SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Tiruchirappalli, Tamil Nadu - 621105.

Department of Computer Applications

PRACTICAL RECORD

NAME :

REGISTER NO : RA25322410500

COURSE : MCA

SEMESTER / YEAR : I / I

SUBJECT CODE : PCA25D02J

SUBJECT NAME : ADVANCED WEB TECHNOLOGY

OCTOBER - 2025

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Tiruchirappalli, Tamil Nadu - 621105.

Department of Computer Applications

BONAFIDE CERTIFICATE

This is to certify that the bonafide work is done by Mr/Ms _____

Register No._____ in the **Advanced Web Technology (Subject Code: PCA25D02J** at Computer Lab, SRM INSTITUTE OF SCIENCE & TECHNOLOGY, Tiruchirappalli Campus, in October 2025.

STAFF IN-CHARGE **HEAD OF THE DEPARTMENT**

Submitted for the University Practical Examination held at SRM Institute of Science & Technology, Tiruchirappalli Campus, Department of Computer Applications on _____.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

TABLE OF CONTENTS

S.NO.	DATE	EXPERIMENT TITLE	PAGE.NO	SIGNATURE
1		Sample Application Development		
2		Create a Responsive Website using HTML5 and CSS3		
3		Create a Progressive Web App (PWA) that can Work Offline		
4		Develop a Dynamic SPA (Single Page Application) using React.JS		
5		Create an Interactive Form with Validation using JavaScript and CSS3		
6		Simulate and Mitigate Common Web Vulnerabilities (XSS, SQL Injection) in a Simple Application		
7		Optimize a Web Application for Performance (Minification, Lazy Loading, etc.)		
8		Implement Automated Testing for a Java Script Based HTML Application		
9		Create a To-Do List Application using React.js with Redux for State Management		
10		Integrate JWT Authentication in a Node.js Backend Application		

Program No: 1 File Name: Ex. No: Date: _____	SAMPLE APPLICATION DEVELOPMENT

AIM:

To design and implement a simple login form using HTML, CSS, and JavaScript that validates a username and password on the client side.

ALGORITHM:

Step 1: Start Visual Studio Code (VS Code) on your computer.

Step 2: Create a new HTML file named login.html.

Step 3: Design the login form inside login.html using HTML input elements for username and password.

Step 4: Style the form using CSS to arrange the layout and background neatly.

Step 5: Write JavaScript in the same HTML file or a separate .js file to check if the entered username and password match predefined values (e.g., username = admin, password = 1234) and display messages like “Login Successful” or “Invalid Username or Password.”

Step 6: Install and run the Live Server extension in VS Code by opening the login.html file, right-clicking inside the editor, and selecting “Open with Live Server” to launch the form in your default browser.

Step 7: Test the login form by entering valid and invalid credentials and observe the output messages.

PROGRAM:

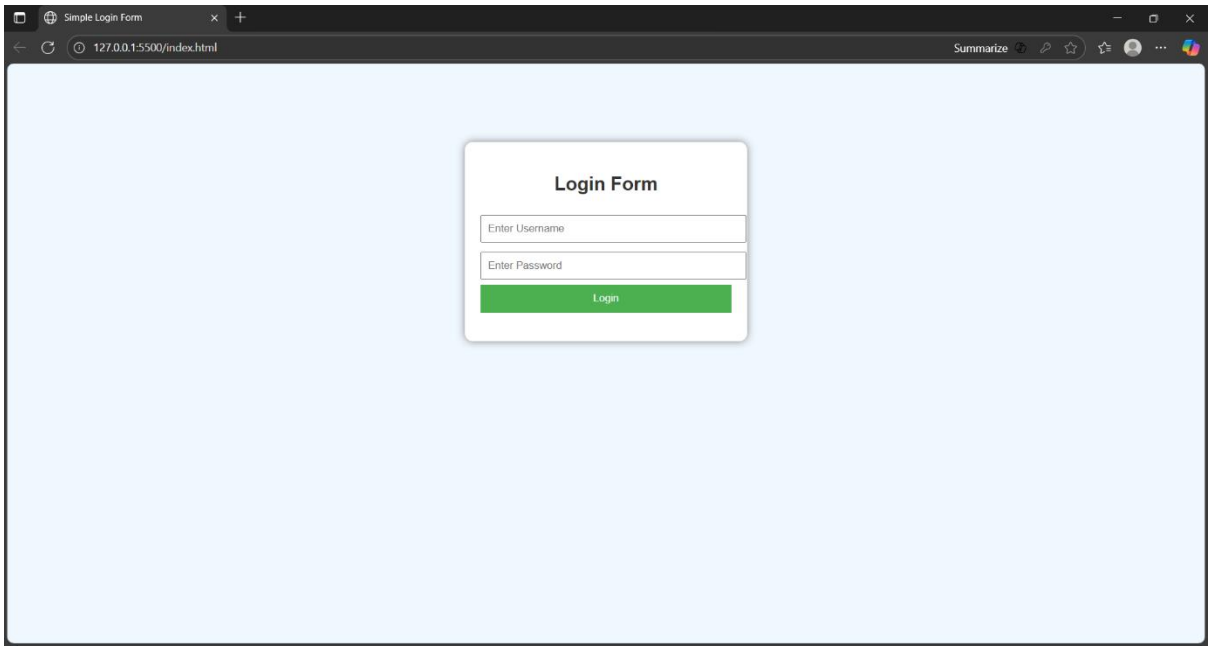
```
<!DOCTYPE html>

<html>

<head>
```

```
<title>Simple Login Form</title>
<style>
  body {
    font-family: Arial;
    background-color: #f0f8ff;
  }
  .container {
    width: 320px;
    margin: 100px auto;
    background: white;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 0 10px gray;
  }
  h2 {
    text-align: center;
    color: #333;
  }
  input {
    width: 100%;
    padding: 8px;
    margin: 6px 0;
  }
  button {
    width: 100%;
    background-color: #4CAF50;
    color: white;
```

```
            padding: 10px;
            border: none;
            cursor: pointer;
        }
        button:hover {
          background-color: #45a049;
        }
        p {
            text-align: center;
            color: red;
        }
    </style>
</head>
<body>
<div class="container">
    <h2>Login Form</h2>
    <input type="text" id="username" placeholder="Enter Username" required>
    <input type="password" id="password" placeholder="Enter Password" required>
    <button onclick="login()">Login</button>
    <p id="message"></p>
</div>
<script>
function login() {
    // Get values from input boxes
    var user = document.getElementById("username").value;
    var pass = document.getElementById("password").value;
    // Set a simple username and password
```

```
    if (user === "admin" && pass === "1234") {
        document.getElementById("message").style.color = "green";
        document.getElementById("message").innerHTML = "Login Successful ✅";
    } else {
        document.getElementById("message").style.color = "red";
        document.getElementById("message").innerHTML = "Invalid Username or Password ❌";
    }
}
</script>
</body>
</html>
```

OUTPUT:



RESULT:

Thus the program was successfully executed.

Program No: 2 File Name: Ex. No: Date: _____	**CREATE A RESPONSIVE WEBSITE USING HTML5 AND CSS3.**

AIM:

To design and develop a simple responsive webpage using HTML5 and CSS3 that adjusts its layout according to different screen sizes (desktop, tablet, and mobile).

ALGORITHM:

Step 1: Open Visual Studio Code (or any preferred text editor) on your computer to begin creating the web page.

Step 2: Create a new file and save it as index.html in a dedicated project folder to store all related files neatly.

Step 3: Write the HTML code to structure the webpage — include essential sections such as header, navigation bar, main content area, and footer.

Step 4: Add CSS3 styles either within a <style> tag in the HTML file or through an external stylesheet (style.css) to customize colors, fonts, and layouts.

Step 5: Implement media queries in CSS to adjust layout and design according to different screen widths such as mobile, tablet, and desktop.

Step 6: Save all changes and open the index.html file in a web browser (or use the Live Server extension in VS Code) to preview your webpage.

Step 7: Test the webpage by resizing the browser window or using developer tools to check how the layout adapts on both desktop and mobile views, ensuring full responsiveness.

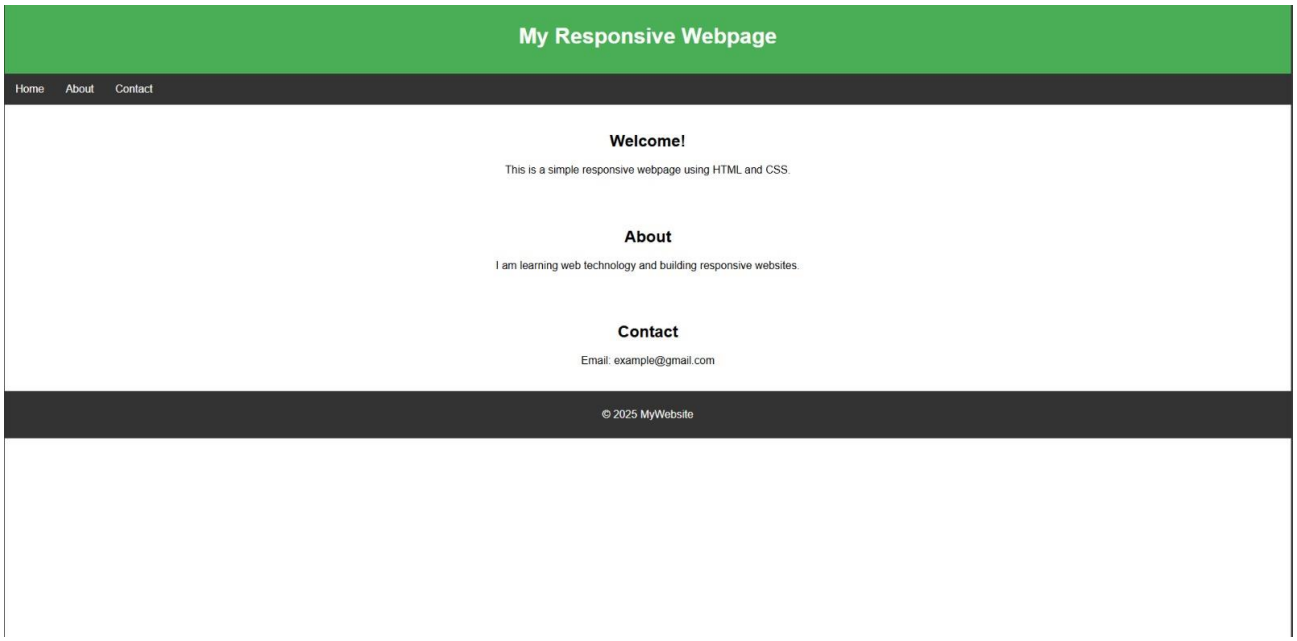
PROGRAM:

```
program:(index.html)<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Simple Responsive Website</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
    }
    header {
      background-color: #4CAF50;
      color: white;
      text-align: center;
      padding: 15px;
    }
    nav {
      background-color: #333;
      overflow: hidden;
    }
    nav a {
      float: left;
      color: white;
      text-align: center;
      padding: 14px 16px;
      text-decoration: none;
```

```
    }
    nav a:hover {
      background-color: #ddd;
      color: black;
    }
    section {
      padding: 20px;
      text-align: center;
    }
    footer {
      background-color: #333;
      color: white;
      text-align: center;
      padding: 10px;
    }
    /* Responsive Design */
    @media screen and (max-width: 600px) {
      nav a {
        float: none;
        display: block;
      }
    }
  </style>
</head>
<body>
  <header>
    <h1>My Responsive Webpage</h1>
  </header>
  <nav>
```

```
    <a href="#home">Home</a>
    <a href="#about">About</a>
    <a href="#contact">Contact</a>
  </nav>
  <section id="home">
    <h2>Welcome!</h2>
    <p>This is a simple responsive webpage using HTML and CSS.</p>
  </section>
  <section id="about">
    <h2>About</h2>
    <p>I am learning web technology and building responsive websites.</p>
  </section>
  <section id="contact">
    <h2>Contact</h2>
    <p>Email: example@gmail.com</p>
  </section>
  <footer>
    <p>&copy; 2025 MyWebsite</p>
  </footer>
</body>
</html>
```

OUTPUT:



RESULT:

Thus the program was successfully executed.

Program No: 3 File Name: Ex. No: Date: _____	**CREATE A PROGRESSIVE WEB APP (PWA) THAT CAN WORK OFFLINE**

AIM:

To design and develop a Progressive Web App (PWA) using HTML, CSS, and JavaScript that can work offline by caching resources through a service worker.

ALGORITHM:

Step 1: Create an HTML file (index.html) as the main structure of the web app.

Step 2: Design the layout using a CSS stylesheet (style.css).

Step 3: Create a manifest.json file to describe the app's name, icons, and start URL for installability.

Step 4: Write a JavaScript file (service-worker.js) that defines caching logic for offline use.

Step 5: Register the service worker in app.js using the navigator.serviceWorker.register() method.

Step 6: Store required assets (HTML, CSS, JS, manifest, icons) inside a cache during the service worker's install event.

Step 7: Intercept network requests during the fetch event and serve cached files when offline.

Step 8: Build and organize all files in one project directory structure.

Step 9: Run a local server using the command npx http-server or python -m http.server.

Step 10: Open http://localhost:8080 in a modern browser (Chrome/Edge).

Step 11: Verify offline functionality by turning on "Offline" mode in DevTools and refreshing the page.

Step 12: Confirm that the app loads from cache and remains accessible without network connectivity.

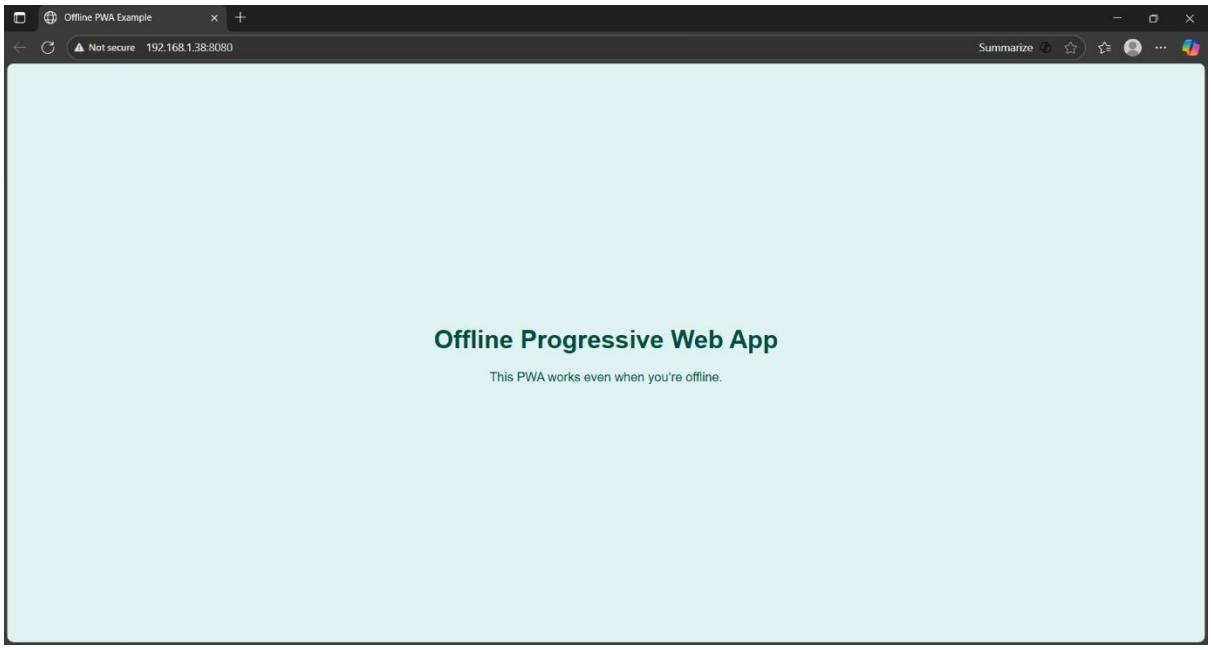
PROGRAM:

```
<!DOCTYPE html>

<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Offline PWA Example</title>

  <link rel="stylesheet" href="style.css" />
  <link rel="manifest" href="manifest.json" />
  <link rel="icon" href="icon-192.png" sizes="192x192" />
  <meta name="theme-color" content="#00796B" />
</head>
<body>
  <main>
    <h1>Offline Progressive Web App</h1>
    <p>This PWA works even when you're offline.</p>
  </main>

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

html, body {
  margin: 0;
  padding: 0;
  height: 100%;
  background-color: #e0f2f1;
  color: #004d40;
  font-family: Arial, sans-serif;
  display: flex;
  justify-content: center;
  align-items: center;
}

main {
  text-align: center;
}

{
  "name": "Offline PWA Example",
  "short_name": "OfflinePWA",
  "start_url": "./index.html",
```

```
  "display": "standalone",
  "background_color": "#e0f2f1",
  "theme_color": "#00796B",
  "icons": [
    {
      "src": "icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
// Register the service worker
if ('serviceWorker' in navigator) {
  window.addEventListener('load', () => {
    navigator.serviceWorker
      .register('service-worker.js')
      .then(reg => console.log('Service Worker registered:', reg))
      .catch(err => console.error('Service Worker registration failed:', err));
  });
}

const CACHE_NAME = 'pwa-offline-cache-v1';
const ASSETS = [
  '/',
  '/index.html',
  '/style.css',
  '/app.js',
  '/manifest.json',
  '/icon-192.png',
  '/icon-512.png'
];

// Install event – cache app shell
self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME).then(cache => cache.addAll(ASSETS))
  );
  self.skipWaiting();
});

// Activate event – remove old caches
```

```
self.addEventListener('activate', event => {
  event.waitUntil(
    caches.keys().then(keys =>
      Promise.all(keys.filter(k => k !== CACHE_NAME).map(k => caches.delete(k)))
    )
  );
});

// Fetch event – serve cached resources when offline
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then(resp => resp || fetch(event.request))
      .catch(() => caches.match('/index.html'))
  );
});
```

OUTPUT:



RESULT:

Thus the program was successfully executed.

Program No: 4 File Name: Ex. No: Date: _____	**DEVELOP A DYNAMIC SPA (SINGLE PAGE APPLICATION) USING REACT.JS.**

AIM:

The aim of developing a dynamic SPA using React.js is to build a fast, interactive web app that loads a single HTML page and updates content dynamically without full page reloads, providing a smooth user experience and better perform

ALGORITHM:

Step 1: Set up the React development environment with Node.js and Create React App.

Step 2: Structure the app with reusable components like Header, Footer, and Main content.

Step 3: Use React Router for client-side navigation without page reloads.

Step 4: Create individual component pages for dynamic content display.

Step 5: Manage app state efficiently using React's state or external libraries.

Step 6: Add styling and optimize performance with minified CSS/JS.

Step 7: Test the SPA on modern browsers and deploy it to a hosting platform.

PROGRAM:

src/main.jsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import App from "./App";
import "./index.css";

ReactDOM.createRoot(document.getElementById("root")).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

src/App.jsx

```
import { Routes, Route } from "react-router-dom";
import Navbar from "./components/Navbar";
import Footer from "./components/Footer";
import Home from "./pages/Home";
import About from "./pages/About";
import Services from "./pages/Services";
import Contact from "./pages/Contact";

export default function App() {
  return (
    <div className="flex flex-col min-h-screen bg-gray-50">
```

```
      <Navbar />
      <main className="flex-1 p-6 container mx-auto">
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/services" element={<Services />} />
          <Route path="/contact" element={<Contact />} />
        </Routes>
      </main>
      <Footer />
    </div>
  );
}
```

src/pages/Home.jsx

```
import data from "../data";
import Card from "../components/Card";

export default function Home() {
  return (
    <div>
      <h2 className="text-3xl font-bold mb-6 text-center text-blue-700">
        Welcome to Dynamic SPA
      </h2>
      <div className="grid gap-6 md:grid-cols-3 sm:grid-cols-2">
        {data.map((item) => (
          <Card key={item.id} title={item.title} description={item.description} />
```

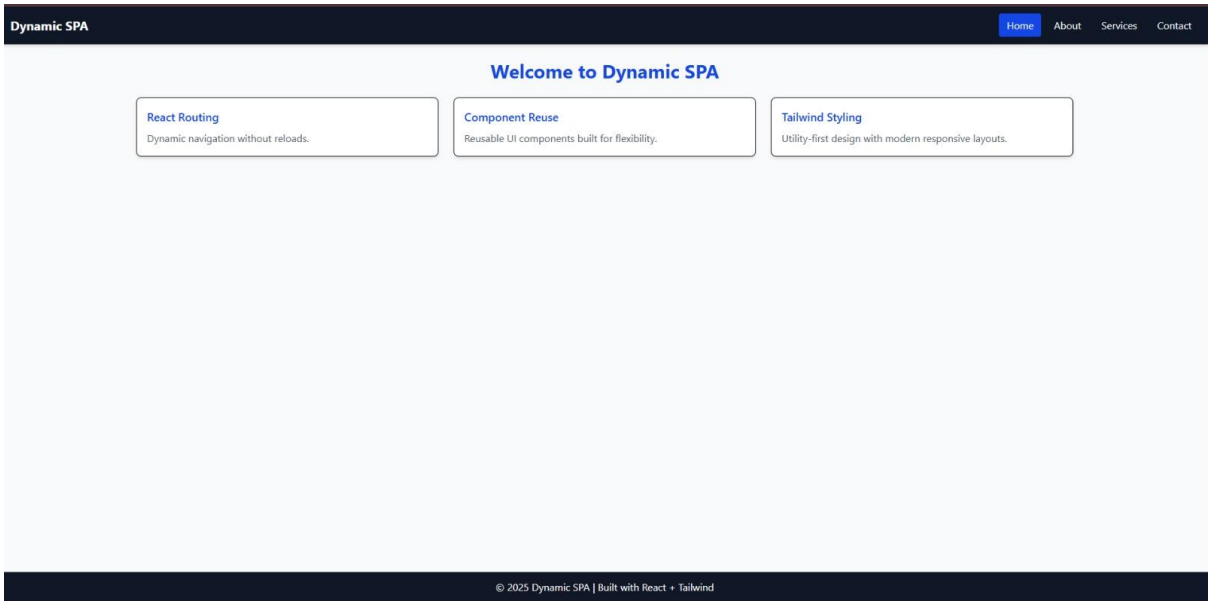
```
      ))}
    </div>
  </div>
 );
}
```

src/components/Navbar.jsx

```
import { Link, useLocation } from "react-router-dom";

export default function Navbar() {
 const location = useLocation();

 const linkClass = (path) =>
  `px-3 py-2 rounded hover:bg-blue-600 ${
   location.pathname === path ? "bg-blue-700" : ""
  }`;

 return (
  <nav className="bg-gray-900 text-white flex justify-between items-center p-4 shadow-
md">
   <h1 className="text-xl font-bold">Dynamic SPA</h1>
   <div className="space-x-2">
    <Link to="/" className={linkClass("/")}>
     Home
    </Link>
    <Link to="/about" className={linkClass("/about")}>
     About
```

```
        </Link>
        <Link to="/services" className={linkClass("/services")}>
          Services
        </Link>
        <Link to="/contact" className={linkClass("/contact")}>
          Contact
        </Link>
      </div>
    </nav>
  );
}
```

OUTPUT:



RESULT:

Thus the program was successfully executed

Program No: 5 File Name: Ex. No: Date: _____	**Create an interactive form with validation using JavaScript and CSS3**

AIM:

To create a simple interactive web form with input validation using HTML, CSS, and JavaScript.

ALGORITHM:

1)Create an HTML file to design the form structure with input fields (e.g., Name, Email, Password).

2)Add CSS to style the form for better appearance.

3)Write JavaScript code to validate the input fields when the form is submitted: Check if required fields are filled. Validate email format. Check password length.

4)Display error messages if any input is invalid.

5)Show success message when all inputs are valid.

6)Test the form by entering valid and invalid data to ensure validation works correctly.

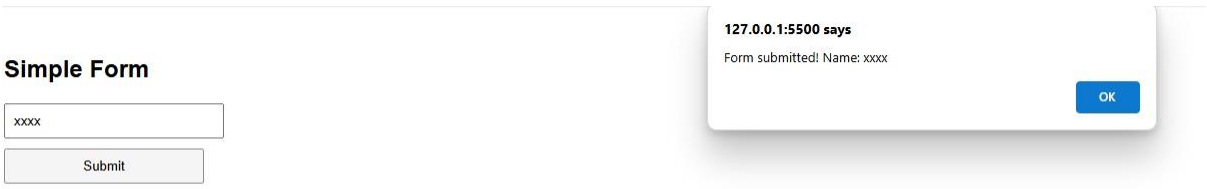
PROGRAM:

```
<!DOCTYPE html>

<html>

<head>

  <title>Easy Form</title>

  <style>

    body { font-family: Arial; padding: 20px; }

    input, button { display: block; margin: 10px 0; padding: 8px; width: 200px; }

    .error { color: red; font-size: 14px; }
```

```
  </style>
</head>
<body>

  <h2>Simple Form</h2>

  <form id="form">
    <input id="name" placeholder="Name" />
    <div id="error" class="error"></div>

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

  <script>
    const form = document.getElementById('form');
    const nameInput = document.getElementById('name');
    const errorDiv = document.getElementById('error');

    form.onsubmit = function(e) {
      e.preventDefault();
      errorDiv.textContent = "";

      if (nameInput.value.trim() === "") {
```

```
        errorDiv.textContent = "Name is required!";
      } else {
        alert("Form submitted! Name: " + nameInput.value);
        nameInput.value = "";
      }
    }
  </script>

</body>
</html>
```

OUTPUT:



RESULT:

Thus the program was successfully executed

Program No: 6 File Name: Ex. No: Date: _____	**Simulate and mitigate common web vulnerabilities (XSS, SQL Injection) in a sample application**

AIM:

To demonstrate common web vulnerabilities (XSS and SQL injection) safely and show simple mitigations in a browser-based demo.

ALGORITHM:

Step 1 : Create an HTML file with a form containing Name, Email, and Password inputs and a submit button.

Step 2 : Add CSS to style the form for readability and better appearance.

Step 3: Write JavaScript to validate inputs on submit: check required fields, validate email format, and check password length.

Step 4: Display error messages next to invalid fields if any validation fails.

Step 5: Show a success message when all inputs are valid.

Step 6: Test the form with valid and invalid data to ensure validation works correctly.

PROGRAM:

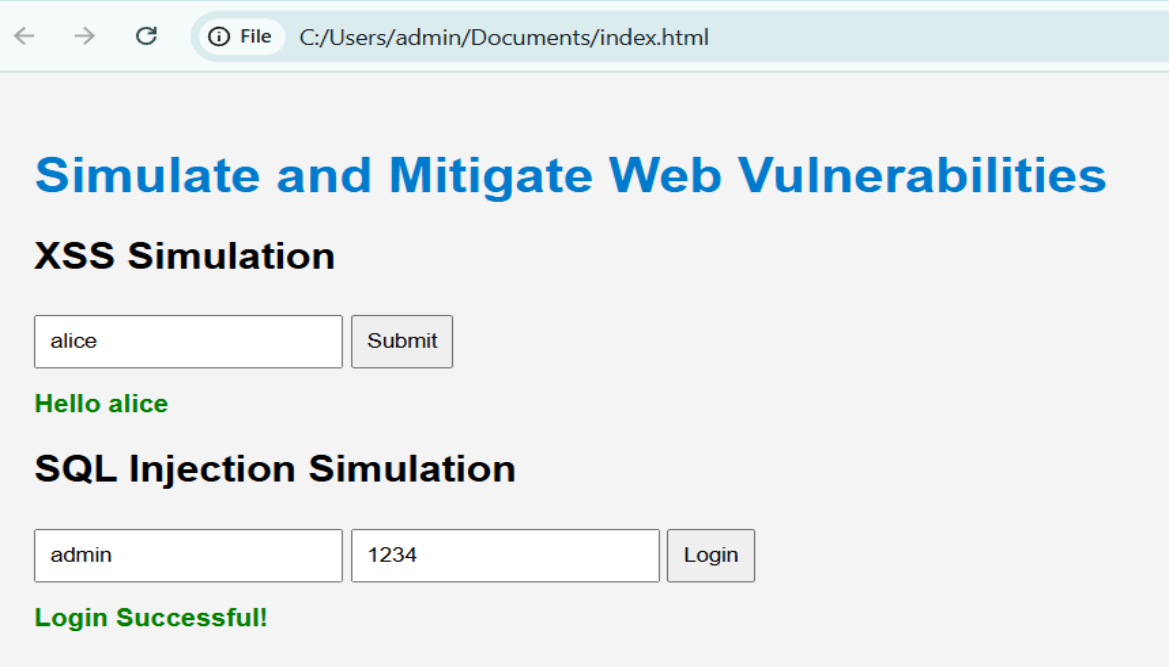
```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Practice 14: Web Vulnerabilities</title>
  <style>
    body { font-family: Arial, sans-serif; padding: 20px; background: #f4f4f4; }
    h1 { color: #007acc; }
    input, button { padding: 8px; margin: 5px 0; }
    .output { margin-top: 10px; color: green; font-weight: bold; }
  </style>
</head>
<body>

  <h1>Simulate and Mitigate Web Vulnerabilities</h1>

  <!-- XSS Example -->
  <h2>XSS Simulation</h2>
  <input type="text" id="xssInput" placeholder="Enter your name">
  <button onclick="showMessage()">Submit</button>
  <div id="xssOutput" class="output"></div>

  <!-- SQL Injection Example (Simulation) -->
  <h2>SQL Injection Simulation</h2>
  <input type="text" id="username" placeholder="Username">
```

```
<input type="text" id="password" placeholder="Password">

<button onclick="login()">Login</button>

<div id="sqlOutput" class="output"></div>

<script>

  // --- XSS Vulnerable Version ---

  function showMessage() {

    const input = document.getElementById('xssInput').value;

    // Vulnerable version (don't use in production)

    // document.getElementById('xssOutput').innerHTML = "Hello " + input;

    // Safe version (mitigation)

    const safeText = document.createTextNode(input);

    const output = document.getElementById('xssOutput');

    output.innerHTML = '';

    output.appendChild(document.createTextNode("Hello "));

    output.appendChild(safeText);

  }

  // --- SQL Injection Simulation ---

  const users = [

    { username: 'admin', password: '1234' },

    { username: 'user', password: 'abcd' }

  ];
```

```
    function login() {
      const username = document.getElementById('username').value;
      const password = document.getElementById('password').value;

      // Vulnerable simulation (for demonstration only)
      // const query = `SELECT * FROM users WHERE username='${username}' AND password='${password}'`;
      // Safe mitigation using simple check (like prepared statements)
      const user = users.find(u => u.username === username && u.password === password);
      const output = document.getElementById('sqlOutput');
      if (user) {
        output.textContent = 'Login Successful!';
      } else {
        output.textContent = 'Invalid username or password.';
      }
    }
  </script>
</body>
</html>
```

OUTPUT:



RESULT:

Thus the program was successfully executed

Program No: 7 File Name: Ex. No: Date: _____	**Optimize a web application for performance (minification, lazy loading, etc.)**

AIM:

To create a simple web page optimized for performance using minified CSS/JS and lazy loading of images.

ALGORITHM:

Step 1: Write basic HTML structure with <!DOCTYPE html>, <html>, <head>, and <body>.

Step 2 : Add a <title> to name the web page.

Step 3 : Include CSS styles inside a <style> tag with minified (compact) CSS to reduce file size.

Step 4 : Add content like headings and paragraphs inside the <body>.

Step 5 : Insert images using <img> tags with the attribute loading="lazy" to enable lazy loading.

Step 6 : Write a small JavaScript function inside <script> with minified code for faster loading.

Step 7 : Save the file with .html extension and open it in a modern browser to see optimized performance.

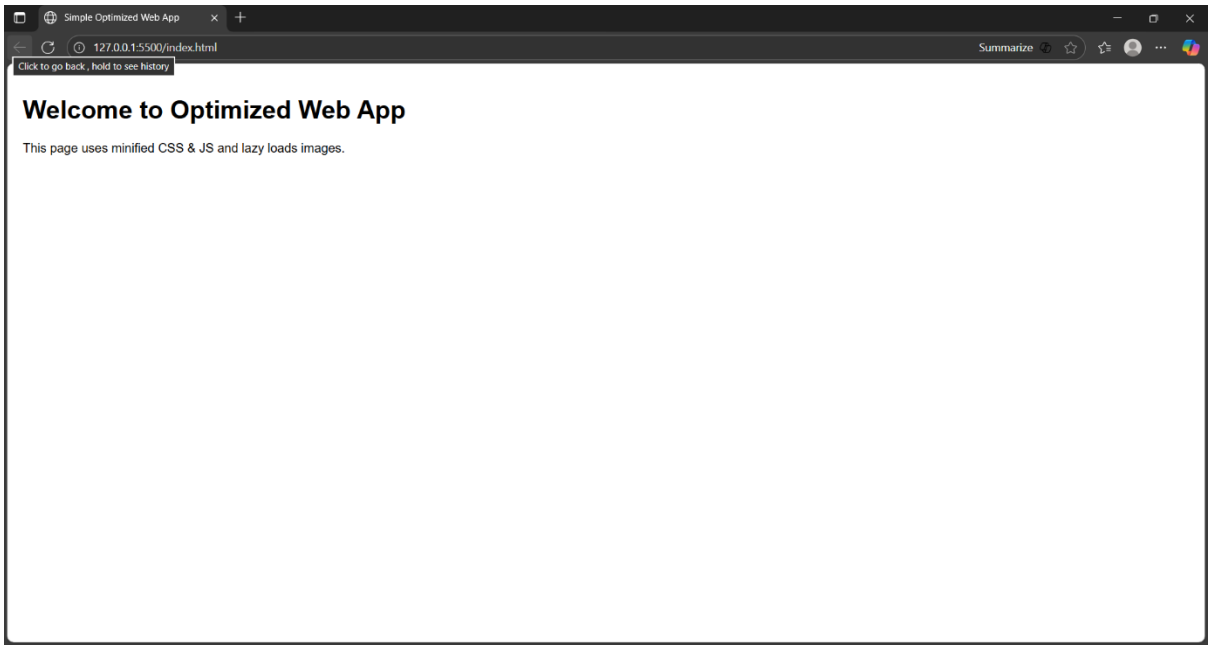
PROGRAM:

<!DOCTYPE html>

<html>

```
<head>
  <title>Simple Optimized Web App</title>
  <style>body{font-family:Arial,sans-
serif;background:#fff;color:#000;padding:20px;margin:0}</style>
</head>
<body>

  <h1>Welcome to Optimized Web App</h1>
  <p>This page uses minified CSS & JS and lazy loads images.</p>

  <img src="https://via.placeholder.com/600x400" alt="Sample 1" loading="lazy" />
  <img src="https://via.placeholder.com/600x400" alt="Sample 2" loading="lazy" />

  <script>function greet(){alert("Hello! Optimized!")}greet();</script>

</body>
</html>
```

OUTPUT:



RESULT:

Thus the program was successfully executed

Program No: 8 File Name: Ex. No: Date: _____	**IMPLEMENT AUTOMATED TESTING FOR A JAVASCRIPT-BASED HTML APPLICATION**

AIM:

To learn how to write and run automated unit tests for JavaScript functions used in a web application using Jest testing framework This helps ensure code correctness and detect bugs early.

ALGORITHM:

1. Create your JavaScript functions file, for example app.js, which contains the code to be tested.

2. Export the functions in app.js using module.exports to make them accessible for testing.

3. Initialize a Node.js project in VS Code by running npm init -y in the terminal.

4. Install Jest as a development dependency with npm install --save-dev jest.

5. Add a test script in package.json under scripts: "test": "jest".

6. Create a test file, e.g. app.test.js, where you import the functions from app.js.

7. Write test cases using Jest's test() and expect() functions to check expected output.

8. Run automated tests in the terminal by running npm test.

9. Optionally, use the Live Server extension in VS Code to serve and view the HTML page during development.

PROGRAM:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

  <title>Tested App</title>

  <script src="app.js" defer></script>

</head>

<body>

  <h1>Simple App for Testing</h1>

</body>

</html>

function add(a, b) {

  return a + b;

}



function multiply(a, b) {

  return a * b;

}



// Export functions for Jest
```

```
if (typeof module !== 'undefined') {
  module.exports = { add, multiply };
}
```

```
npm init -y
npm install --save-dev jest
```

```
"scripts": {
  "test": "jest"
}
```

```
const { add, multiply } = require('./app.js');


test('adds two numbers', () => {
  expect(add(2, 3)).toBe(5);
});


test('multiplies two numbers', () => {
  expect(multiply(2, 3)).toBe(6);
});
```

```
npm test
```

OUTPUT:

RESULT:

Thus the program was successfully executed.

Program No: 9 File Name: Ex. No: Date: _____	**Create a to-do list application using React.js with Redux for state management.**

AIM:

To create a simple, interactive to-do list web application using React and Redux for managing tasks.

ALGORITHM:

Step 1: Load the HTML file in a modern web browser to initialize the React app with Redux store.

Step 2: Enter a new to-do item in the input field and click "Add" to append it to the list.

Step 3: Check the checkbox next to a to-do item to mark it as completed, applying strikethrough styling.

Step 4: Double-click on a to-do item's text or click "Edit" to enter edit mode and modify the text.

Step 5: In edit mode, update the draft text and press Enter or click "Save" to confirm changes.

Ste p 6: Click "Cancel" in edit mode to discard changes and return to view mode.

Step 7: Click "Delete" next to a to-do item to remove it permanently from the list.

Step 8: Click "Clear Completed" to remove all completed to-do items at once.

Step 9: The app persists state in memory via Redux, updating the UI reactively on each action

PROGRAM:

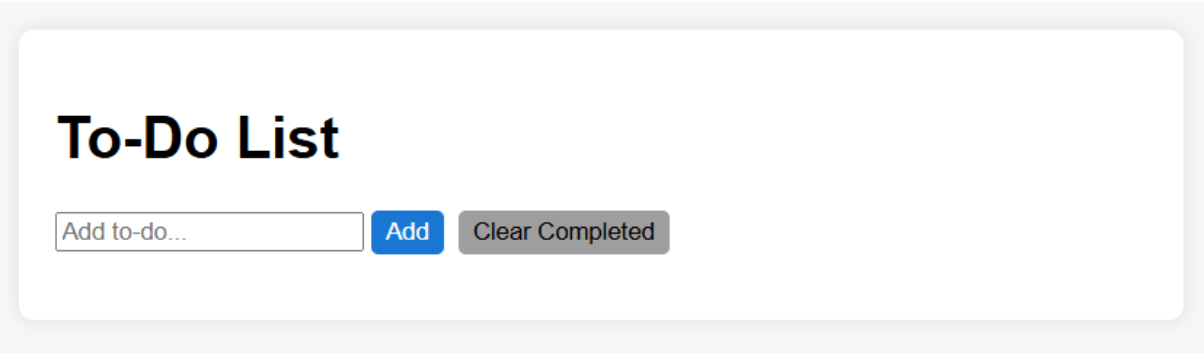
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>To-Do List</title>
  <script src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/redux@4.2.1/dist/redux.js"></script>
  <script src="https://unpkg.com/react-redux@8.0.5/dist/react-redux.js"></script>
  <script src="https://unpkg.com/@reduxjs/toolkit@1.9.5/dist/redux-toolkit.umd.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
  <style>
    body { font-family: Arial; background: #f7f7f8; padding: 20px; }
    .app { max-width: 600px; margin: auto; background: white; padding: 20px; border-radius: 8px; box-shadow: 0 0 10px rgba(0,0,0,0.1); }
    ul { list-style: none; padding: 0; }
    li { display: flex; align-items: center; padding: 8px; border-bottom: 1px solid #eee; }
    .completed { text-decoration: line-through; opacity: 0.7; }
    button { margin: 0 4px; padding: 4px 8px; border: none; border-radius: 4px; cursor: pointer; }
    .add { background: #1976d2; color: white; }
    .clear { background: #9e9e9e; }
    .edit { background: #ffd54f; }
    .delete { background: #ef5350; color: white; }
    .save { background: #4caf50; color: white; }
    .cancel { background: #9e9e9e; }
  </style>
</head>
```

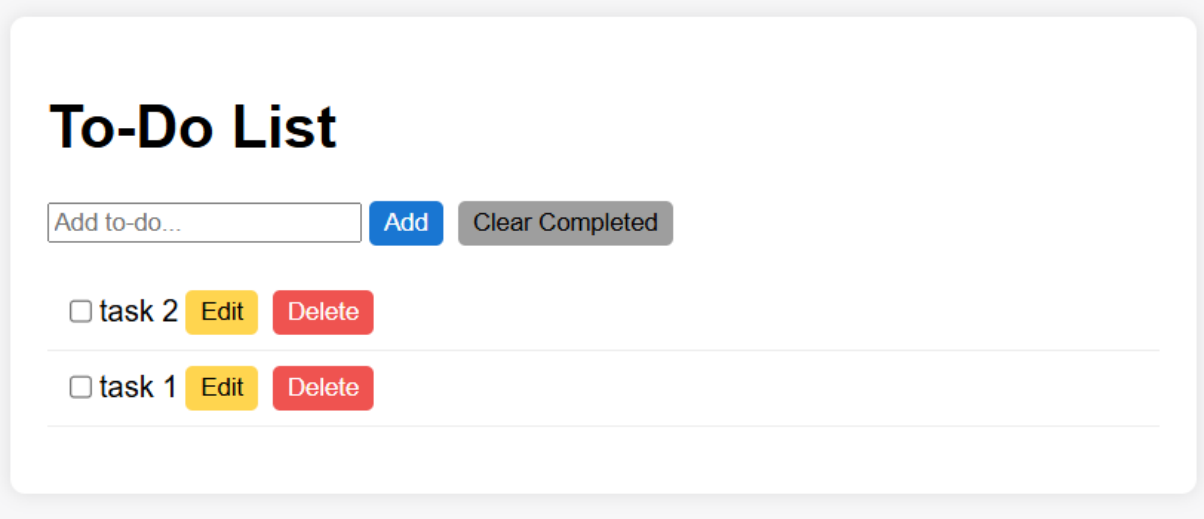
```
<body>
  <div id="root"></div>
  <script type="text/babel">
    const { useState } = React;
    const { Provider, useSelector, useDispatch } = ReactRedux;
    const { configureStore, createSlice, nanoid } = RTK;

    const todosSlice = createSlice({
      name: "todos",
      initialState: [],
      reducers: {
        addTodo: (state, action) => { state.unshift({ id: nanoid(), text: action.payload.trim(), completed: false }); },
        toggleTodo: (state, action) => { const t = state.find(t => t.id === action.payload); if (t) t.completed = !t.completed; },
        deleteTodo: (state, action) => state.filter(t => t.id !== action.payload),
        editTodo: (state, action) => { const t = state.find(t => t.id === action.payload.id); if (t) t.text = action.payload.text.trim(); },
        clearCompleted: (state) => state.filter(t => !t.completed),
      },
    });
    const { addTodo, toggleTodo, deleteTodo, editTodo, clearCompleted } = todosSlice.actions;
    const store = configureStore({ reducer: todosSlice.reducer });
    function App() {
      const [text, setText] = useState("");
      const [editing, setEditing] = useState(null);
      const [draft, setDraft] = useState("");
      const todos = useSelector(state => state);
      const dispatch = useDispatch();
      const add = (e) => { e.preventDefault(); if (text.trim()) { dispatch(addTodo(text)); setText(""); } };
```

```
    const save = () => { if (draft.trim()) dispatch(editTodo({ id: editing, text: draft })); setEditing(null); };


    return (
      <div className="app">
        <h1>To-Do List</h1>
        <form onSubmit={add}>
          <input value={text} onChange={e => setText(e.target.value)} placeholder="Add to-do..." />
          <button className="add" type="submit">Add</button>
          <button className="clear" type="button" onClick={() => dispatch(clearCompleted())}>Clear Completed</button>
        </form>
        <ul>
          {todos.map(t => (
            <li key={t.id} className={t.completed ? "completed" : ""}>
              <input type="checkbox" checked={t.completed} onChange={() => dispatch(toggleTodo(t.id))} />
              {editing === t.id ? (
                <>
                  <input value={draft} onChange={e => setDraft(e.target.value)} onKeyDown={e => e.key === "Enter" && save()} autoFocus />
                  <button className="save" onClick={save}>Save</button>
                  <button className="cancel" onClick={() => setEditing(null)}>Cancel</button>
                </>
              ) : (
                <>
                  <span onDoubleClick={() => { setEditing(t.id); setDraft(t.text); }}>{t.text}</span>
                  <button className="edit" onClick={() => { setEditing(t.id); setDraft(t.text); }}>Edit</button>
```

```
                <button                 className="delete"                 onClick={()                 =>
dispatch(deleteTodo(t.id))}>Delete</button>

                </>

              )}

            </li>

          ))}

        </ul>

      </div>

    );

  }

  ReactDOM.createRoot(document.getElementById("root")).render(<Provider
store={store}><App /></Provider>);

 </script>

</body>

</html>
```

OUTPUT:





RESULT:

Thus the program was successfully executed.

Program No: 10 File Name: Ex. No: Date: _____	**INTEGRATE JWT AUTHENTICATION IN A NODE.JS BACKEND APPLICATION**

AIM:

To create a Progressive Web App (PWA) in Visual Studio Code that can be installed like a native app and works offline with caching, using a web app manifest and a service worker.

ALGORITHM:

Step 1: Build a fast, responsive website and serve it over HTTPS for security.

Step 2: Create an application shell that includes your basic HTML, CSS, and JavaScript files.

Step 3: Set up a service worker in service-worker.js to cache your app shell files and enable offline functionality.

Step 4: Add a web app manifest (manifest.json) with app metadata and link it in your HTML for installability.

Step 5: Test your PWA locally using VS Code's Live Server and then deploy it on a secure HTTPS host like GitHub Pages or Netlify to enable full PWA features.

PROGRAM:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>My PWA</title>
```

```
  <link rel="stylesheet" href="styles.css">
  <link rel="manifest" href="manifest.json">
</head>
<body>
  <h1>Welcome to My PWA</h1>
  <p>This app works offline!</p>
  <script src="app.js"></script>
</body>
</html>


{
  "name": "My PWA",
  "short_name": "PWA",
  "start_url": ".",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#000000",
  "icons": [
    {
      "src": "icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    }
  ]
}


if ('serviceWorker' in navigator) {
```

```
window.addEventListener('load', () => {
 navigator.serviceWorker.register('/service-worker.js')
  .then(registration => {
   console.log('Service Worker registered with scope:', registration.scope);
  })
  .catch(error => {
   console.error('Service Worker registration failed:', error);
  });
```

OUTPUT

RESULT:

Thus the program was successfully executed.