SRM VALLIAMMAI ENGINEERINGCOLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203

DEPARTMENT OF INFORMATION TECHNOLOGY

ACADEMIC YEAR: 2025-2026

EVEN SEMESTER

LAB MANUAL

(REGULATION - 2023)

IT3663- FULL STACK DEVELOPMENT LAB

SIXTH SEMESTER

Prepared By

Ms. S. Shenbagavadivu, Assistant Professor (Sel.G)

IT3663 FULL STACK DEVELOPMENT LABORATORY L T P C

0 0 3 1.5

OBJECTIVES:

1. Usage of various front-end Tools and back-end Tools
2. They can understand and create applications on their own
3. Demonstrate and Designing of Websites can be carried out.
4. Develop web-based application using suitable client side and server-side code.
5. Implement web-based application using effective database access.

LIST OF EXPERIMENTS:

1. Write a program to create a simple webpage using HTML.
2. Write a program to create a website using HTML CSS and JavaScript.
3. Write a program to build a Chat module using HTML CSS and JavaScript.
4. Write a program to create a simple calculator Application using React JS.
5. Write a program to create a voting application using React JS.
6. Write a program to create and Build a Password Strength Check using Jquery.
7. Write a program to create and Build a star rating System using Jquery.
8. Create a simple Login form using React JS.
9. Create a blog using React JS.
10. Create a project on Grocery delivery application

Operating Systems: Windows, macOS, or Linux.

Browser: Google Chrome, Firefox, or any modern web browser.

Code Editor/IDE: VS Code (preferred) or any similar text editor with support for JavaScript and web development tools.

Node.js, MongoDB, Express.js (Backend Framework)

Angular CLI (Command Line Interface), npm or yarn (for package management), Heroku, Vercel, or Netlify for deploying web applications.

STOTAL: 45 PERIODS

OUTCOMES:

At the end of the course, the student should be able to:

1. Usage of various front and back end Tools.
2. They can understand and create applications on their own
3. Demonstrate and Designing of Websites can be carried out.
4. Develop web based application using suitable client side and server side code.
5. Implement web-based application using effective database access.

CO – PO – PSO Mapping

CO	PO											PSO		
	1	2	3	4	5	6	7	8	9	10	11	1	2	3
1	3	2	2	-	3	-	-	2	2	-	-	-	3	-
2	3	2	2	-	3	-	-	2	2	-	-	-	3	-
3	3	2	2	-	3	-	-	2	2	-	-	-	3	-
4	3	2	2	-	3	-	-	2	2	-	-	-	3	-
5	3	2	2	-	3	-	-	2	2	-	-	-	3	-
Avg	**3.0**	**2.0**	**2.0**	**-**	**3.0**	**-**	**-**	**2.0**	**2.0**	**-**	**-**	**-**	**3**	**-**

s

MODE OF ASSESSMENT

EVALUATION PROCEDURE FOR EACH EXPERIMENT

S.No	Description	Mark
1.	Aim & Pre-Lab discussion	30
2.	Observation	20
3.	Conduction and Execution	30
4.	Output & Result	10
5.	Viva	10
Total		**100**

INTERNAL ASSESSMENT FOR LABORATORY

S.No	Description	Mark
1.	Conduction & Execution of Experiment	30
2.	Record	10
3.	Model Test	20
Total		**60**

Ex. No: 1 — CREATE A SIMPLE WEBPAGE USING HTML

Aim: To design and develop a structured webpage using essential and advanced HTML tags.

Software Requirements:

- VS Code / Sublime Text / Atom
- Modern browser (Chrome / Firefox / Edge)

Description:

HTML (HyperText Markup Language) is the core technology for creating webpages.
In this experiment, students will build a webpage that includes:

- Page structure using semantic tags
- Text formatting elements
- Lists and tables
- Images and hyperlinks
- A simple form
- Organized content using <header>, <nav>, <section>, <footer>

Procedure:

1. Open VS Code and create a new folder (e.g., *Exp1_AdvancedHTML*).
2. Create a new file named **index.html**.
3. Add the standard HTML document structure.
4. Insert semantic tags and layout sections.
5. Add lists, images, hyperlinks, and a simple form.
6. Save and open the page in a browser.
7. Verify each section renders correctly.

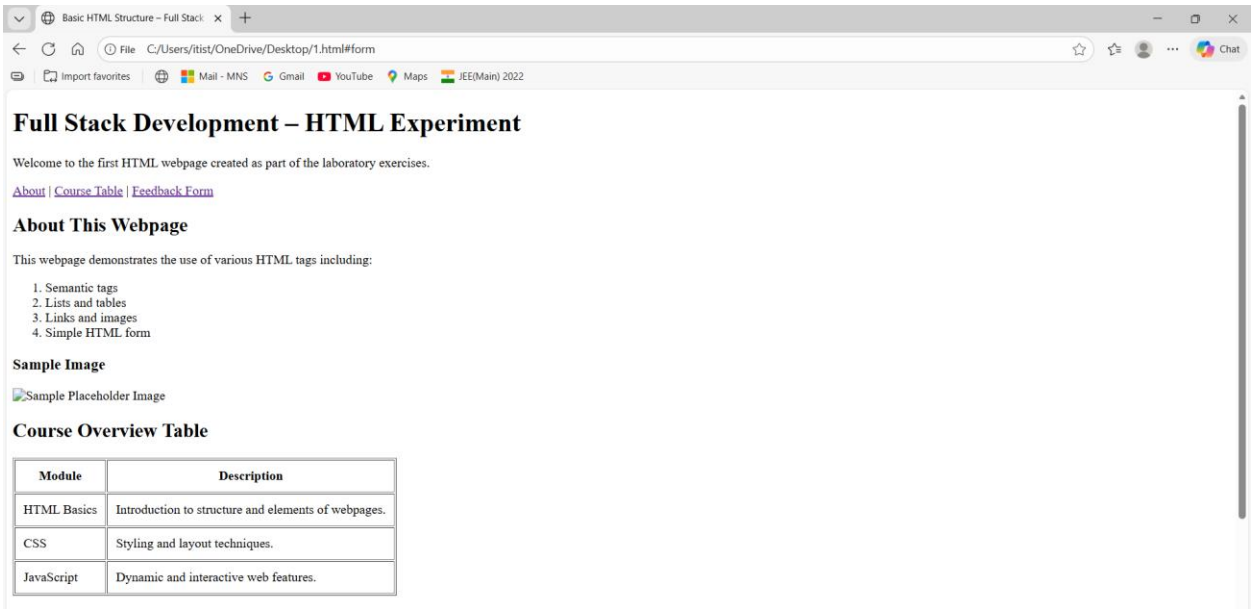
PROGRAM:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Basic HTML Structure – Full Stack Lab</title>
</head>

<body>
```

```
<header>
  <h1>Full Stack Development – HTML Experiment</h1>
  <p>Welcome to the first HTML webpage created as part of the laboratory exercises.</p>
</header>

<nav>
  <a href="#about">About</a> |
  <a href="#table">Course Table</a> |
  <a href="#form">Feedback Form</a>
</nav>

<section id="about">
  <h2>About This Webpage</h2>
  <p>This webpage demonstrates the use of various HTML tags including:</p>
  <ol>
    <li>Semantic tags</li>
    <li>Lists and tables</li>
    <li>Links and images</li>
    <li>Simple HTML form</li>
  </ol>

  <h3>Sample Image</h3>
  <img src="https://via.placeholder.com/250" alt="Sample Placeholder Image">
</section>

<section id="table">
  <h2>Course Overview Table</h2>
  <table border="1" cellpadding="10">
    <tr>
      <th>Module</th>
      <th>Description</th>
    </tr>
    <tr>
      <td>HTML Basics</td>
      <td>Introduction to structure and elements of webpages.</td>
    </tr>
    <tr>
      <td>CSS</td>
      <td>Styling and layout techniques.</td>
    </tr>
    <tr>
      <td>JavaScript</td>
      <td>Dynamic and interactive web features.</td>
    </tr>
  </table>
</section>
```

```
    <section id="form">
      <h2>Feedback Form</h2>
      <form>
        <label>Name:</label><br>
        <input type="text" placeholder="Enter your name"><br><br>

        <label>Email:</label><br>
        <input type="email" placeholder="Enter your email"><br><br>

        <label>Favourite Module:</label><br>
        <select>
          <option>HTML</option>
          <option>CSS</option>
          <option>JavaScript</option>
        </select><br><br>

        <input type="submit" value="Submit">
      </form>
    </section>

    <footer>
      <p>&copy; 2025 Full Stack Development Lab – Experiment 1</p>
    </footer>

</body>
</html>
```

OUTPUT:



Course Overview Table

Module	Description
HTML Basics	Introduction to structure and elements of webpages.
CSS	Styling and layout techniques.
JavaScript	Dynamic and interactive web features.

Feedback Form

Name:
Enter your name

Email:
Enter your email

Favourite Module:
HTML

Submit

© 2025 Full Stack Development Lab – Experiment 1

Result:

Thus, an advanced webpage using semantic elements, tables, lists, images, forms, and hyperlinks was designed and executed successfully.

Viva Questions:

1. What is the purpose of semantic tags like <header>, <section>, and <footer>?
2. Differentiate between ordered and unordered lists.
3. What attributes are commonly used with the <img> tag?
4. What is the use of the <form> tag?
5. Explain the difference between <table>, <tr>, <td>, and <th>.

Ex. No: 2 — CREATE A WEBSITE USING HTML, CSS, AND JAVASCRIPT

Aim:

To design and develop a responsive, interactive website using HTML for structure, CSS for layout and styling, and JavaScript for dynamic functionality.

Software Requirements:

- Visual Studio Code
- Chrome / Firefox Browse

Description:

This experiment focuses on building an advanced, structured website using front-end technologies. Students will learn:

- Page layout using **semantic HTML5 elements**
- Styling using **CSS Flexbox**, custom classes, button effects, and responsive design
- Dynamic behavior using **JavaScript DOM manipulation**
- Implementing features like:
    - Smooth scrolling navigation
    - Dark mode toggling
    - Interactive announcements section
    - Dynamic date rendering

Procedure:

1. Create a folder named **Exp2_AdvancedWebsite**.
2. Inside it, create three files:
    - index.html
    - styles.css
    - script.js
3. Design the structure using HTML semantic sections.
4. Add CSS Flexbox layout, color themes, and responsive rules.
5. Implement JavaScript functions for interactivity.
6. Launch index.html in a browser to view functionality.
7. Modify colors, fonts, and JS logic to understand customization.

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>Advanced FSD Website</title>
   <link rel="stylesheet" href="styles.css">
</head>

<body>

   <header>
      <h1>Full Stack Development – Mini Website</h1>

      <nav class="navbar">
         <a href="#home">Home</a>
         <a href="#services">Services</a>
         <a href="#gallery">Gallery</a>
         <a href="#contact">Contact</a>
         <button class="dark-btn" onclick="toggleDarkMode()">Toggle Dark Mode</button>
      </nav>
   </header>

   <section id="home" class="hero">
      <h2>Welcome to My Website</h2>
      <p id="dateDisplay"></p>
      <button class="action-btn" onclick="showAnnouncement()">Show
Announcement</button>
      <p id="announcementText" class="hidden"></p>
   </section>

   <section id="services" class="cards-section">
      <h2>Our Services</h2>

      <div class="card-container">
         <div class="card">
            <h3>Web Design</h3>
            <p>Creating modern, responsive interfaces using HTML, CSS & JS.</p>
         </div>

         <div class="card">
            <h3>Frontend Development</h3>
            <p>Interactive layouts using Vanilla JS, React, or Angular.</p>
```

```
        </div>

        <div class="card">
          <h3>Backend Development</h3>
          <p>Building APIs using Node.js, Express & MongoDB.</p>
        </div>
      </div>
    </section>

    <section id="gallery">
      <h2>Gallery</h2>
      <img src="https://via.placeholder.com/300" alt="Sample Image" class="gallery-img">
      <img src="https://via.placeholder.com/300" alt="Sample Image" class="gallery-img">
      <img src="https://via.placeholder.com/300" alt="Sample Image" class="gallery-img">
    </section>

    <section id="contact" class="contact-section">
      <h2>Contact Us</h2>

      <form class="contact-form">
        <label>Name:</label>
        <input type="text" placeholder="Enter your name">

        <label>Email:</label>
        <input type="email" placeholder="Enter your email">

        <label>Message:</label>
        <textarea placeholder="Enter your message"></textarea>

        <button type="submit">Submit</button>
      </form>
    </section>

    <footer>
      <p>&copy; <span id="year"></span> Full Stack Development Lab – Experiment 2</p>
    </footer>

    <script src="script.js"></script>
  </body>
  </html>
```

styles.css:

```
body {
    margin: 0;
    font-family: Arial, sans-serif;
    transition: background 0.3s, color 0.3s;
```

```
}

header {
    background: #1976d2;
    color: white;
    padding: 20px;
    text-align: center;
}

.navbar a {
    margin: 0 10px;
    color: white;
    text-decoration: none;
}

.dark-btn {
    margin-left: 20px;
    padding: 8px 12px;
    cursor: pointer;
}

.hero {
    padding: 40px;
    text-align: center;
}

.action-btn {
    background: #1976d2;
    color: white;
    border: none;
    padding: 12px 15px;
    cursor: pointer;
    transition: 0.3s;
}

.action-btn:hover {
    background: #0d47a1;
}

.cards-section {
    padding: 40px;
}

.card-container {
    display: flex;
    gap: 20px;
```

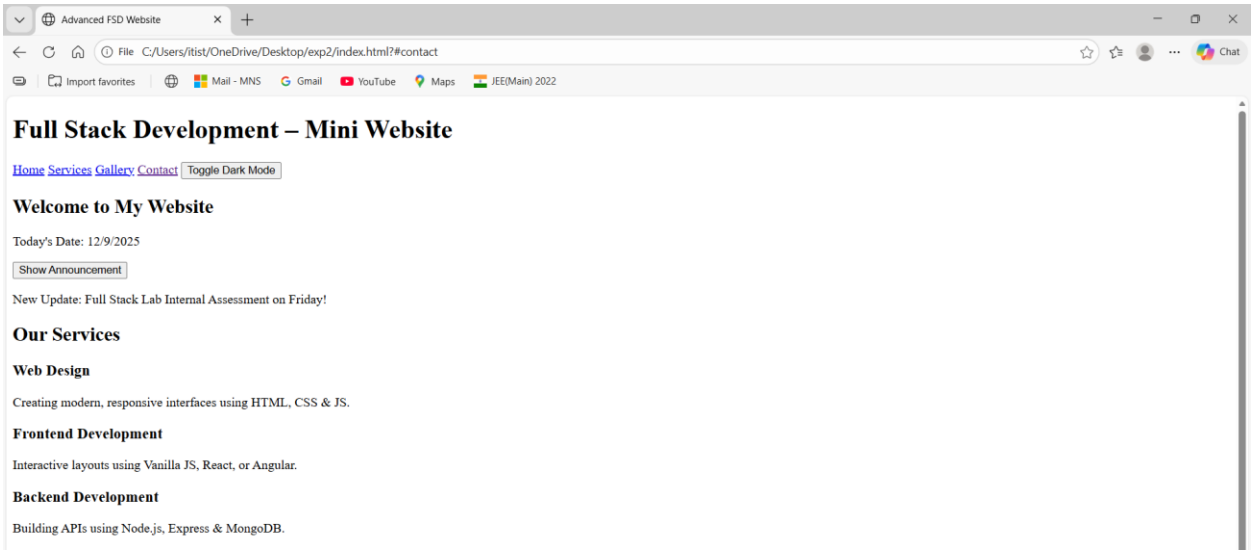
```
    justify-content: space-around;
}

.card {
    width: 30%;
    padding: 20px;
    background: #e3f2fd;
    border-radius: 10px;
    box-shadow: 0 2px 5px gray;
    text-align: center;
}

.gallery-img {
    width: 300px;
    margin: 10px;
    border-radius: 10px;
}

.contact-section {
    padding: 40px;
}

.contact-form input,
textarea {
    width: 100%;
    padding: 10px;
    margin: 10px 0;
}

footer {
    background: #eeeeee;
    text-align: center;
    padding: 20px;
}

.hidden {
    display: none;
}

/* Dark Mode */
.dark-mode {
    background: #1a1a1a;
    color: white;
}

.dark-mode header {
```

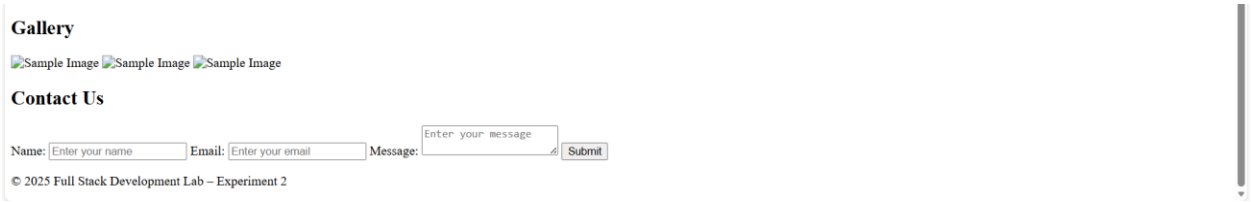
```
    background: #0d47a1;
}

.dark-mode .card {
    background: #424242;
}
```

Script.js:

```
function toggleDarkMode() {
    document.body.classList.toggle("dark-mode");
}

function showAnnouncement() {
    const msg = "New Update: Full Stack Lab Internal Assessment on Friday!";
    const announcement = document.getElementById("announcementText");
    announcement.textContent = msg;
    announcement.classList.remove("hidden");
}

document.getElementById("dateDisplay").textContent =
    "Today's Date: " + new Date().toLocaleDateString();

document.getElementById("year").textContent =
    new Date().getFullYear();
```

OUTPUT:





Result:

Thus, a fully functional, responsive and interactive website using HTML, CSS and JavaScript was successfully designed and executed.

Viva Questions:

1. What is the difference between inline, internal and external CSS?
2. Explain Flexbox and its advantages.
3. How does JavaScript interact with HTML through the DOM?
4. What is the purpose of event handlers such as onclick?
5. How does dark mode work using CSS class toggling?

Ex. No: 3 — BUILD A CHAT MODULE USING HTML, CSS, AND JAVASCRIPT

Aim:

To design and implement a centered, professional chat module using HTML, CSS, and JavaScript.

Software Requirements:

- Visual Studio Code
- Chrome / Firefox Browser

Description:

This experiment demonstrates building a polished and centered chat module UI with:

- Centered chat container
- Clean, symmetric design
- JavaScript-powered message generation
- Scrollable conversation area
- Auto-reply simulation
- Message timestamps

Procedure:

1. Create a project folder named **Exp3_ChatModule**.
2. Inside it create:
    - index.html
    - styles.css
    - script.js
3. Build the layout: chat window, message area, input box, and send button.
4. Apply CSS to style chat bubbles, align messages, and format UI.
5. Implement JavaScript:
    - Add messages to chat window
    - Generate timestamps
    - Auto-scroll behavior
    - Simple bot reply after a delay
6. Open index.html in a browser and test messaging.

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>Chat Module – FSD Lab</title>
   <link rel="stylesheet" href="styles.css">
</head>

<body>

   <div class="chat-container">

      <div class="chat-header">
         <h2>FSD Chat Module</h2>
      </div>

      <div class="chat-window" id="chatWindow"></div>

      <div class="chat-input-area">
         <input type="text" id="messageInput" placeholder="Type your message...">
         <button onclick="sendMessage()">Send</button>
      </div>

   </div>

   <script src="script.js"></script>
</body>
</html>
```

Style.css

```
body {
   margin: 0;
   background: #f0f2f5;
   font-family: Arial, sans-serif;
   display: flex;
   justify-content: center;
   padding-top: 40px;
}

.chat-container {
   width: 380px;
   height: 550px;
```

```
    background: white;
    box-shadow: 0 3px 10px gray;
    border-radius: 10px;
    display: flex;
    flex-direction: column;
}

.chat-header {
    background: #1976d2;
    color: white;
    padding: 15px;
    border-radius: 10px 10px 0 0;
}

.chat-window {
    flex: 1;
    padding: 15px;
    overflow-y: auto;
    background: #e9ebee;
}

.message {
    max-width: 70%;
    padding: 10px;
    margin-bottom: 10px;
    border-radius: 8px;
}

.sent {
    background: #1976d2;
    color: white;
    margin-left: auto;
}

.received {
    background: #ffffff;
    color: black;
}

.timestamp {
    font-size: 10px;
    opacity: 0.7;
    margin-top: 3px;
}

.chat-input-area {
```

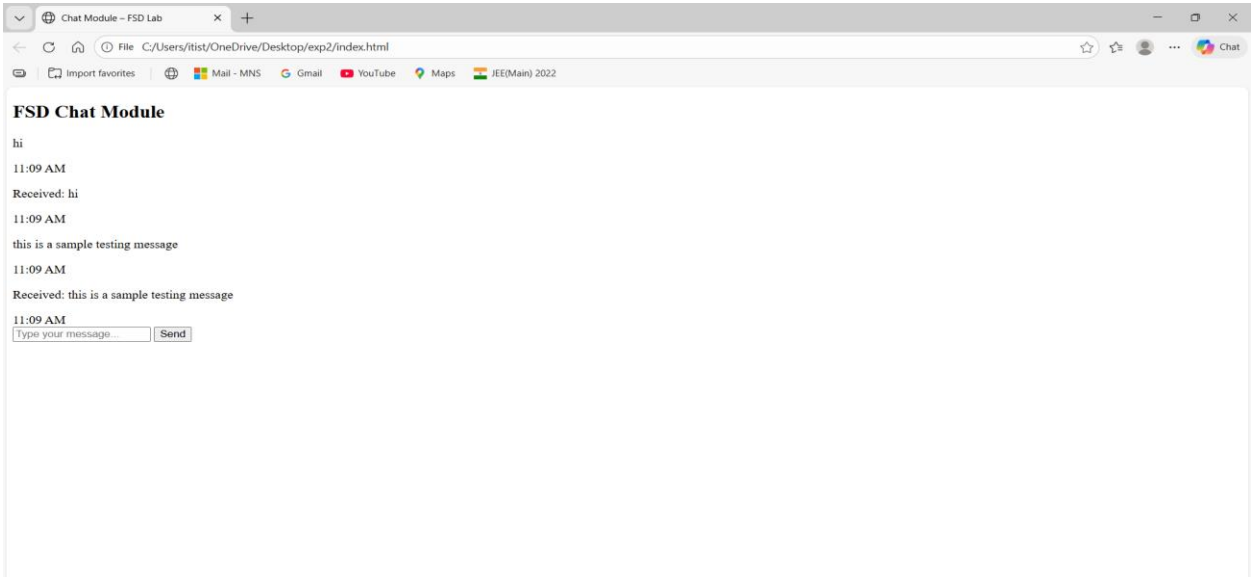
```
    display: flex;
    border-top: 1px solid #cccccc;
}

.chat-input-area input {
    flex: 1;
    padding: 15px;
    border: none;
}

.chat-input-area button {
    padding: 15px 20px;
    border: none;
    background: #1976d2;
    color: white;
    cursor: pointer;
}

.chat-input-area button:hover {
    background: #0d47a1;
}
```

Script.js

```
function sendMessage() {
    const input = document.getElementById("messageInput");
    const text = input.value.trim();

    if (text === "") return;

    appendMessage(text, "sent");

    input.value = "";
    autoScroll();

    setTimeout(() => {
        appendMessage("Received: " + text, "received");
        autoScroll();
    }, 600);
}

function appendMessage(text, type) {
    const chatWindow = document.getElementById("chatWindow");

    const messageDiv = document.createElement("div");
    messageDiv.classList.add("message", type);
```

```
    const time = new Date().toLocaleTimeString([], { hour: "2-digit", minute: "2-digit" });

    messageDiv.innerHTML = `
        <p>${text}</p>
        <div class="timestamp">${time}</div>
    `;

    chatWindow.appendChild(messageDiv);
}

function autoScroll() {
    const chatWindow = document.getElementById("chatWindow");
    chatWindow.scrollTop = chatWindow.scrollHeight;
}
```

OUTPUT:



Result:

Thus, an interactive chat module using HTML, CSS, and JavaScript was successfully designed and implemented.

Viva Questions:

1. What is DOM manipulation and how is it used in this experiment?
2. How does JavaScript dynamically create and append new elements?
3. What is the purpose of setTimeout() in this module?
4. Explain how message alignment is controlled using CSS classes.
5. Why is scrollTop updated after adding each message?

Ex. No: 4 — CREATE A SIMPLE CALCULATOR APPLICATION USING REACT JS

Aim:

To design and implement a responsive calculator application using React JS with functional components and state management.

Software Requirements:

- Node.js & npm
- React (via create-react-app)
- VS Code
- Modern browser

Description:

React JS is a component-based library used for building dynamic user interfaces.
In this experiment, students will learn to:

- Create a React project using **create-react-app**
- Use **functional components**
- Handle states using the **useState hook**
- Create reusable button components
- Build a calculator UI with CSS Grid
- Update the display dynamically based on user input
- Handle operations like addition, subtraction, multiplication, and division

This experiment introduces key concepts of React: state, props, event handling, component UI design, and controlled user input.

Procedure:

1. Open VS Code → Terminal → Run: npx create-react-app calculator-app
2. Navigate into the folder: cd calculator-app
3. Open the project in VS Code.
4. Replace the code inside App.js and App.css with the calculator implementation.
5. Start the application: npm start
6. Test each button: digits, operators, clear, delete, equals.
7. Validate functionality and UI responsiveness.

PROGRAM:

App.js

```
import React, { useState } from "react";
import "./App.css";

function App() {
  const [expression, setExpression] = useState("");

  const handleClick = (value) => {
    setExpression(expression + value);
  };

  const calculate = () => {
    try {
      // Using eval just for lab-level demonstration
      const result = eval(expression);
      setExpression(String(result));
    } catch {
      setExpression("Error");
    }
  };

  const clear = () => setExpression("");

  const deleteChar = () => setExpression(expression.slice(0, -1));

  return (
    <div className="calculator-container">
      <h2>React Calculator</h2>

      <input
        type="text"
        className="display"
        value={expression}
        readOnly
      />

      <div className="button-grid">
        <button onClick={clear}>AC</button>
        <button onClick={deleteChar}>DEL</button>
        <button onClick={() => handleClick("%")}>%</button>
        <button onClick={() => handleClick("/")}>/</button>

        <button onClick={() => handleClick("7")}>7</button>
        <button onClick={() => handleClick("8")}>8</button>
        <button onClick={() => handleClick("9")}>9</button>
        <button onClick={() => handleClick("*")}>*</button>
```

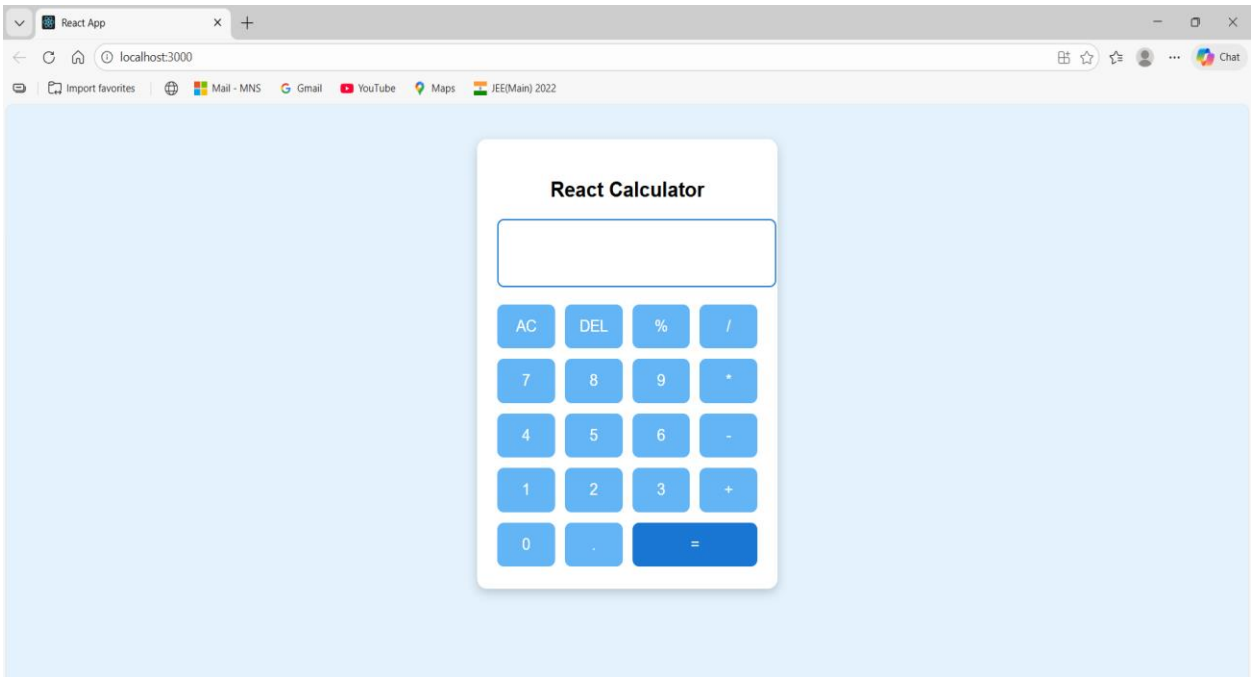
```
        <button onClick={() => handleClick("4")}>4</button>
        <button onClick={() => handleClick("5")}>5</button>
        <button onClick={() => handleClick("6")}>6</button>
        <button onClick={() => handleClick("-")}>-</button>

        <button onClick={() => handleClick("1")}>1</button>
        <button onClick={() => handleClick("2")}>2</button>
        <button onClick={() => handleClick("3")}>3</button>
        <button onClick={() => handleClick("+")}>+</button>

          <button onClick={() => handleClick("0")}>0</button>
        <button onClick={() => handleClick(".")}>.</button>
        <button className="equal-btn" onClick={calculate}>=</button>
      </div>
    </div>
  );
}
export default App;
```

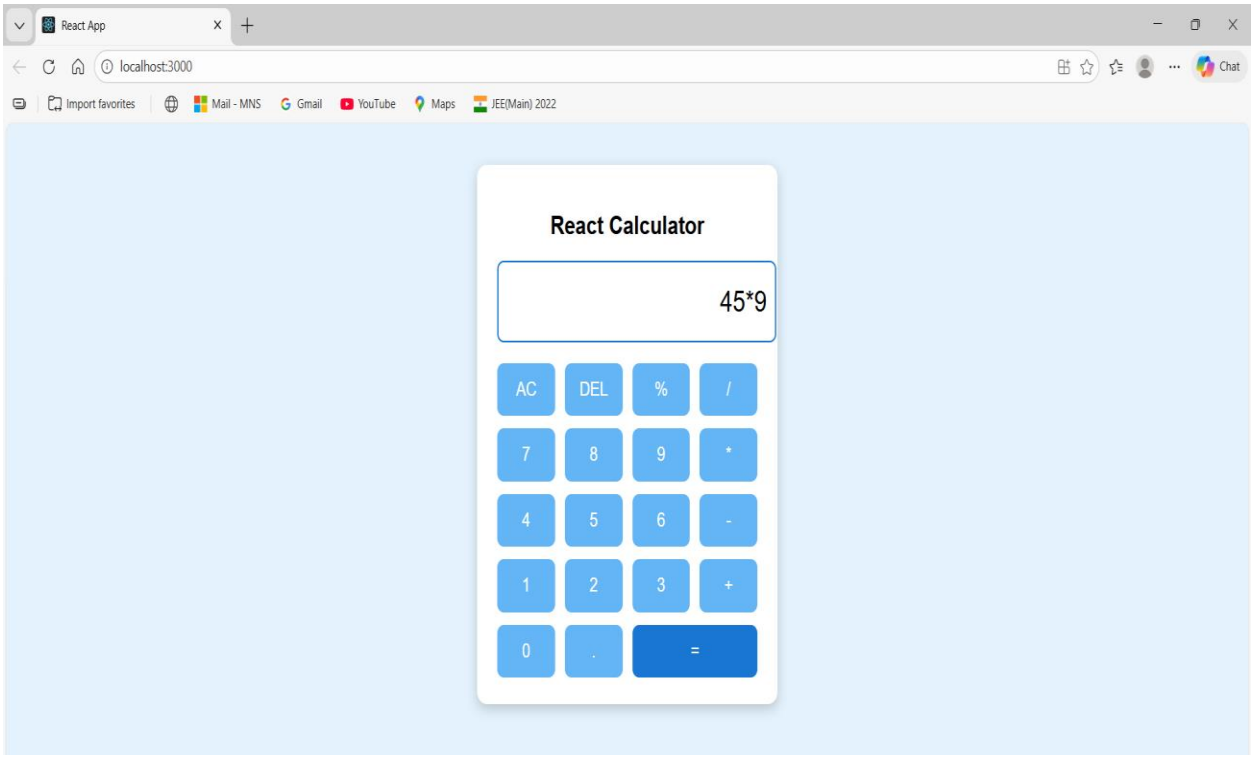
App.css

```
body {
  background: #e3f2fd;
  font-family: Arial, sans-serif;
  display: flex;
  justify-content: center;
  padding-top: 40px;
}

.calculator-container {
  background: white;
  padding: 25px;
  width: 320px;
  border-radius: 12px;
  box-shadow: 0 4px 12px rgba(0,0,0,0.2);
  text-align: center;
}

.display {
  width: 100%;
  height: 55px;
  font-size: 28px;
  margin-bottom: 20px;
  padding: 10px;
```

```
  text-align: right;
  border: 2px solid #1976d2;
  border-radius: 8px;
}

.button-grid {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  gap: 12px;
}

button {
  padding: 15px;
  font-size: 18px;
  border: none;
  border-radius: 8px;
  background: #64b5f6;
  color: white;
  cursor: pointer;
  transition: 0.2s;
}

button:hover {
  background: #42a5f5;
}

.equal-btn {
  background: #1976d2;
  grid-column: span 2;
}

.equal-btn:hover {
  background: #0d47a1;
}
```

OUTPUT:





Result:
Thus, a fully functional calculator application was developed using React JS.

Viva Questions:

1. What is the purpose of React's useState hook?
2. Why is React preferred for building UI components?
3. How does React handle re-rendering on state updates?
4. What are controlled components in React?
5. Why is eval() not recommended in production applications?

Ex. No: 5 — CREATE A VOTING APPLICATION USING REACT JS

Aim:

To develop a dynamic voting application using React JS that allows users to vote for candidates, update vote counts in real-time, and manage state effectively using React Hooks.

Software Requirements:

- Node.js & npm
- React (via create-react-app)
- VS Code
- Chrome / Firefox browser

Description:

This experiment teaches students to build an interactive React application using:

- **Functional components**
- **useState Hook for state management**
- **Props for reusable components**
- **Event handling**
- **Dynamic rendering of UI elements**

The application features:

- A list of candidates displayed as cards
- Vote and unvote buttons
- Live vote count updates
- Prevention of negative votes
- Responsive and user-friendly UI

Procedure:

1. Create a new React project: npx create-react-app voting-app
2. Navigate into the project folder: cd voting-app
3. Open the folder in VS Code.
4. Modify App.js and App.css with the provided code.
5. Start the application: npm start
6. Test the "Vote" and "Unvote" buttons for each candidate.

7. Validate UI responsiveness and component behavior.

PROGRAM:

App.js

```
import React, { useState } from "react";
import "./App.css";

function App() {
  const [candidates, setCandidates] = useState([
    { id: 1, name: "Alice", votes: 0 },
    { id: 2, name: "Bob", votes: 0 },
    { id: 3, name: "Charlie", votes: 0 }
  ]);

  const vote = (id) => {
    setCandidates(
      candidates.map((c) =>
        c.id === id ? { ...c, votes: c.votes + 1 } : c
      )
    );
  };

  const unvote = (id) => {
    setCandidates(
      candidates.map((c) =>
        c.id === id && c.votes > 0
          ? { ...c, votes: c.votes - 1 }
          : c
      )
    );
  };

  return (
    <div className="container">
      <h2>React Voting Application</h2>

      <div className="card-container">
        {candidates.map((candidate) => (
          <div key={candidate.id} className="card">
            <h3>{candidate.name}</h3>
            <p>Votes: {candidate.votes}</p>

            <div className="button-row">
              <button
                className="vote-btn"
                onClick={() => vote(candidate.id)}
```

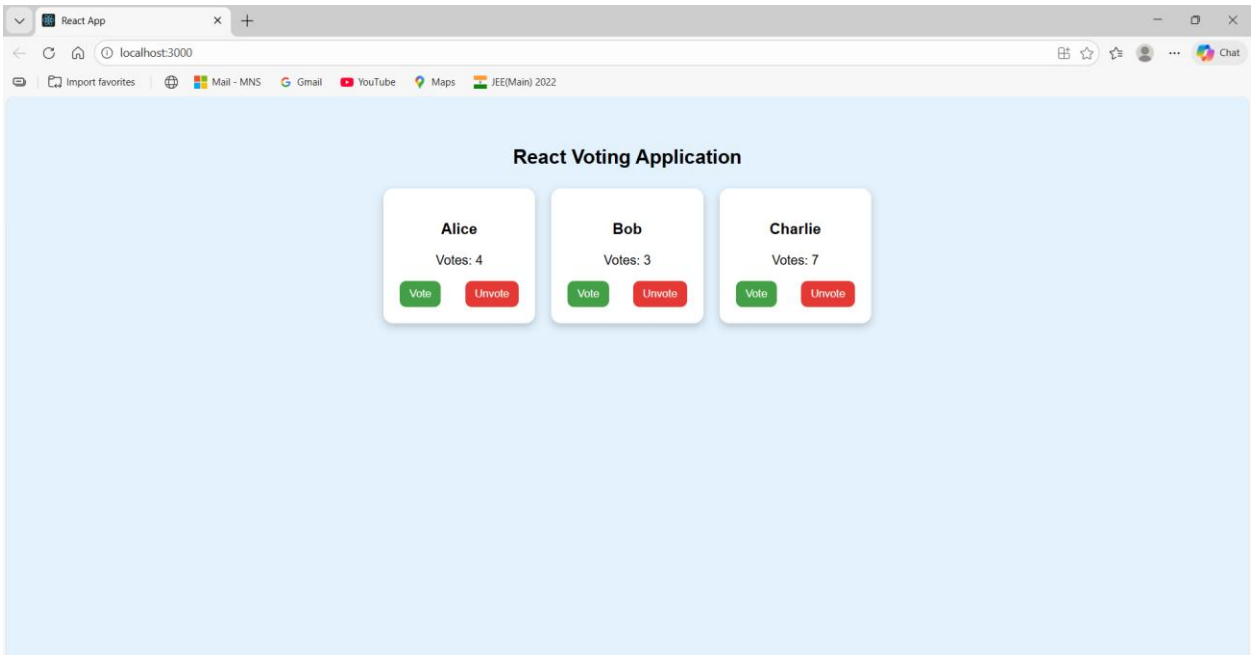
```
            >
              Vote
            </button>

            <button
              className="unvote-btn"
              onClick={() => unvote(candidate.id)}
            >
              Unvote
            </button>
          </div>
        </div>
      ))}
    </div>
  </div>
 );
}

export default App;
```

App.css:

```
body {
  background: #e3f2fd;
  font-family: Arial, sans-serif;
  display: flex;
  justify-content: center;
  padding-top: 40px;
}

.container {
  text-align: center;
  width: 600px;
}

.card-container {
  display: flex;
  justify-content: center;
  gap: 20px;
  margin-top: 25px;
}

.card {
  background: white;
  padding: 20px;
  width: 160px;
  border-radius: 12px;
  box-shadow: 0 4px 10px rgba(0,0,0,0.2);
```

```
}

.button-row {
  display: flex;
  justify-content: space-between;
  margin-top: 10px;
}

.vote-btn {
  background: #43a047;
  color: white;
  padding: 8px 12px;
  border-radius: 8px;
  border: none;
  cursor: pointer;
}

.vote-btn:hover {
  background: #2e7d32;
}

.unvote-btn {
  background: #e53935;
  color: white;
  padding: 8px 12px;
  border-radius: 8px;
  border: none;
  cursor: pointer;
}

.unvote-btn:hover {
  background: #b71c1c;
}
```

OUTPUT:



Result:

Thus, an interactive voting application using React JS was successfully designed and implemented.

Viva Questions:

1. What is the use of the useState Hook in React?
2. Why do we use the spread operator (...) in React state updates?
3. What is the purpose of mapping through arrays in React?
4. What is a key prop, and why is it needed in list rendering?
5. How does React re-render components after state changes?

Ex. No: 6 — BUILD A PASSWORD STRENGTH CHECK USING JQUERY

Aim:

To design and develop a webpage that checks the **strength of a password** using **jQuery**, based on predefined validation rules.

Software Requirements:

• VS Code / Sublime Text / Atom
• Modern browser (Chrome / Firefox / Edge)
• jQuery Library

Description:

jQuery is a fast and lightweight JavaScript library that simplifies **HTML DOM manipulation, event handling, and validation**.
In this experiment, a password strength checker is implemented to help users understand whether their password is **weak, medium, or strong**.

The strength is evaluated based on:
• Length of the password
• Use of uppercase letters
• Use of lowercase letters
• Presence of numbers
• Presence of special characters

The strength is displayed dynamically using jQuery as the user types the password.

Procedure:

1. Open VS Code and create a new folder (e.g., Exp6_PasswordStrength_jQuery).
2. Create a new file named index.html.
3. Add the basic HTML document structure.
4. Include the jQuery CDN inside the <head> section.
5. Create an input field for password entry.
6. Write jQuery code to check password strength on key press.
7. Display the strength result dynamically.
8. Save the file and open it in a browser.
9. Verify that password strength updates correctly.

PROGRAM:

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Password Strength Checker – jQuery</title>

    <!-- External CSS -->
    <link rel="stylesheet" href="styles.css">

    <!-- jQuery CDN -->
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

    <!-- External JS -->
    <script src="script.js" defer></script>
</head>

<body>

    <h2>Password Strength Checker</h2>

    <label>Enter Password:</label><br><br>
    <input type="password" id="password" placeholder="Type your password">

    <div id="strength"></div>

</body>
</html>
```

styles.css

```
body {
    font-family: Arial, sans-serif;
    margin: 40px;
}

input {
    padding: 8px;
    width: 250px;
    font-size: 14px;
}

#strength {
    margin-top: 10px;
    font-weight: bold;
}
```

script.js

```
$(document).ready(function () {

    $("#password").keyup(function () {

        var password = $(this).val();
        var strengthText = "";

        if (password.length < 6) {
            strengthText = "Weak Password";
            $("#strength").css("color", "red");
        }
        else if (password.match(/[A-Z]/) &&
```

```
            password.match(/[0-9]/) &&
            password.match(/[@$!%*?&]/)) {

        strengthText = "Strong Password";
        $("#strength").css("color", "green");
      }
      else {
        strengthText = "Medium Password";
        $("#strength").css("color", "orange");
      }

      $("#strength").text(strengthText);
    });
});
```

OUTPUT:

- When a **short password** is entered → *Weak Password*
- When password contains **letters and numbers** → *Medium Password*
- When password contains **uppercase, number, and special character** → *Strong Password*

Result:

Thus, a **Password Strength Checker** using **jQuery** was successfully designed and implemented. The program dynamically evaluates the password strength and displays appropriate feedback to the user.

Viva Questions:

1. What is jQuery and why is it used?
2. What is the purpose of $(document).ready() in jQuery?
3. How does jQuery handle events like keyup()?
4. What are regular expressions and how are they used here?
5. Differentiate between JavaScript and jQuery.

Ex. No: 7 — BUILD A STAR RATING SYSTEM USING JQUERY

Aim:

To design and develop an interactive **Star Rating System** using **jQuery**, where users can rate an item by selecting stars and view the selected rating dynamically.

Software Requirements:

- VS Code / Sublime Text / Atom
- Modern browser (Chrome / Firefox / Edge)
- jQuery Library

Description:

A **Star Rating System** is a commonly used UI component in modern web applications such as product reviews, movie ratings, and feedback systems.
Using **jQuery**, the process of handling mouse events and updating the UI dynamically becomes simple and efficient.

In this experiment, a star rating system is implemented where:

- Users can hover over stars to preview rating
- Users can click to select a rating
- The selected rating value is displayed dynamically

The system improves **user interaction** and provides **instant visual feedback**.

Procedure:

1. Open VS Code and create a new folder (e.g., Exp7_StarRating_jQuery).
2. Create three files: index.html, styles.css, and script.js.
3. Add the basic HTML document structure in index.html.
4. Include the jQuery CDN inside the <head> section.
5. Create star icons using HTML elements.
6. Apply styling using CSS to represent stars.
7. Write jQuery code to handle hover and click events.
8. Display the selected rating dynamically.
9. Save all files and open index.html in a browser.

10. Verify that star rating works correctly.

PROGRAM:

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Star Rating System – jQuery</title>

  <!-- External CSS -->
  <link rel="stylesheet" href="styles.css">

  <!-- jQuery CDN -->
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

  <!-- External JS -->
  <script src="script.js" defer></script>
</head>

<body>

  <h2>Star Rating System</h2>
```

```
    <div class="stars">
      <span class="star" data-value="1">★</span>
      <span class="star" data-value="2">★</span>
      <span class="star" data-value="3">★</span>
      <span class="star" data-value="4">★</span>
      <span class="star" data-value="5">★</span>
    </div>

    <p id="rating">Your Rating: 0</p>

  </body>
  </html>
```

styles.css

```
  body {
    font-family: Arial, sans-serif;
    margin: 40px;
  }

  .stars {
    font-size: 35px;
    cursor: pointer;
  }
```

```
.star {
    color: lightgray;
}

.star.selected,
.star.hover {
    color: gold;
}

#rating {
    margin-top: 10px;
    font-weight: bold;
}
```

script.js

```
$(document).ready(function () {
    var selectedRating = 0;
    $(".star").hover(
        function () {
            var value = $(this).data("value");
            $(".star").each(function () {
                if ($(this).data("value") <= value) {
                    $(this).addClass("hover");
                }
            });
        },
        function () {
            $(".star").removeClass("hover");
        }
    );
```

```
    $(".star").click(function () {
        selectedRating = $(this).data("value");
        $(".star").removeClass("selected");
        $(".star").each(function () {
            if ($(this).data("value") <= selectedRating) {
                $(this).addClass("selected");
            }
        });
        $("#rating").text("Your Rating: " + selectedRating);
    });

});
```

OUTPUT:

• Hovering over stars highlights them
• Clicking on stars selects the rating
• Selected rating value is displayed dynamically

Result:

Thus, a **Star Rating System using jQuery** was successfully designed and implemented. The system allows users to interactively select a rating and provides instant visual feedback using dynamic DOM manipulation.

Viva Questions:

1. What is the purpose of jQuery in this experiment?
2. How does the hover event improve user interaction?
3. What is the use of data-value attributes?
4. Explain the difference between hover and click events.
5. How can this rating system be used in real-time applications?

Ex. No: 8 — CREATE A SIMPLE LOGIN FORM USING REACT JS

Aim:

To design and develop a **simple login form** using **React JS** that accepts user credentials and validates input using component state.

Software Requirements:

• VS Code
• Node.js and npm
• Modern browser (Chrome / Edge / Firefox)
• React JS

Description:

React JS is a JavaScript library used for building **component-based user interfaces**. In this experiment, a **Login Form** is created using React functional components and **useState hook**.

The form includes:
• Username input
• Password input
• Submit button
• Basic validation

React manages form data dynamically without reloading the page.

Procedure:

1. Install Node.js and npm.
2. Create a React application using create-react-app.
3. Open the project in VS Code.
4. Create a functional component for Login.
5. Use useState to manage input values.
6. Handle form submission.
7. Display login success message.
8. Run the application using npm start.
9. Verify login form functionality.

PROGRAM:

App.js

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

function App() {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    alert("Login Successful");
  };

  return (
    <div style={{ margin: "40px" }}>
      <h2>Login Form</h2>

      <form onSubmit={handleSubmit}>
        <label>Username:</label><br />
        <input
          type="text"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
        /><br /><br />

        <label>Password:</label><br />
        <input
          type="password"
          value={password}
```

```
                onChange={(e) => setPassword(e.target.value)}
            /><br /><br />

            <button type="submit">Login</button>
        </form>
    </div>
  );
}

export default App;
```

OUTPUT:

- Login form is displayed
- User enters username and password
- On submit, login success alert appears

Result:

Thus, a **simple login form using React JS** was successfully designed and implemented using functional components and state management.

Viva Questions:

1. What is React JS?
2. What is a component in React?
3. What is the use of useState hook?
4. How does React handle form submission?
5. Difference between controlled and uncontrolled components?

Ex. No: 9 — CREATE A BLOG USING REACT JS

Aim:

To create a **simple blog application** using **React JS** that displays blog posts dynamically using components.

Software Requirements:

• VS Code
• Node.js and npm
• Modern browser
• React JS

Description:

A **Blog Application** displays articles or posts in an organized format.
React allows breaking UI into reusable components such as:
• Blog header
• Blog post
• Blog content

In this experiment, blog posts are displayed using **props and components**.

Procedure:

1. Create a React application using create-react-app.
2. Open project in VS Code.
3. Create a Blog component.
4. Use props to pass blog data.
5. Render multiple blog posts.
6. Run the application.
7. Verify blog content display.

PROGRAM:

App.js

```
import React from "react";
```

```
function BlogPost(props) {
  return (
    <div style={{ border: "1px solid #ccc", padding: "10px", margin: "10px" }}>
      <h3>{props.title}</h3>
      <p>{props.content}</p>
    </div>
  );
}

function App() {
  return (
    <div style={{ margin: "40px" }}>
      <h2>My Blog</h2>

      <BlogPost
        title="React Basics"
        content="React is a JavaScript library for building UI components."
      />

      <BlogPost
        title="Why React?"
        content="React makes UI development simple and efficient."
      />
    </div>
  );
}

export default App;
```

OUTPUT:

• Blog page is displayed
• Multiple blog posts appear
• Content is rendered dynamically

Result:

Thus, a **blog application using React JS** was successfully created using reusable components and props.

Viva Questions:

1. What are props in React?
2. Difference between props and state?
3. What is component reusability?
4. What is JSX?
5. How does React update the UI?

Ex. No: 10 — GROCERY DELIVERY APPLICATION USING REACT JS

Aim:

To design and develop a **basic grocery delivery application interface** using **React JS**.

Software Requirements:

• VS Code
• Node.js and npm
• Modern browser
• React JS

Description:

A **Grocery Delivery Application** allows users to view products and place orders online.
In this experiment, a **basic frontend UI** is created using React to:
• Display grocery items
• Show price details
• Add items to cart

This experiment demonstrates **real-world application development** using React components.

Procedure:

1. Create a React application using create-react-app.
2. Open project in VS Code.
3. Create components for grocery items.
4. Use state to manage cart count.
5. Display items dynamically.
6. Run the application.
7. Verify grocery listing and cart updates.

PROGRAM:

App.js

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

```
function App() {
   const [cart, setCart] = useState(0);

   return (
      <div style={{ margin: "40px" }}>
         <h2>Grocery Delivery App</h2>
         <h4>Cart Items: {cart}</h4>

         <div>
            <p>Rice - ₹50</p>
            <button onClick={() => setCart(cart + 1)}>Add to Cart</button>
         </div><br />

         <div>
            <p>Milk - ₹30</p>
            <button onClick={() => setCart(cart + 1)}>Add to Cart</button>
         </div>
      </div>
   );
}

export default App;
```

OUTPUT:

- Grocery items are displayed
- Add to Cart button works
- Cart count updates dynamically

Result:

Thus, a **basic grocery delivery application interface using React JS** was successfully designed and implemented using React components and state management.

Viva Questions:

1. What is state in React?
2. How does React update data dynamically?
3. What is the use of hooks?
4. How can backend be integrated with this application?
5. Mention real-world uses of React applications.

TOPIC BEYOUND SYLLABUS

Ex. No: 11 — REAL-TIME CHAT APPLICATION USING REACT JS AND FIREBASE (BEYOND SYLLABUS)

Aim:

To design and develop a real-time chat application using React JS and Firebase, enabling users to send and receive messages instantly.

Software Requirements:

- **VS Code**
- **Node.js and npm**
- **Modern browser**
- **React JS**
- **Firebase (Cloud Firestore)**

Description:

A real-time chat application allows users to exchange messages instantly without refreshing the page.
This experiment integrates React JS with Firebase, a cloud-based backend service.

Firebase provides:
- Real-time database
- Cloud storage
- User authentication

This experiment demonstrates full-stack interaction using frontend React and backend cloud services, making it an advanced topic beyond syllabus.

Procedure:

1. Install Node.js and create a React app using create-react-app.
2. Create a Firebase project in Firebase Console.
3. Enable Firestore database.
4. Connect Firebase with React project.
5. Design chat UI using React components.
6. Store and retrieve messages from Firestore.

7. Display messages in real time.
8. Run the application using npm start.
9. Verify real-time message updates.

PROGRAM (Sample Code):

App.js

```
import React, { useState } from "react";
import { initializeApp } from "firebase/app";
import { getFirestore, collection, addDoc, onSnapshot } from "firebase/firestore";

const firebaseConfig = {
  apiKey: "YOUR_API_KEY",
  authDomain: "YOUR_DOMAIN",
  projectId: "YOUR_PROJECT_ID"
};

const app = initializeApp(firebaseConfig);
const db = getFirestore(app);

function App() {
  const [message, setMessage] = useState("");
  const [messages, setMessages] = useState([]);

  const sendMessage = async () => {
    await addDoc(collection(db, "messages"), {
      text: message,
      timestamp: new Date()
    });
    setMessage("");
```

```
  };

  onSnapshot(collection(db, "messages"), (snapshot) => {
    setMessages(snapshot.docs.map(doc => doc.data()));
  });

  return (
    <div style={{ margin: "40px" }}>
      <h2>Real-Time Chat App</h2>

      <input
        type="text"
        value={message}
        onChange={(e) => setMessage(e.target.value)}
        placeholder="Type message"
      />
      <button onClick={sendMessage}>Send</button>

      <ul>
        {messages.map((msg, index) => (
          <li key={index}>{msg.text}</li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

OUTPUT:

- Chat interface is displayed
- Messages appear instantly
- No page refresh required

Result:

Thus, a Real-Time Chat Application using React JS and Firebase was successfully developed. The application demonstrates real-time data synchronization using cloud services, which is an advanced topic beyond syllabus.

Viva Questions:

1. Why is Firebase used in this application?
2. What is real-time database?
3. Difference between traditional backend and Firebase?
4. How does React update chat messages instantly?
5. Why is this considered beyond syllabus?