contributions to societal challenges in the fast-paced technologicallandscape.

M 3 : To facilitate the students to work with modern tools and technologies to foster innovation, azest for higher studies and to build leadership qualities by inculcating the spirit of ethical values.--

Program Educational Objectives (PEOs)

PEO1 : The graduates will have sound knowledge in Mathematics, Science and Engineering concepts necessary to formulate, analyse, design and solve Engineering problems and to prepare them for higher learning, research and industry.

PEO2 : The graduates will possess innovative skills to assess and apply the rapid changes intechnology and to engage in research leading to novel solutions for human, social and global competency.

PEO3 : The graduates will acquire knowledge and grab opportunities to work as teams in a multidisciplinary environment, communicate ideas effectively with diverse audiences demonstrate leadership qualities with ethical values and engage in lifelong learning.

622CIP07 FULL STACK DEVELOPMENT LABORATORY

OBJECTIVES

- To develop full stack applications with clear understanding of user interface, business logic and data Storage.
- To design and develop user interface screens for a given scenario
- To develop the functionalities as web components as per the requirements
- To implement the database according to the functional requirements
- To integrate the user interface with the functionalities and data storage

LIST OF EXPERIMENTS:

1. Create a basic Angular Application which Demonstrates About the parent and child components.
2. Develop a portfolio website for yourself which gives details about yourself for a potential recruiter
3. Create a simple micro blogging application (like twitter) that allows people to post their content which can be viewed by people who follow them.
4. Design a following static web pages required for online book store website 1. Home page 2. Login page 3. Catalogue page 4. Cart page 5. Registration page
5. Develop a Angular JS application that displays a list of shopping items and allow user to add and remove items from the list using directives and controllers
6. Write a program to create a simple calculator Application using React JS
7. Write a program to create a voting application using React JS.
8. Write a server side program for manipulating mongo DB from Node.js
9. Develop a project using Angular, Node.js, Express.js and Mongo DB from .This has to be CRUD Application for managing items(e.g., a task manager or inventory app) and emphasis on integrating Security Practices.

TOTAL: 60 PERIODS

COURSE OUTCOMES:

On successful completion of the course the students will be able to

CO1: Design full stack applications with clear understanding of user interface, business logic and data Storage

CO2: Design and develop user interface screens

CO3: Implement the functional requirements using appropriate tool

CO4: Design database based on the requirements

CO5: Develop all the necessary components of the application.

CO's-PO's& PSO's MAPPING

CO's/	622CIP07-FULL STACK DEVELOPMENT LABORATORY														
PO's	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3	3	1	3	1	1	1	2	1	1	1	2	2	1
CO2	3	3	3	2	3	1	1	1	2	1	1	1	2	2	1
CO3	3	3	3	3	3	1	1	1	2	1	1	1	2	2	1
CO4	3	3	3	3	3	2	1	1	1	1	2	1	1	2	1
CO5	3	3	3	3	2	1	1	1	1	1	1	1	2	2	1
AVG	3	3	3	2	3	1	1	1	1	1	1	1	2	2	1

1 - Low 2 - Medium, 3 - High,"-"- No Correlation

INDEX

Exp.NO:1 Date:	**Create a basic Angular Application which Demonstrates About the parent and child components.**

Aim: To Create a basic Angular Application which Demonstrates About the parent and child components.

ALGORITHM:

Step 1 : Create an Angular Project → ng new parent-child-demo → cd parent-child-demo

Step 2: Generate Components → ng g c parent → ng g c child

Step 3: Use @Input() and @Output() → Pass data from parent to child and emit events from child to parent.

Step 4: Update Templates → Parent uses <app-child> with bindings, and Child displays/handles data.

Step 5: Run & Test → ng serve and verify interaction.

Program:

app.module.ts :

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

import { AppComponent } from './app.component';
import { ParentComponent } from './parent/parent.component';
import { ChildComponent } from './child/child.component';

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

app.component.html :

```
<app-parent></app-parent>
```

parent.componet.ts:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-parent',
  templateUrl: './parent.component.html',
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {}
```

parent.component.html:

```
<h1>This is the Parent Component</h1>
<app-child></app-child>
```

child.component.ts:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent {}
```

child.componet.html:

```
h1>This is the Child Component</h1>
<app-parent></app-parent>
```

Output:


Ex-1
127.0.0.1:5500/index.html
Parent component : Hello from parent component
Child component: Hello from parent component

Result:

To Create a basic Angular Application which Demonstrates About the parent and child components is executed successfully.

Exp.NO:2 Date:	**Develop a portfolio website for yourself which gives details about yourself for a potential recruiter**

Aim: To Develop a portfolio website for yourself which gives details about yourself for a potential recruiter

ALGORITHM:

Step 1 : install node.js and initialize the project folder for angular js setup.

Step 2: Create index.html with section for header and embedded angular.js data binding.

Step 3: Apply advanced css and use angular controller.

Step 4: Open index.html in a browser.

Program:

app.component.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Portfolio</title>
  <link rel="stylesheet" href="css/style.css">
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-app="portfolioApp" ng-controller="MainController">
  <header>
    <h1>{{name}}</h1>
    <p>{{role}}</p>
  </header>
  <section>
    <h2>About Me</h2>
    <p>{{about}}</p>
  </section>
  <section>
    <h2>Skills</h2>
    <ul>
      <li ng-repeat="skill in skills">{{skill}}</li>
    </ul>
  </section>
  <section>
    <h2>Contact</h2>
    <p>Email: {{email}}</p>
    <p>Phone: {{phone}}</p>
  </section>

  <script>
    angular.module('portfolioApp', [])
    .controller('MainController', function ($scope) {
```

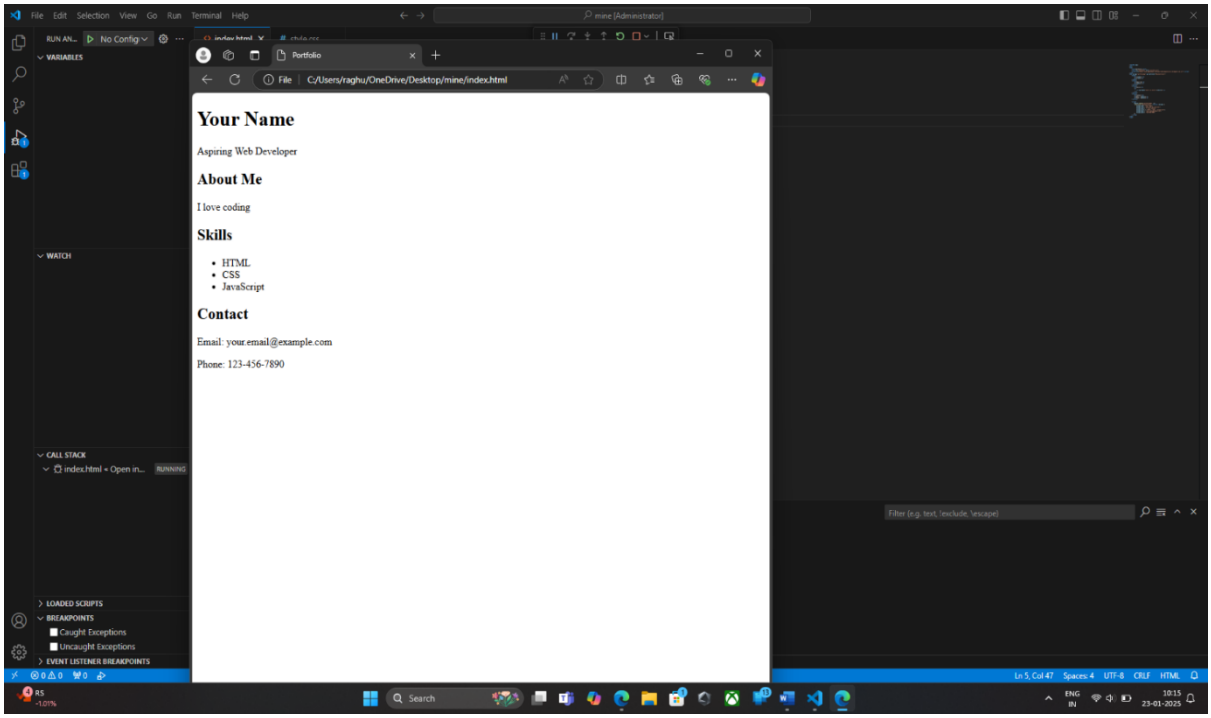
```
        $scope.name = "Your Name";
        $scope.role = "Aspiring Web Developer";
        $scope.about = "I love coding";
        $scope.skills = ["HTML", "CSS", "JavaScript"];
        $scope.email = "your.email@example.com";
        $scope.phone = "123-456-7890";
      });
  </script>
</body>
</html>
```

app.component.css:

```
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  text-align: center;
}
header {
  background-color: #333;
  color: white;
  padding: 1rem 0;
}
section {
  margin: 1rem;
}

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

Output:


Portfolio
C:/Users/raghu/OneDrive/Desktop/mine/index.html
Your Name
Aspiring Web Developer
About Me
I love coding
Skills
HTML
CSS
JavaScript
Contact
Email: your.email@example.com
Phone: 123-456-7890

Result:

To Develop a portfolio website for yourself which gives details about yourself for a potential recruiter is successfully executed.

Exp.NO:3 Date:	**Create a simple micro blogging application (like twitter) that allows people to post their content which can be viewed by people who follow them.**

Aim: To Create a simple micro blogging application (like twitter) that allows people to post their content which can be viewed by people who follow them.

ALGORITHM:

Step 1: Initialize project and install dependencies.

Step 2: Create backend with express server.

Step 3: Add routes for posts and feeds.

Step 4: Develop frontend using HTML/JS

Step 5: Connect and test the app.

Program:

BACKEND:

Server.js:

```
const express = require("express");
const bodyParser = require("body-parser");
const path = require("path");

const app = express();
app.use(bodyParser.json());

// Serve static files (like index.html) from the current directory
app.use(express.static(path.join(__dirname)));

// In-memory storage for posts
const posts = [];

// API to create a post
app.post("/post", (req, res) => {
  posts.push({ author: req.body.author, content: req.body.content });
  res.json({ message: "Post created" });
});
// API to get all posts (feed)
app.get("/feed", (req, res) => {
  res.json(posts);
});

// Start the server
app.listen(5000, () => {
  console.log("Server running on http://localhost:5000");
});
```

FRONTEND:

Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Micro-Blogging App</title>
</head>
<body>
  <h1>Micro-Blogging App</h1>

  <!-- Create a Post -->
  <h3>Create Post</h3>
  <input id="author" placeholder="Your name" />
  <input id="content" placeholder="Write something..." />
  <button onclick="createPost()">Post</button>

  <!-- Feed -->
  <h3>Feed</h3>
  <div id="feed"></div>

  <script>
    const API_URL = "http://localhost:5000";

    // Create a post
    async function createPost() {
      const author = document.getElementById("author").value;
      const content = document.getElementById("content").value;

      await fetch(`${API_URL}/post`, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
```

```
      body: JSON.stringify({ author, content }),
    });

    // Reload the feed
    loadFeed();
  }

  // Load the feed
  async function loadFeed() {
    const res = await fetch(`${API_URL}/feed`);
    const posts = await res.json();

    document.getElementById("feed").innerHTML = posts
      .map(post => `<p><strong>${post.author}:</strong> ${post.content}</p>`)
      .join("");
  }

  // Load feed on page load
  loadFeed();
 </script>
</body>
</html>
```

Output:

Micro-Blogging App

Create Post

Your name | Write something... | Post

Feed

john: hello world

rosy: heyy john

Result:

To Create a simple micro blogging application (like twitter) that allows people to post their content which can be viewed by people who follow them successfully executed

Exp.NO:4 Date:	**Design a following static web pages required for online book store website 1. Home page 2. Login page 3. Catalogue page 4. Cart page 5. Registration page**

Aim: To Design a following static web pages required for online book store website

1. Home page
2. Login page
3. Catalogue page
4. Cart page
5. Registration page

ALGORITHM:

Step 1 : Create Project Folder

Step 2: Create HTML Files for Each Page

Step 3: Design Each Page Using HTML & CSS

Step 4: Link Pages

Step 5: Test in Browser

Program:

Home.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Home - Online Bookstore</title>
</head>
<body>
  <h1>Online Bookstore</h1>
  <nav>
    <a href="home.html">Home</a> |
    <a href="catalogue.html">Catalogue</a> |
    <a href="cart.html">Cart</a> |
    <a href="login.html">Login</a> |
    <a href="registration.html">Register</a>
  </nav>
  <p>Welcome to our bookstore! Find your next favorite book.</p>
</body>
</html>
```

Login.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Login - Online Bookstore</title>
</head>
<body>
  <h2>Login</h2>
  <nav>
    <a href="home.html">Home</a> |
    <a href="catalogue.html">Catalogue</a> |
    <a href="cart.html">Cart</a> |
    <a href="login.html">Login</a> |
```

```
    <a href="registration.html">Register</a>
  </nav>
  <form>
    <input type="text" placeholder="Username" required><br>
    <input type="password" placeholder="Password" required><br>
    <button>Login</button>
  </form>
</body>
</html>
```

Catalogue.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Catalogue - Online Bookstore</title>
</head>
<body>
  <h2>Book Catalogue</h2>
  <nav>
    <a href="home.html">Home</a> |
    <a href="catalogue.html">Catalogue</a> |
    <a href="cart.html">Cart</a> |
    <a href="login.html">Login</a> |
    <a href="registration.html">Register</a>
  </nav>
  <ul>
    <li>Book 1</li>
    <li>Book 2</li>
    <li>Book 3</li>
  </ul>
</body>
</html>
```

Cart.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Cart - Online Bookstore</title>
</head>
<body>
  <h2>Your Cart</h2>
  <nav>
    <a href="home.html">Home</a> |
    <a href="catalogue.html">Catalogue</a> |
    <a href="cart.html">Cart</a> |
    <a href="login.html">Login</a> |
    <a href="registration.html">Register</a>
  </nav>
  <p>No items in your cart yet!</p>
</body>
</html>
```

Registration.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Register - Online Bookstore</title>
</head>
<body>
  <h2>Register</h2>
  <nav>
    <a href="home.html">Home</a> |
    <a href="catalogue.html">Catalogue</a> |
    <a href="cart.html">Cart</a> |
    <a href="login.html">Login</a> |
    <a href="registration.html">Register</a>
```

```
  </nav>
  <form>
    <input type="text" placeholder="Full Name" required><br>
    <input type="email" placeholder="Email" required><br>
    <input type="password" placeholder="Password" required><br>
    <button>Register</button>
  </form>
</body>
</html>
```

Output:


Login
Home | Catalogue | Cart | Login | Register
Username
Password
Login

Result:

Design a following static web pages required for online book store website 1. Home page 2. Login page 3. Catalogue page 4. Cart page 5. Registration page is executed successfully.

Exp.NO:5 Date:	**Develop a Angular JS application that displays a list of shopping items and allow user to add and remove items from the list using directives and controllers**

Aim: To Develop a Angular JS application that displays a list of shopping items and allow user to add and remove items from the list using directives and controllers

ALGORITHM:

Step 1 : Set Up Your Development Environment

Step 2: Create the HTML Code (index.html)

Step 3: Create the AngularJS Code (app.js)

Step 4: Run the Application

Program:

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Shopping Items Application</title>
   <script type="text/javascript"
   src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
   <script src="app.js"></script>
</head>
<body ng-app="myApp">
<div ng-controller="myCntrl">
   <h2>Shopping Application</h2>
   <h4>List of Shopping Items</h4>
   <table border="1">
      <tr>
         <th>SLNO</th>
         <th>Item</th>
      </tr>
      <tr ng-repeat="item in shoppingItems">
         <td>{{$index + 1}}</td>
         <td>{{item}}</td>
      </tr>
   </table>
<br/>
   <div>
      Enter an Item to Add: <input type="text" ng-model="newItem">
      <button ng-click="addItem()">Add Item</button>
   </div>

   <div>
```

```
        Select an Item to Remove:
        <select ng-model="selectItem" ng-options="item for item in shoppingItems"></select>
        <button ng-click="removeItem()">Remove Item</button>
    </div>
</div>
</body>
</html>
```

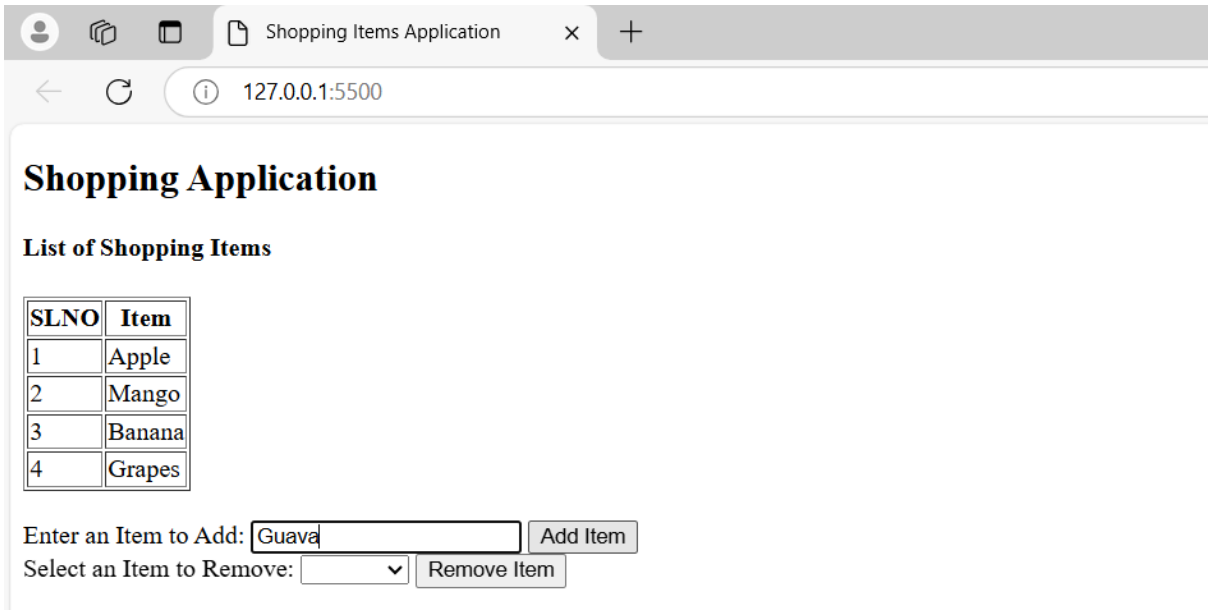
app.js:

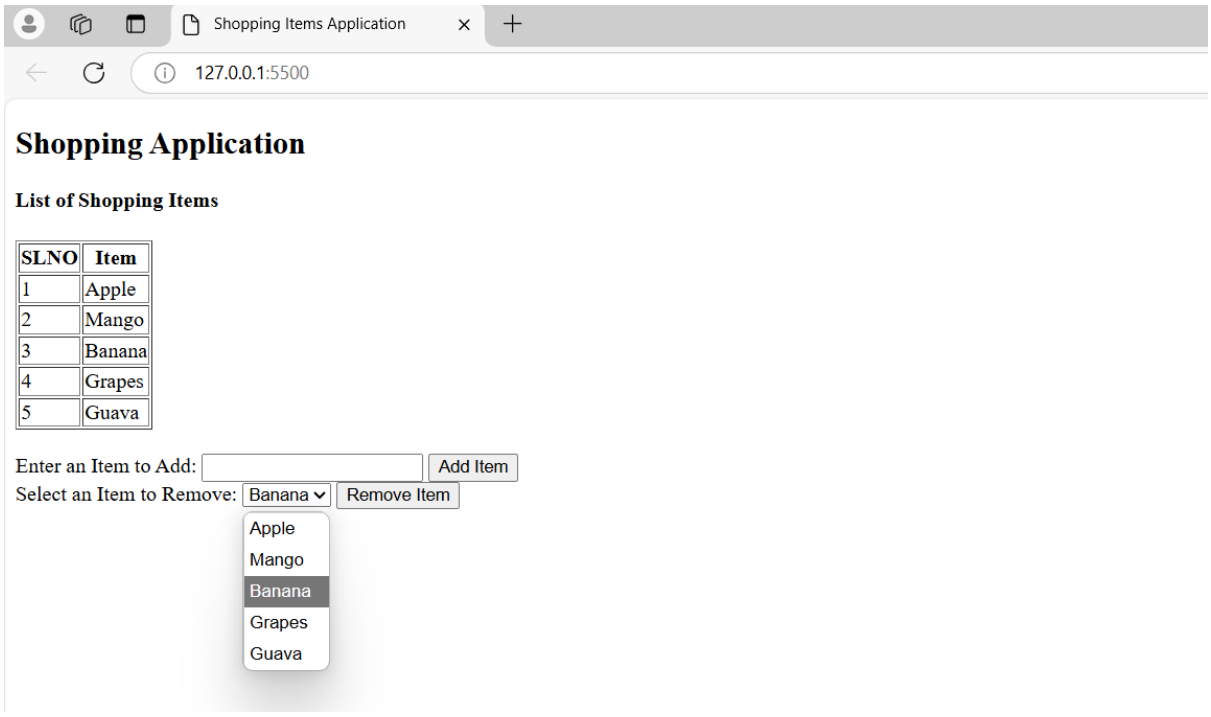
```
var app = angular.module("myApp", []);
app.controller("myCntrl", function($scope) {
    // Initial Shopping List Items
    $scope.shoppingItems = ['Apple', 'Mango', 'Banana', 'Grapes'];
 // Function to Add an Item
    $scope.addItem = function() {
        if ($scope.newItem && $scope.shoppingItems.indexOf($scope.newItem) === -1) {
            $scope.shoppingItems.push($scope.newItem);
            $scope.newItem = "";
        } else {
            if ($scope.newItem) {
                alert("This item is already in the shopping list.");
            } else {
                alert("Please enter an item to add.");
            }
        }
    };

    // Function to Remove an Item
    $scope.removeItem = function() {
        if ($scope.shoppingItems.indexOf($scope.selectItem) === -1) {
            alert("Please select an item to remove.");
        } else {
            var index = $scope.shoppingItems.indexOf($scope.selectItem);
            $scope.shoppingItems.splice(index, 1);
```

```
            $scope.selectItem = "";
        }
    };
});
```

Output:


Shopping Items Application
127.0.0.1:5500
Shopping Application
List of Shopping Items
SLNO
Item
1
Apple
2
Mango
3
Banana
4
Grapes
Enter an Item to Add: Guava
Add Item
Select an Item to Remove:
Remove Item


Shopping Items Application
127.0.0.1:5500
Shopping Application
List of Shopping Items
SLNO
Item
1
Apple
2
Mango
3
Banana
4
Grapes
5
Guava
Enter an Item to Add:
Add Item
Select an Item to Remove: Banana
Remove Item
Apple
Mango
Banana
Grapes
Guava

Result:

To Develop a Angular JS application that displays a list of shopping items and allow user to add and remove items from the list using directives and controllers is executed successfully.

Exp.NO:6 Date:	Write a program to create a simple calculator Application using React JS

Aim: To Write a program to create a simple calculator Application using React JS

ALGORITHM:

Step 1 : Set Up React Project

Step 2: Create Calculator Component

Step 3: Design UI in JSX

Step 4: Implement Logic in State

Step 5: Test and Run the App

Program:

Jsx file:

```
import React, { useState } from 'react';
import './Calculator.css';

const Calculator = () => {
  const [display, setDisplay] = useState('0');

  const handleNumber = (num) => {
    if (display === '0') {
      setDisplay(num);
    } else {
      setDisplay(display + num);
    }
  };

  const handleOperator = (op) => {
    const lastChar = display.slice(-1);
    const operators = ['+', '-', '*', '/'];

    if (operators.includes(lastChar)) {
      setDisplay(display.slice(0, -1) + op);
    } else {
      setDisplay(display + op);
    }
  };

  const handleDecimal = () => {
    const parts = display.split(/[+\-*/]/);
    const currentNumber = parts[parts.length - 1];

    if (!currentNumber.includes('.')) {
      setDisplay(display + '.');
```

```
  }
};

const calculate = () => {
  try {
    const result = eval(display);
    setDisplay(result.toString());
  } catch (error) {
    setDisplay('Error');
  }
};

const clear = () => {
  setDisplay('0');
};

return (
  <div className="calculator">
    <div className="display" data-testid="display">{display}</div>
    <div className="buttons">
      <button className="clear" onClick={clear}>C</button>
      <button onClick={() => handleOperator('/')}>/</button>
      <button onClick={() => handleOperator('*')}>*</button>
      <button onClick={() => handleNumber('7')}>7</button>
      <button onClick={() => handleNumber('8')}>8</button>
      <button onClick={() => handleNumber('9')}>9</button>
      <button onClick={() => handleOperator('-')}>-</button>
      <button onClick={() => handleNumber('4')}>4</button>
      <button onClick={() => handleNumber('5')}>5</button>
      <button onClick={() => handleNumber('6')}>6</button>
      <button onClick={() => handleOperator('+')}>+</button>
      <button onClick={() => handleNumber('1')}>1</button>
      <button onClick={() => handleNumber('2')}>2</button>
```

```
        <button onClick={() => handleNumber('3')}>3</button>
        <button className="equals" onClick={calculate}>=</button>
        <button className="zero" onClick={() => handleNumber('0')}>0</button>
        <button onClick={handleDecimal}>.</button>
      </div>
    </div>
  );
};
```

Css file:

```
.calculator {
  width: 320px;
  margin: 50px auto;
  padding: 20px;
  background-color: #f5f5f5;
  border-radius: 10px;
  box-shadow: 0 0 10px rgba(0,0,0,0.2);
}

.display {
  background-color: #fff;
  padding: 20px;
  margin-bottom: 20px;
  border-radius: 5px;
  text-align: right;
  font-size: 2em;
  min-height: 60px;
}

.buttons {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  gap: 10px;
```

```
}

button {
  padding: 20px;
  font-size: 1.2em;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  background-color: #e0e0e0;
  transition: background-color 0.3s;
}

button:hover {
  background-color: #d0d0d0;
}
.clear {
  background-color: #ff6666;
  color: white;
}

.equals {
  background-color: #4CAF50;
  color: white;
  grid-row: span 2;
}

.zero {
  grid-column: span 2;
}
```

Output:

Result:

To Write a program to create a simple calculator Application using React JS is executed successfully.

Exp.NO:7 Date:	Write a program to create a voting application using React JS.

Aim: To Write a program to create a voting application using React JS.

ALGORITHM:

Step 1 : Initialize Project

Step 2: Create Components

Step 3: Manage State

Step 4: Display Results

Step 5: Run & Test

Program:

JSX file:

```
import React, { useState } from 'react';


const VotingApp = () => {
  const [candidates, setCandidates] = useState([
    { id: 1, name: 'Candidate 1', votes: 0 },
    { id: 2, name: 'Candidate 2', votes: 0 },
    { id: 3, name: 'Candidate 3', votes: 0 },
  ]);

  const handleVote = (candidateId) => {
    setCandidates(prevCandidates =>
      prevCandidates.map(candidate =>
        candidate.id === candidateId
          ? { ...candidate, votes: candidate.votes + 1 }
          : candidate
      )
    );
  };

  const totalVotes = candidates.reduce((sum, candidate) => sum + candidate.votes, 0);

  return (
    <div style={styles.container}>
      <h1 style={styles.title}>Voting System</h1>
      <div style={styles.total}>Total Votes: {totalVotes}</div>

      <div style={styles.candidatesContainer}>
        {candidates.map(candidate => (
          <div key={candidate.id} style={styles.candidateCard}>
            <h3>{candidate.name}</h3>
            <p>Votes: {candidate.votes}</p>
```

```
            <button
              style={styles.button}
              onClick={() => handleVote(candidate.id)}
            >
              Vote
            </button>
          </div>
        ))}
      </div>
    </div>
  );
};
```

Css file:

```
const styles = {
  container: {
    maxWidth: '800px',
    margin: '0 auto',
    padding: '20px',
    fontFamily: 'Arial, sans-serif',
  },
  title: {
    textAlign: 'center',
    color: '#333',
  },
  total: {
    textAlign: 'center',
    fontSize: '1.2em',
    margin: '20px 0',
    fontWeight: 'bold',
  },
  candidatesContainer: {
    display: 'flex',
```

```
    justifyContent: 'space-around',
    flexWrap: 'wrap',
  },
  candidateCard: {
    border: '1px solid #ddd',
    borderRadius: '8px',
    padding: '20px',
    margin: '10px',
    width: '200px',
    textAlign: 'center',
    boxShadow: '0 2px 4px rgba(0,0,0,0.1)',
  },
  button: {
    backgroundColor: '#4CAF50',
    color: 'white',
    padding: '10px 20px',
    border: 'none',
    borderRadius: '4px',
    cursor: 'pointer',
    fontSize: '16px',
    marginTop: '10px',
  },
};
```

Output:


Voting System
Total Votes: 8
Candidate 1
Votes: 5
Vote
Candidate 2
Votes: 2
Vote
Candidate 3
Votes: 1
Vote

Result:

To Write a program to create a voting application using React JS successfully executed.

Exp.NO:8 Date:	Write a server side program for manipulating mongo DB from Node.js

Aim: To Write a server side program for manipulating mongo DB from Node.js

ALGORITHM:

Step 1 : Install Dependencies

Step 2: Setup Express Server

Step 3: Define a Mongoose Model

Step 4: Create Routes for CRUD Operations

Step 5: Integrate Routes in Server

Program:

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User Management</title>
    <script>
        const apiUrl = "http://localhost:5000/users";

        // Fetch and display users
        function fetchUsers() {
            fetch(apiUrl)
                .then(response => response.json())
                .then(users => {
                    const userList = document.getElementById("userList");
                    userList.innerHTML = "";
                    users.forEach(user => {
                        const li = document.createElement("li");
                        li.innerHTML = `${user.name} (${user.email}) - ${user.age} years
                            <button onclick="deleteUser('${user._id}')">Delete</button>`;
                        userList.appendChild(li);
                    });
                })
                .catch(error => console.error("Error fetching users:", error));
        }

        // Add a new user
        function addUser() {
            const name = document.getElementById("name").value;
            const email = document.getElementById("email").value;
            const age = document.getElementById("age").value;
```

```
            fetch(apiUrl, {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ name, email, age: Number(age) })
            })
            .then(response => response.json())
            .then(() => {
                fetchUsers(); // Refresh user list
                document.getElementById("name").value = "";
                document.getElementById("email").value = "";
                document.getElementById("age").value = "";
            })
            .catch(error => console.error("Error adding user:", error));
        }

        // Delete a user
        function deleteUser(id) {
            fetch(`${apiUrl}/${id}`, { method: "DELETE" })
                .then(() => fetchUsers())
                .catch(error => console.error("Error deleting user:", error));
        }

        document.addEventListener("DOMContentLoaded", fetchUsers);
    </script>
</head>
<body>
    <h2>User Management</h2>
    <div>
        <input type="text" id="name" placeholder="Name">
        <input type="email" id="email" placeholder="Email">
        <input type="number" id="age" placeholder="Age">
        <button onclick="addUser()">Add User</button>
    </div>
```

```
    <h3>Users:</h3>
    <ul id="userList"></ul>
</body>
</html>
```

Server.js:

```
require("dotenv").config();
const express = require("express");
const mongoose = require("mongoose");
const bodyParser = require("body-parser");
const cors = require("cors");

const app = express();

// Middleware
app.use(cors());
app.use(bodyParser.json());
// Connect to MongoDB
mongoose
  .connect(process.env.MONGO_URI, { useNewUrlParser: true, useUnifiedTopology: true , family: 4})
  .then(() => console.log("MongoDB Connected"))
  .catch((err) => console.error("MongoDB Connection Error:", err));
// Define a User Schema
const UserSchema = new mongoose.Schema({
  name: String,
  email: String,
  age: Number,
});
const User = mongoose.model("Schema", UserSchema);

// Routes
app.get("/", (req, res) => {
```

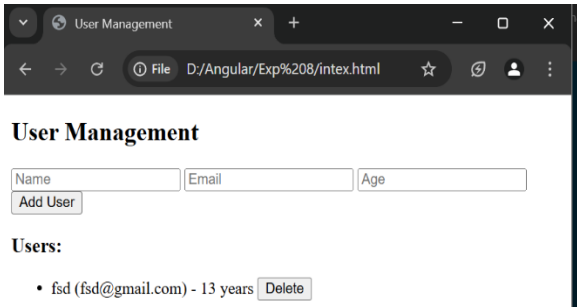
```
  res.send("Welcome to the MongoDB Node.js API!");
});
// Create a User
app.post("/users", async (req, res) => {
    try {
      const newUser = new User(req.body);
      const savedUser = await newUser.save();
      console.log("User saved:", savedUser);
      res.status(201).json(savedUser);
    } catch (error) {
      console.error("Error saving user:", error);
      res.status(400).json({ error: error.message });
    }
  });
// Get All Users
app.get("/users", async (req, res) => {
  const users = await User.find();
  res.json(users);
});
// Get a Single User
app.get("/users/:id", async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    if (!user) return res.status(404).json({ message: "User not found" });
    res.json(user);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});
// Update a User
app.put("/users/:id", async (req, res) => {
  try {
    const updatedUser = await User.findByIdAndUpdate(req.params.id, req.body, {
```

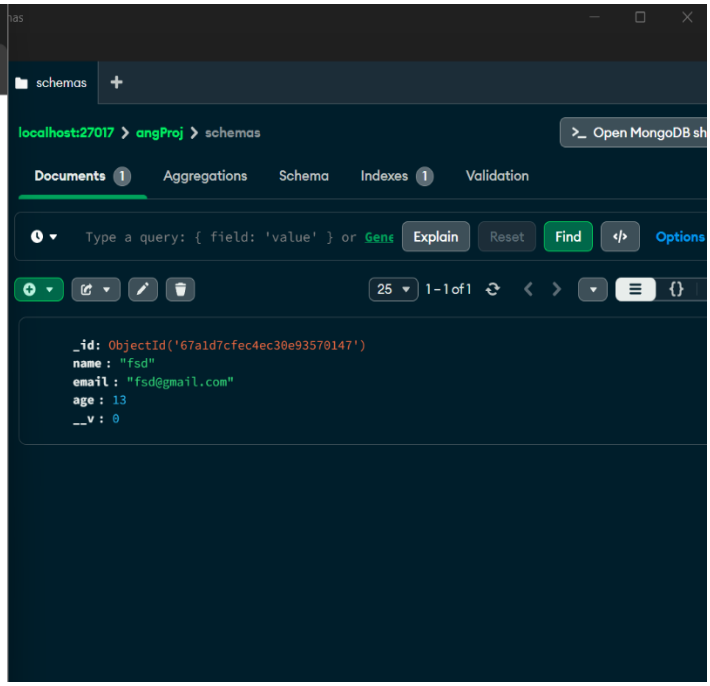
```
      new: true,
    });
    if (!updatedUser) return res.status(404).json({ message: "User not found" });
    res.json(updatedUser);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});
// Delete a User
app.delete("/users/:id", async (req, res) => {
  try {
    const deletedUser = await User.findByIdAndDelete(req.params.id);
    if (!deletedUser) return res.status(404).json({ message: "User not found" });
    res.json({ message: "User deleted successfully" });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});
// Start Server
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

.env file:

```
MONGO_URI="mongodb://localhost:27017/”
```

Output:


User Management
File D:/Angular/Exp%208/intex.html
User Management
Name
Email
Age
Add User
Users:
fsd (fsd@gmail.com) - 13 years Delete


schemas
localhost:27017 > angProj > schemas
Open MongoDB sh
Documents 1
Aggregations
Schema
Indexes 1
Validation
Type a query: { field: 'value' } or Gene
Explain
Reset
Find
Options
25
1 - 1 of 1
_id: ObjectId('67a1d7cfec4ec30e93570147')
name : "fsd"
email : "fsd@gmail.com"
age : 13
__v : 0

Result:

To Write a server side program for manipulating mongo DB from Node.js is executed successfully

Exp.NO:9 Date:	**Develop a project using Angular, Node. js, Express .js and Mongo DB from .This has to be CRUD Application for managing items(e.g.,a task manager or inventory app) and emphasis on integrating Security Practices.**

Aim: To Develop a project using Angular, Node.js, Express.js and Mongo DB from. This has to be CRUD Application for managing items(e.g., a task manager or inventory app) and emphasis on integrating Security Practices.

PROCEDURE:

Step 1 : Project Setup: Backend (Node.js, Express.js, MongoDB)

Step 2: Build the Backend (Node.js + Express.js)

Step 3: Build the Frontend (Angular).

Step 4: Testing & Deployment.

Step 5: Maintenance & Security Best Practices.

Result:

To Develop a project using Angular, Node. js, Express .js and Mongo DB from .This has to be CRUD Application for managing items(e.g.,a task manager or inventory app) and emphasis on integrating Security Practices is executed successfully.