1.----------

Aim

To develop a simple sample web application demonstrating the use of HTML, CSS, and JavaScript (front-end) and optionally connecting to a back-end.

___

⚙ Procedure

1. Define the Objective

- Example: Create a Student Registration Form or a To-Do List Application.

2. Set Up Development Environment

- Install a code editor (VS Code, Sublime).
- Choose a browser for testing (Chrome, Firefox).
- Optionally, set up a local server (Node.js / XAMPP / Python SimpleHTTPServer).

3. Create Project Structure

sample-app/

├── index.html

├── style.css

└── script.js

4. Develop the Front-End

- Use HTML5 for structure.
- Use CSS3 for styling (responsive design if needed).
- Use JavaScript for interactivity.

5. Add Back-End (Optional)

- Use PHP / Node.js / Python Flask to process form data or store tasks.
- Connect to a Database (MySQL / MongoDB) if persistence is required.

6. Test the Application

- Check in different browsers.
- Validate HTML & CSS (W3C Validator).
- Ensure responsive behavior (desktop, mobile).

☑ Sample Code: To-Do List App

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Sample To-Do App</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h2>My To-Do List</h2>
  <input type="text" id="taskInput" placeholder="Enter a task">
  <button onclick="addTask()">Add Task</button>
  <ul id="taskList"></ul>
  <script src="script.js"></script>
</body>
</html>
```

style.css

```
body {
  font-family: Arial, sans-serif;
```

```
  margin: 20px;
}
input, button {
  padding: 8px;
  margin: 5px;
}
ul {
  list-style-type: none;
  padding: 0;
}
li {
  margin: 5px 0;
  padding: 5px;
  background: #f0f0f0;
}
```

script.js

```
function addTask() {
  let input = document.getElementById("taskInput");
  let taskText = input.value.trim();
  if (taskText === "") return;

  let li = document.createElement("li");
  li.textContent = taskText;

  // Add delete option
  let btn = document.createElement("button");
  btn.textContent = "❌";
  btn.style.marginLeft = "10px";
  btn.onclick = () => li.remove();
```

```
  li.appendChild(btn);
  document.getElementById("taskList").appendChild(li);

  input.value = "";
}
```

Perfect 👍 Let's make Practice 2: Create a Responsive Website using HTML5 and CSS3 into a proper lab-style writeup.

📝 Practice 2: Create a Responsive Website using HTML5 and CSS3

🎯 Aim

To design and develop a simple responsive website using HTML5 and CSS3 features such as semantic tags, media queries, and flexible layouts.

⚙️ Procedure

1. Create the Project Structure

```
responsive-website/
├── index.html
└── style.css
```

2. Design the HTML5 Layout

- Use semantic tags (<header>, <nav>, <main>, <section>, <footer>).
- Add sample content like navigation, about section, and contact info.

3. Apply CSS3 Styling

- Use flexbox or grid for layout.
- Add basic colors, padding, and fonts.
- Use media queries to adjust layout for desktop, tablet, and mobile.

4. Test Responsiveness

- Resize browser window.
- Test in mobile view (browser dev tools).

☑ Sample Code

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Responsive Website</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header>
    <h1>My Responsive Website</h1>
    <nav>
      <a href="#">Home</a>
      <a href="#">About</a>
```

```
    <a href="#">Services</a>
    <a href="#">Contact</a>
   </nav>
  </header>

  <main>
   <section class="intro">
    <h2>Welcome</h2>
    <p>This is a sample responsive website built using HTML5 and CSS3.</p>
   </section>

   <section class="grid">
    <div class="card">Card 1</div>
    <div class="card">Card 2</div>
    <div class="card">Card 3</div>
   </section>
  </main>

  <footer>
   <p>© 2025 My Responsive Site</p>
  </footer>
</body>
</html>
```

style.css

```
/* Base styles */
body {
  font-family: Arial, sans-serif;
  margin: 0;
  line-height: 1.6;
```

```
}
header {
  background: #333;
  color: #fff;
  padding: 1rem;
  text-align: center;
}
nav a {
  color: #fff;
  margin: 0 10px;
  text-decoration: none;
}
.intro {
  padding: 20px;
  text-align: center;
}
.grid {
  display: flex;
  gap: 10px;
  padding: 20px;
}
.card {
  flex: 1;
  background: #f4f4f4;
  padding: 20px;
  text-align: center;
  border-radius: 8px;
}
footer {
  background: #333;
  color: #fff;
```

```
  text-align: center;
  padding: 10px;
}

/* Responsive Design */
@media (max-width: 768px) {
  .grid {
    flex-direction: column;
  }
  nav {
    display: flex;
    flex-direction: column;
  }
  nav a {
    margin: 5px 0;
  }
}
```

Practice 3: Develop a Simple Client-Server Application using Node.js and Express

---

☑ Aim

To develop a basic client–server application using Node.js and the Express framework where the client sends requests and the server responds.

☑ Requirements

- Install Node.js (v14 or later).
- Install Express.js (via npm).

```
mkdir node-express-app
cd node-express-app
npm init -y
npm install express
```

☑ Steps

1. Create the Server (server.js)

```
// Import express module
const express = require('express');
const app = express();
const PORT = 3000;

// Middleware to parse JSON data
app.use(express.json());

// Root route
app.get('/', (req, res) => {
  res.send('Hello! This is a simple Node.js + Express server.');
});

// Example API route
app.get('/api/data', (req, res) => {
```

```
  res.json({ message: 'This is sample data from server', status: 'success' });
});

// Example POST route
app.post('/api/send', (req, res) => {
  const userData = req.body;
  res.json({ message: 'Data received successfully', data: userData });
});

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

---

2. Create the Client (Simple HTML Page client.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Client-Server Demo</title>
</head>
<body>
  <h2>Client-Server Application (Node.js + Express)</h2>
  <button onclick="getData()">Fetch Data</button>
  <button onclick="sendData()">Send Data</button>
```

```
  <p id="output"></p>

  <script>
    async function getData() {
      const response = await fetch('http://localhost:3000/api/data');
      const data = await response.json();
      document.getElementById('output').innerText = JSON.stringify(data);
    }

    async function sendData() {
      const response = await fetch('http://localhost:3000/api/send', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ name: 'Shreevasthava', role: 'Student' })
      });
      const data = await response.json();
      document.getElementById('output').innerText = JSON.stringify(data);
    }
  </script>
</body>
</html>
```

Practice 4: Develop a Dynamic SPA (Single Page Application) using React.js

---

☑ Aim

To develop a dynamic Single Page Application (SPA) using React.js, demonstrating component-based architecture, state management, and client-side routing.

☑ Requirements

- Node.js & npm installed.
- Basic knowledge of HTML, CSS, JavaScript.
- Code editor (VS Code recommended).

☑ Steps

1 Create React App

```
npx create-react-app my-spa
cd my-spa
npm start
```

- Opens default React app at http://localhost:3000.

---

2 Create Project Structure

```
my-spa/
├── src/
|   ├── components/
|   |   ├── Header.js
|   |   ├── Home.js
|   |   ├── About.js
|   |   └── Contact.js
```

```
|   ├── App.js
|   └── index.js
```

---

3 Create Components

Header.js

```
import React from 'react';
import { Link } from 'react-router-dom';

const Header = () => {
  return (
    <nav>
      <Link to="/">Home</Link> |
      <Link to="/about">About</Link> |
      <Link to="/contact">Contact</Link>
    </nav>
  );
}

export default Header;
```

Home.js

```
import React from 'react';
const Home = () => <h2>Welcome to Home Page</h2>;
export default Home;
```

About.js

```
import React from 'react';
const About = () => <h2>About Us</h2>;
export default About;
```

Contact.js

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

const Contact = () => {
  const [name, setName] = useState('');
  const [message, setMessage] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(Thank you ${name}, we received your message: "${message}");
    setName(''); setMessage('');
  }

  return (
    <form onSubmit={handleSubmit}>
      <input placeholder="Name" value={name} onChange={(e) => setName(e.target.value)} required
/>
      <textarea placeholder="Message" value={message} onChange={(e) =>
setMessage(e.target.value)} required />
      <button type="submit">Send</button>
    </form>
  );
}

export default Contact;
```

---

4 Configure Routing (App.js)

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Header from './components/Header';
import Home from './components/Home';
import About from './components/About';
import Contact from './components/Contact';

function App() {
  return (
    <Router>
      <Header />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </Router>
  );
}

export default App;
```

---

5 Add Basic Styling (App.css)

```
nav {
  background: #333;
  padding: 1rem;
}
nav a {
  color: white;
  margin: 0 10px;
  text-decoration: none;
}
form {
  display: flex;
  flex-direction: column;
  width: 300px;
  margin: 20px;
}
input, textarea {
  margin-bottom: 10px;
  padding: 5px;
}
button {
  padding: 5px;
  background-color: #333;
  color: white;
  border: none;
}
```

Let’s make Practice 5: To-Do List App using React.js with Redux into a structured lab note.

📝 Practice 5: Create a To-Do List Application using React.js with Redux

---

☑ Aim

To develop a dynamic to-do list application using React.js with Redux for state management, allowing users to add, delete, and mark tasks as completed.

☑ Requirements

- Node.js & npm installed.
- React.js knowledge.
- Redux for state management.

☑ Steps

1️⃣ Create React App

```
npx create-react-app redux-todo
cd redux-todo
npm install redux react-redux
npm start
```

- Opens default React app at http://localhost:3000.

---

2 Project Structure

```
redux-todo/
├── src/
|   ├── components/
|   |   ├── TodoList.js
|   |   └── TodoItem.js
|   ├── redux/
|   |   ├── actions.js
|   |   ├── reducers.js
|   |   └── store.js
|   ├── App.js
|   └── index.js
```

---

3 Setup Redux Store

actions.js

```
export const ADD_TODO = 'ADD_TODO';
export const TOGGLE_TODO = 'TOGGLE_TODO';
export const DELETE_TODO = 'DELETE_TODO';

export const addTodo = (task) => ({ type: ADD_TODO, payload: task });
export const toggleTodo = (id) => ({ type: TOGGLE_TODO, payload: id });
export const deleteTodo = (id) => ({ type: DELETE_TODO, payload: id });
```

reducers.js

```
import { ADD_TODO, TOGGLE_TODO, DELETE_TODO } from './actions';

const initialState = {
  todos: []
};

export const todoReducer = (state = initialState, action) => {
  switch(action.type) {
    case ADD_TODO:
      return { ...state, todos: [...state.todos, { id: Date.now(), task: action.payload, completed: false }] };
    case TOGGLE_TODO:
      return {
        ...state,
        todos: state.todos.map(todo =>
          todo.id === action.payload ? { ...todo, completed: !todo.completed } : todo
        )
      };
    case DELETE_TODO:
      return { ...state, todos: state.todos.filter(todo => todo.id !== action.payload) };
    default:
      return state;
  }
};
```

store.js

```
import { createStore } from 'redux';
import { todoReducer } from './reducers';
```

```
export const store = createStore(todoReducer);
```

---

4 Configure React with Redux (index.js)

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import { Provider } from 'react-redux';
import { store } from './redux/store';

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);
```

---

5 Create Components

TodoList.js

```
import React, { useState } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { addTodo, toggleTodo, deleteTodo } from '../redux/actions';
```

```
const TodoList = () => {
  const [task, setTask] = useState('');
  const todos = useSelector(state => state.todos);
  const dispatch = useDispatch();

  const handleAdd = () => {
    if(task.trim()) {
      dispatch(addTodo(task));
      setTask('');
    }
  }

  return (
    <div>
      <h2>Todo List</h2>
      <input value={task} onChange={(e) => setTask(e.target.value)} placeholder="Enter task"/>
      <button onClick={handleAdd}>Add</button>
      <ul>
        {todos.map(todo => (
          <li key={todo.id}>
            <span style={{ textDecoration: todo.completed ? 'line-through' : 'none', cursor: 'pointer' }}
                onClick={() => dispatch(toggleTodo(todo.id))}>
              {todo.task}
            </span>
            <button onClick={() => dispatch(deleteTodo(todo.id))}>Delete</button>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

```
export default TodoList;
```

App.js

```
import React from 'react';
import TodoList from './components/TodoList';

function App() {
  return (
    <div className="App">
      <TodoList />
    </div>
  );
}
```

Practice 6: Create an Interactive Form with Validation

☑ Aim

To create a responsive and interactive HTML form using JavaScript for validation and CSS3 for styling, ensuring user inputs are correct before submission.

☑ Requirements

- Basic knowledge of HTML, CSS, and JavaScript.
- Code editor (VS Code recommended).

- Browser for testing.

☑ Steps

1 HTML Form Structure

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Interactive Form</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="form-container">
    <h2>Sign Up Form</h2>
    <form id="signupForm">
      <label for="name">Name:</label>
      <input type="text" id="name" placeholder="Enter your name" required>

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

      <label for="password">Password:</label>
      <input type="password" id="password" placeholder="Enter password" required>

      <label for="confirmPassword">Confirm Password:</label>
      <input type="password" id="confirmPassword" placeholder="Confirm password" required>
```

```
    <button type="submit">Submit</button>
    <p id="errorMsg"></p>
   </form>
  </div>
<script src="script.js"></script>
</body>
</html>
```

---

2 CSS3 Styling (style.css)

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f9;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

.form-container {
  background: white;
  padding: 30px;
  border-radius: 10px;
  box-shadow: 0 0 15px rgba(0,0,0,0.2);
  width: 350px;
}
```

```
h2 {
  text-align: center;
  margin-bottom: 20px;
}

input {
  width: 100%;
  padding: 10px;
  margin-bottom: 15px;
  border-radius: 5px;
  border: 1px solid #ccc;
  transition: 0.3s;
}

input:focus {
  border-color: #333;
  box-shadow: 0 0 5px #aaa;
}

button {
  width: 100%;
  padding: 10px;
  background-color: #333;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: 0.3s;
}

button:hover {
```

```
  background-color: #555;
}

#errorMsg {
  color: red;
  font-size: 14px;
  text-align: center;
}
```

---

3 JavaScript Validation (script.js)

```
const form = document.getElementById('signupForm');
const errorMsg = document.getElementById('errorMsg');

form.addEventListener('submit', function(e) {
  e.preventDefault(); // Prevent form submission

  const name = document.getElementById('name').value.trim();
  const email = document.getElementById('email').value.trim();
  const password = document.getElementById('password').value.trim();
  const confirmPassword = document.getElementById('confirmPassword').value.trim();

  // Validation
  if(name === '' || email === '' || password === '' || confirmPassword === '') {
    errorMsg.textContent = 'All fields are required!';
    return;
  }
```

```
  const emailPattern = /^[^ ]+@[^ ]+\.[a-z]{2,3}$/;
  if(!email.match(emailPattern)) {
    errorMsg.textContent = 'Invalid email address!';
    return;
  }

  if(password.length < 6) {
    errorMsg.textContent = 'Password must be at least 6 characters!';
    return;
  }

  if(password !== confirmPassword) {
    errorMsg.textContent = 'Passwords do not match!';
    return;
  }

  // Success
  errorMsg.style.color = 'green';
  errorMsg.textContent = 'Form submitted successfully!';
  form.reset();
});
```