Full Stack Development – Lab Programs
(AD2604-1)

Program 6: Interactive Counter Application Using React

Aim: To develop a simple counter application in React that enables users to increment and reset the count, demonstrating state management, event handling, functional components, and styling in React.

Explanation:

- This program is a simple counter app built using React, which is a JavaScript library for creating user interfaces. It uses JSX, which allows writing HTML-like code inside JavaScript.
- The app has a Counter component that keeps track of a number (count) and allows the user to increase or reset it.
- The useState function is used to store and update the count. When the "Increment" button is clicked, the count increases by 1. When the "Reset" button is clicked, the count goes back to 0.
- A separate DisplayCount component is used to show the current count on the screen. This makes the program modular and easy to understand.
- The main App component brings everything together. It displays the heading "Counter App" and includes the Counter component inside it.
- CSS is used to improve the appearance of the app. It centers the content, adds a background color, and styles the buttons to make them look better.
- When the app runs, the user can see the current count and click the buttons to change it. React updates the screen instantly whenever the count changes.

Code:

#Set Up the React Project:

```
# Step 1: Create a new React project named "counter-app"
npx create-react-app counter-app
# Step 2: Navigate into the project folder
cd counter-app
# Step 3: Start the development server to run the app
npm start
```

#Code Implementation

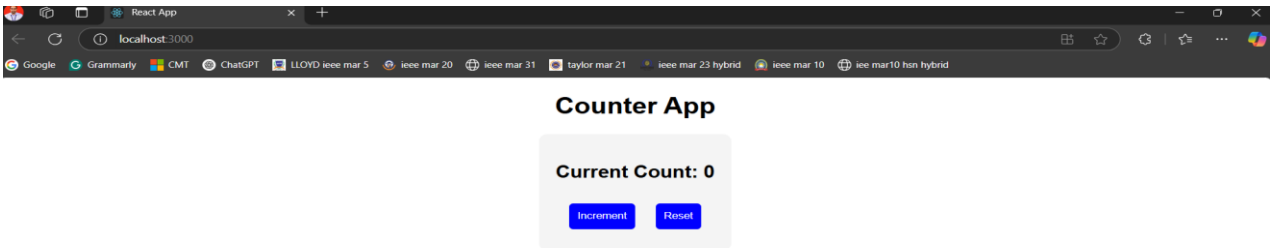
```
// Import React and useState
import React, { useState } from "react";
import "./App.css";  // Import styles
// Component to display the count
const DisplayCount = ({ count }) => <h2>Current Count: {count}</h2>;
// Counter component
const Counter = () => {
   const [count, setCount] = useState(0);  // State to store count
   return (
      <div className="counter">
         <DisplayCount count={count} />  {/* Show count */}
```

```
        <button onClick={() => setCount(count + 1)}>Increment</button>  {/* Increase count */}
        <button onClick={() => setCount(0)}>Reset</button>  {/* Reset count */}
      </div>
    );
};
// Main App component
const App = () => (
    <div className="App">
      <h1>Counter App</h1>  {/* App title */}
      <Counter />  {/* Show counter */}
    </div>
);
export default App;  // Export App component
```

#Styling the Counter (App.css)

```
/* Center text and set font */
.App {
    text-align: center;
    font-family: Arial, sans-serif;
}
/* Box for counter */
.counter {
    background: #f4f4f4;
    padding: 20px;
    border-radius: 10px;
    display: inline-block;
}
/* Style for buttons */
button {
    margin: 10px;
    padding: 10px;
    border: none;
    background: blue;
    color: white;
    cursor: pointer;
    border-radius: 5px;
}
```

Output:



Program 7: User Authentication System Using React

Aim: To create a simple login form using React that allows users to enter a username and password, demonstrating the use of React forms, state management, event handling, and basic CSS styling.

Explanation:

- This program is a simple Login Form created using React. It allows users to enter a username and password and displays a message when they submit the form.
- The program consists of three main files: LoginForm.js, LoginForm.css, and App.js.
- In LoginForm.js, the form is created using React functional components. It includes two input fields for the username and password, and a button to submit the form.
- The program uses the useState hook to store and manage the values entered by the user in the input fields. The username and password fields are updated dynamically as the user types.
- When the form is submitted, an event handler function prevents the default page refresh and shows an alert message displaying the entered username.
- The form is styled using LoginForm.css, which gives a structured layout with a background, input field styles, and button styles.
- The App.js file is the main component that renders the LoginForm component inside a webpage along with a heading.
- The application is built using React, and to run it, dependencies need to be installed using npm install, followed by starting the application using npm start. This will open the form in a web browser.
- The program demonstrates how to handle user input, manage state using React, and process form submission without refreshing the page.

Code:

#Code for Login Form (LoginForm.js)

```
import React, { useState } from "react"; // Import React and useState hook
import "./LoginForm.css"; // Import CSS for styling
function LoginForm() {
  // State variables for storing username and password
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  // Function to handle form submission
  const handleSubmit = (e) => {
    e.preventDefault(); // Prevents page reload
    alert(`Login successful for: ${username}`); // Displays an alert with username
  };
  return (
    <div className="login-container"> {/* Container for the form */}
      <h2>Login</h2>
      <form onSubmit={handleSubmit}>
        {/* Username input */}
        <label>Username:</label>
```

```
            <input type="text" value={username} onChange={(e) => setUsername(e.target.value)}
required />
            {/* Password input */}
            <label>Password:</label>
            <input type="password" value={password} onChange={(e) =>
setPassword(e.target.value)} required />
            {/* Submit button */}
            <button type="submit">Login</button>
          </form>
        </div>
    );
}
export default LoginForm; // Export component
```

#Styling (LoginForm.css)

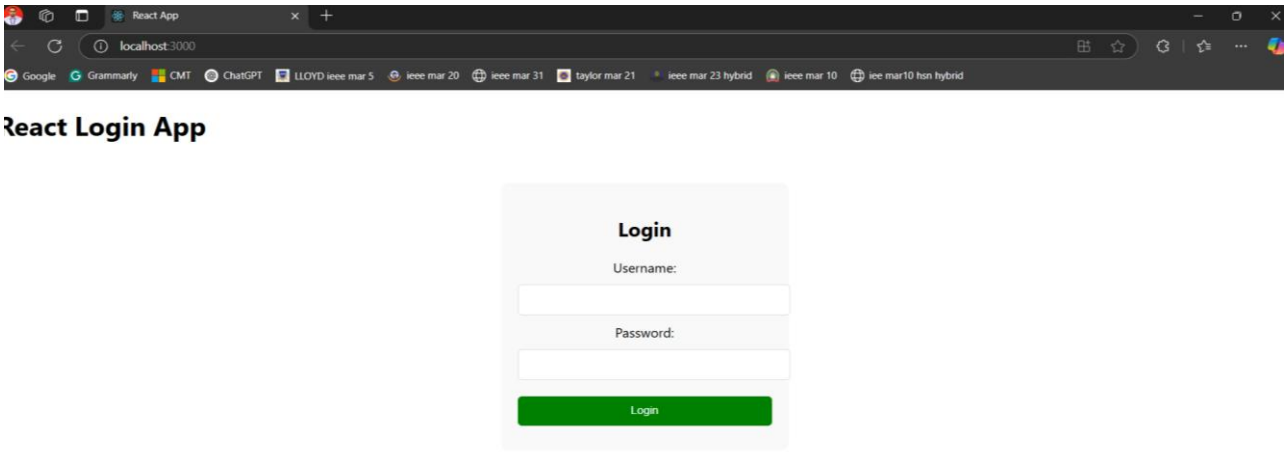
```
/* Styling for the login container */
.login-container {
    width: 300px;
    margin: 50px auto;
    padding: 20px;
    background: #f9f9f9;
    border-radius: 10px;
    text-align: center;
}
/* Common styling for input fields and button */
input, button {
    width: 100%;
    padding: 10px;
    margin: 10px 0;
    border-radius: 5px;
    border: 1px solid #ddd;
}
/* Styling for the submit button */
button {
    background: green;
    color: white;
    border: none;
    cursor: pointer;
}
```

#Update App.js to Include Login Form

```
import React from "react"; // Import React
import LoginForm from "./LoginForm"; // Import LoginForm component
function App() {
    return (
        <div>
            <h1>React Login App</h1> {/* App heading */}
            <LoginForm /> {/* Render the LoginForm component */}
        </div>
    );
```

```
}
export default App; // Export App component
```

Output:



Program 8: Developing a Full-Stack Web Application Using React.js and Express.js

Aim: To demonstrate a client-server architecture using React.js for the frontend and Express.js for the backend by building a simple full-stack web application where the frontend and backend communicate with each other.

Explanation:

- This program creates a simple full-stack web application using React.js for the frontend and Express.js for the backend.
- Backend – Created using Express.js. It works like the brain of the program, running on the server and handling requests from the frontend. When the frontend asks for data, the backend responds with "Hello, World!".
- Frontend – Created using React.js. It works like the face of the program, running in the browser and showing the message from the backend.
- Node.js sets up the backend and allows running JavaScript code outside the browser.
- Express.js creates an endpoint /api/message to send a JSON response with "Hello, World!".
- CORS (Cross-Origin Resource Sharing) allows the frontend to access the backend even when running on different ports.
- The frontend sends a request to the backend using the fetch API and converts the response into JSON format.
- useState stores the message received from the backend. useEffect sends the request to the backend when the component loads.

- The message is displayed in the frontend using JSX (JavaScript XML).
- The program shows how to connect a client (React.js) and a server (Express.js) using HTTP communication.

Code:

 # Open a terminal and create a folder for the backend:

```
mkdir backend
cd backend
# Initialize a Node.js project:
npm init -y
# Install the required dependencies
npm install express cors
```

 # Create a new file named server.js inside the backend folder and add the following code:

```
// Import required modules
const express = require('express');
const cors = require('cors');
const app = express(); // Create an Express app
const PORT = 5000; // Define the port number
// Middleware to enable CORS (Allows requests from different origins)
app.use(cors());
// Define a route to send a JSON response
app.get('/api/message', (req, res) => {
   res.json({ message: "Hello, World!" }); // Send "Hello, World!" message as a JSON response
});
// Start the server and listen on PORT 5000
app.listen(PORT, () => {
   console.log(`Server is running on http://localhost:${PORT}`);
});
```

#Start the backend server:

```
node server.js
```

#Open a new terminal and create a React.js project

```
npx create-react-app frontend
#Navigate to the frontend folder
cd frontend
```

 # Open the App.js file in the frontend/src folder and replace the existing code with the following:

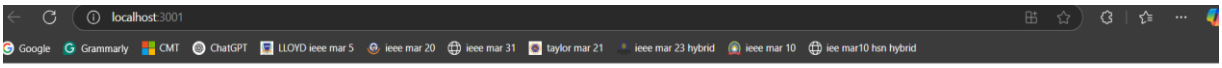
```
import React, { useState, useEffect } from "react";
function App() {
   // State to store the fetched message
```

```
  const [message, setMessage] = useState("Loading...");
  // Fetch the message from the backend when the component is loaded
  useEffect(() => {
    fetch("http://localhost:5000/api/message") // API endpoint from backend
      .then(response => response.json()) // Convert response to JSON format
      .then(data => setMessage(data.message)) // Update the state with the message
      .catch(error => console.error("Error fetching message:", error)); // Handle errors
  }, []); // Empty dependency array ensures it runs only once when the component loads
  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      {/* Display the fetched message */}
      <h1>{message}</h1>
    </div>
  );
}
export default App;
```

 # Start the React frontend:

```
npm start
```

Output:



Hello, World!