Chapter 14

Capstone Project

**Example Project Code**

Here’s a fully functional to-do list app using **HTML, CSS, and JavaScript**:

**HTML (Structure)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>To-Do List App</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h1>To-Do List</h1>

<div class="input-container">

<input type="text" id="taskInput" placeholder="Enter a task">

<button id="addTaskBtn">Add Task</button>

</div>

<ul id="taskList"></ul>

</div>

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

</body>

</html>

**CSS (Styling)**

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

text-align: center;

}

.container {

width: 50%;

margin: 50px auto;

background: white;

padding: 20px;

border-radius: 10px;

box-shadow: 0px 0px 10px #ccc;

}

.input-container {

display: flex;

justify-content: center;

gap: 10px;

}

input {

padding: 10px;

width: 70%;

border: 1px solid #ccc;

border-radius: 5px;

}

button {

padding: 10px;

background: #28a745;

color: white;

border: none;

cursor: pointer;

border-radius: 5px;

}

button:hover {

background: #218838;

}

ul {

list-style: none;

padding: 0;

}

li {

display: flex;

justify-content: space-between;

align-items: center;

background: #f9f9f9;

padding: 10px;

margin: 5px 0;

border-radius: 5px;

}

.completed {

text-decoration: line-through;

color: gray;

}

**JavaScript (Logic)**

// Select elements

const taskInput = document.getElementById("taskInput");

const addTaskBtn = document.getElementById("addTaskBtn");

const taskList = document.getElementById("taskList");

// Load tasks from local storage

document.addEventListener("DOMContentLoaded", loadTasks);

addTaskBtn.addEventListener("click", addTask);

function addTask() {

const taskText = taskInput.value.trim();

if (taskText === "") return alert("Please enter a task!");

const li = document.createElement("li");

li.innerHTML = `

<span>${taskText}</span>

<div>

<button class="complete-btn">✔</button>

<button class="delete-btn">✖</button>

</div>

`;

taskList.appendChild(li);

saveTasks();

// Event listeners for buttons

li.querySelector(".complete-btn").addEventListener("click", completeTask);

li.querySelector(".delete-btn").addEventListener("click", deleteTask);

taskInput.value = "";

}

function completeTask(event) {

event.target.parentElement.parentElement.classList.toggle("completed");

saveTasks();

}

function deleteTask(event) {

event.target.parentElement.parentElement.remove();

saveTasks();

}

// Save tasks to local storage

function saveTasks() {

const tasks = [];

document.querySelectorAll("#taskList li").forEach(li => {

tasks.push({

text: li.querySelector("span").innerText,

completed: li.classList.contains("completed")

});

});

localStorage.setItem("tasks", JSON.stringify(tasks));

}

// Load tasks from local storage

function loadTasks() {

const tasks = JSON.parse(localStorage.getItem("tasks")) || [];

tasks.forEach(task => {

const li = document.createElement("li");

li.innerHTML = `

<span>${task.text}</span>

<div>

<button class="complete-btn">✔</button>

<button class="delete-btn">✖</button>

</div>

`;

if (task.completed) li.classList.add("completed");

taskList.appendChild(li);

li.querySelector(".complete-btn").addEventListener("click", completeTask);

li.querySelector(".delete-btn").addEventListener("click", deleteTask);

});

}

**Project Structure**

File Organization

to-do-app/

│── index.html # Main HTML file (UI structure)

│── styles.css # CSS file (Styling)

│── script.js # JavaScript file (Logic & Functionality)

│── assets/ # (Optional) Images, icons, fonts

│── README.md # Documentation

**Code Structure (script.js)**

// Model: Handles data

const TaskModel = {

tasks: JSON.parse(localStorage.getItem("tasks")) || [],

addTask(taskText) { ... },

deleteTask(index) { ... },

toggleComplete(index) { ... },

saveTasks() { ... }

};

// View: Updates UI

const TaskView = {

renderTasks() { ... }

};

// Controller: Handles user actions

const TaskController = {

init() { ... },

handleAddTask() { ... },

handleDeleteTask(index) { ... },

handleToggleComplete(index) { ... }

};

// Initialize App

TaskController.init();

**Setting Up the Project**

To get started, set up a React project using Vite for better performance.

npx create-vite@latest my-app --template react

cd my-app

npm install

npm run dev

**Create Core UI Components**

Example: components/Navbar.jsx

import { Link } from "react-router-dom";

const Navbar = () => {

return (

<nav className="p-4 bg-blue-600 text-white flex justify-between">

<h1 className="text-xl font-bold">My App</h1>

<ul className="flex space-x-4">

<li><Link to="/">Home</Link></li>

<li><Link to="/dashboard">Dashboard</Link></li>

</ul>

</nav>

);

};

export default Navbar;

**Handle State Management**

Example: hooks/useAuth.js

import { useState } from "react";

export const useAuth = () => {

const [user, setUser] = useState(null);

const login = (username) => setUser({ name: username });

const logout = () => setUser(null);

return { user, login, logout };

};

**API Integration**

Use fetch or Axios for API calls.

Example: Fetching data from an API

import { useEffect, useState } from "react";

import axios from "axios";

const Dashboard = () => {

const [data, setData] = useState([]);

useEffect(() => {

axios.get("https://api.example.com/data")

.then((response) => setData(response.data))

.catch((error) => console.error("Error fetching data:", error));

}, []);

return (

<div>

<h2>Dashboard</h2>

<ul>

{data.map((item) => (

<li key={item.id}>{item.name}</li>

))}

</ul>

</div>

);

};

export default Dashboard;

**Add Routing**

Use React Router for navigation.

Install it:

npm install react-router-dom

Setup App.jsx

import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

import Navbar from "./components/Navbar";

import Dashboard from "./pages/Dashboard";

import Home from "./pages/Home";

function App() {

return (

<Router>

<Navbar />

<Routes>

<Route path="/" element={<Home />} />

<Route path="/dashboard" element={<Dashboard />} />

</Routes>

</Router>

);

}

export default App;

Example: Lazy Loading

import { lazy, Suspense } from "react";

const Dashboard = lazy(() => import("./pages/Dashboard"));

<Suspense fallback={<div>Loading...</div>}>

<Dashboard />

</Suspense>

**Deployment**

Example Deployment on Vercel:

npm install -g vercel

vercel

**Basic Fetch Request**

fetch("https://jsonplaceholder.typicode.com/posts/1")

.then(response => response.json()) // Convert response to JSON

.then(data => console.log(data)) // Handle data

.catch(error => console.error("Error fetching data:", error)); // Handle errors

**Fetching Data with Async/Await**

async function fetchPost() {

try {

const response = await fetch("https://jsonplaceholder.typicode.com/posts/1");

const data = await response.json();

console.log(data);

} catch (error) {

console.error("Error fetching data:", error);

}

}

fetchPost();

**Making a POST Request**

async function createPost() {

const postData = {

title: "My New Post",

body: "This is the content of my post.",

userId: 1

};

try {

const response = await fetch("https://jsonplaceholder.typicode.com/posts", {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify(postData)

});

const result = await response.json();

console.log("Post created:", result);

} catch (error) {

console.error("Error creating post:", error);

}

}

createPost();

**Making a PUT Request (Update)**

async function updatePost(postId) {

const updatedData = {

title: "Updated Title",

body: "Updated content.",

};

try {

const response = await fetch(`https://jsonplaceholder.typicode.com/posts/${postId}`, {

method: "PUT",

headers: { "Content-Type": "application/json" },

body: JSON.stringify(updatedData)

});

const result = await response.json();

console.log("Post updated:", result);

} catch (error) {

console.error("Error updating post:", error);

}

}

updatePost(1);

**Making a DELETE Request**

async function deletePost(postId) {

try {

const response = await fetch(`https://jsonplaceholder.typicode.com/posts/${postId}`, {

method: "DELETE"

});

if (response.ok) {

console.log("Post deleted successfully.");

}

} catch (error) {

console.error("Error deleting post:", error);

}

}

deletePost(1);

**Example of Error Handling**

async function fetchData() {

try {

const response = await fetch("https://jsonplaceholder.typicode.com/invalid-url");

if (!response.ok) {

throw new Error(`HTTP Error! Status: ${response.status}`);

}

const data = await response.json();

console.log(data);

} catch (error) {

console.error("Error fetching data:", error);

}

}

fetchData();

**API Integration in a Web Application**

**async\_index.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Currency Converter</title>

    <link rel="stylesheet" href="async\_style.css">

</head>

<body>

    <h2>Real-Time Currency Converter</h2>

    <div class="container">

        <input type="number" id="amount" placeholder="Enter amount" />

        <select id="fromCurrency">

            <option value="USD">USD - US Dollar</option>

            <option value="EUR">EUR - Euro</option>

            <option value="GBP">GBP - British Pound</option>

            <option value="INR">INR - Indian Rupee</option>

            <option value="JPY">JPY - Japanese Yen</option>

        </select>

        <select id="toCurrency">

            <option value="USD">USD - US Dollar</option>

            <option value="EUR">EUR - Euro</option>

            <option value="GBP">GBP - British Pound</option>

            <option value="INR">INR - Indian Rupee</option>

            <option value="JPY">JPY - Japanese Yen</option>

        </select>

        <button onclick="convertCurrency()">Convert</button>

        <p id="result"></p>

    </div>

    <script src="async\_script.js"></script>

</body>

</html>

**async\_script.js**

async function convertCurrency() {

    const amount = document.getElementById("amount").value;

    const fromCurrency = document.getElementById("fromCurrency").value;

    const toCurrency = document.getElementById("toCurrency").value;

    if (amount === "" || amount <= 0) {

        alert("Please enter a valid amount.");

        return;

    }

    const API\_KEY = "08ab5b200cd7c946e1d73645";  // Replace with your API key

 const API\_URL =

`https://v6.exchangerateapi.com/v6/${API\_KEY}/latest/${fromCurrency}`;

    try {

        const response = await fetch(API\_URL);

        if (!response.ok) {

            throw new Error(`Error fetching exchange rates: ${response.status}`);

        }

        const data = await response.json();

        const exchangeRate = data.conversion\_rates[toCurrency];

        const convertedAmount = (amount \* exchangeRate).toFixed(2);

        document.getElementById("result").innerHTML =

            `${amount} ${fromCurrency} = <strong>${convertedAmount} ${toCurrency}</strong>`;

    } catch (error) {

 document.getElementById("result").innerHTML = `<p style="color:red;">${error.message}</p>`;

    }

}

**async\_style.css**

body { font-family: Arial, sans-serif; text-align: center; margin: 20px; }

.container { max-width: 400px; margin: auto; padding: 20px; border: 1px solid #ccc; }

input, select, button { width: 100%; padding: 10px; margin: 5px 0; }

button { cursor: pointer; background-color: #007bff; color: white; border: none; }

#result { margin-top: 20px; font-size: 18px; }

**Setting Up React.js**

**Option 1: Using React CDN (For Small Projects)**

Add this inside an index.html file:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>React App</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/babel-standalone@6/babel.min.js"></script>

</head>

<body>

<div id="root"></div>

<script type="text/babel">

function App() {

return <h1>Hello, React!</h1>;

}

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

</script>

</body>

</html>

**Option 2: Using Create-React-App (Recommended)**

npx create-react-app my-app

cd my-app

npm start

* This sets up a **React project with Webpack, Babel, and development tools**.
* Open **http://localhost:3000/** in the browser.

**Setting Up Vue.js**

**Option 1: Using Vue CDN (For Quick Use)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Vue App</title>

<script src="https://cdn.jsdelivr.net/npm/vue@3/dist/vue.global.js"></script>

</head>

<body>

<div id="app">{{ message }}</div>

<script>

const app = Vue.createApp({

data() {

return { message: "Hello, Vue!" };

}

}).mount("#app");

</script>

</body>

</html>

**Option 2: Using Vue CLI (Recommended)**

npm install -g @vue/cli

vue create my-vue-app

cd my-vue-app

npm run serve

Open **http://localhost:8080/** in the browser.

**Setting Up Angular**

**Option 1: Using Angular CDN (Not Recommended)**

Angular is a **full-fledged framework** and does not work well with CDNs.

**Option 2: Using Angular CLI (Recommended)**

npm install -g @angular/cli

ng new my-angular-app

cd my-angular-app

ng serve

Open **http://localhost:4200/** in the browser.

**Setting Up Svelte**

**Option 1: Using Svelte via CDN (For Small Projects)**

Svelte does not have a CDN method. It **compiles to vanilla JavaScript**, so using the **official SvelteKit** setup is recommended.

**Option 2: Using SvelteKit (Recommended)**

npm create svelte@latest my-svelte-app

cd my-svelte-app

npm install

npm run dev

Open **http://localhost:5173/** in the browser.

**Building a Sample App in Each Framework**

Let’s build a **simple counter application** in **React, Vue, Angular, and Svelte**.

**Counter App in React**

import { useState } from "react";

function App() {

const [count, setCount] = useState(0);

return (

<div>

<h1>Counter: {count}</h1>

<button onClick={() => setCount(count + 1)}>Increase</button>

</div>

);

}

export default App;

**Counter App in Vue**

<template>

<div>

<h1>Counter: {{ count }}</h1>

<button @click="count++">Increase</button>

</div>

</template>

<script>

export default {

data() {

return { count: 0 };

}

};

</script>

**Counter App in Angular**

import { Component } from "@angular/core";

@Component({

selector: "app-root",

template: `

<h1>Counter: {{ count }}</h1>

<button (click)="increase()">Increase</button>

`,

})

export class AppComponent {

count = 0;

increase() {

this.count++;

}

}

**Counter App in Svelte**

<script>

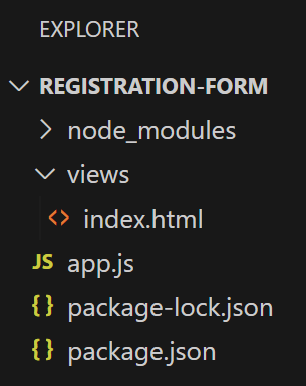
let count = 0;

</script>

<h1>Counter: {count}</h1>

<button on:click={() => count++}>Increase</button>

Project Structure:



***Figure 14.7:*** *Project Structure-MongoDB*

**Create folder views inside registration-form**

**Create file index.html inside views folder and paste the following code:**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Registration Form</title>

  <style>

    body {

      font-family: Arial, sans-serif;

    }

    .container {

      width: 300px;

      margin: 50px auto;

    }

    input {

      display: block;

      width: 100%;

      padding: 10px;

      margin-bottom: 10px;

    }

    button {

      padding: 10px;

      background-color: blue;

      color: white;

      border: none;

      cursor: pointer;

    }

  </style>

</head>

<body>

<div class="container">

  <h2>Register</h2>

  <form action="/register" method="POST">

    <input type="text" name="name" id="name" placeholder="Name" required>

    <input type="email" name="email" id="email" placeholder="Email" required>

    <input type="password" name="password" id="password" placeholder="Password" required>

    <button type="submit">Register</button>

  </form>

  <h2>Update User</h2>

  <form action="/update" method="POST">

    <input type="email" name="email" placeholder="Enter email to update" required>

    <input type="text" name="newName" placeholder="New Name" required>

    <input type="password" name="newPassword" placeholder="New Password" required>

    <button type="submit">Update</button>

  </form>

  <h2>Delete User</h2>

  <form action="/delete" method="POST">

    <input type="email" name="email" placeholder="Enter email to delete" required>

    <button type="submit">Delete</button>

  </form>

  <h2>All Users</h2>

  <a href="/users" target="\_blank">View all users</a>

</div>

</body>

</html>

**Create a file named app.js to handle the backend logic. Here is an example backend that connects to MongoDB and performs CRUD operations:   
Paste the following code**

const express = require('express');

const { MongoClient } = require('mongodb');

const path = require('path');

const app = express();

const port = 3000;

// Middleware to parse JSON bodies

app.use(express.json());

// Middleware to parse JSON and URL-encoded form data

app.use(express.urlencoded({ extended: true }));  // For form-urlencoded bodies

// Serve static HTML files

app.use(express.static(path.join(\_\_dirname, 'views')));

// MongoDB connection URI

const uri = "mongodb://localhost:27017"; // Replace with your MongoDB URI

const client = new MongoClient(uri);

// Serve the index.html on the root route

app.get('/', (req, res) => {

  res.sendFile(path.join(\_\_dirname, 'views', 'index.html'));

});

// POST endpoint to register a user

app.post('/register', async (req, res) => {

  const user = {

    name: req.body.name,

    email: req.body.email,

    password: req.body.password

  };

  console.log('Received user data:', user);

   try {

    await client.connect(); // Connect to MongoDB

    const database = client.db('testdb'); // Replace with your database name

 const usersCollection = database.collection('users'); // Replace with your collection name

    const result = await usersCollection.insertOne(user); // Insert the user

    res.status(201).send(`User registered successfully: ${result.insertedId}`);

  } catch (err) {

    console.error("Error registering user:", err);

    res.status(500).send('Error registering user');

  } finally {

    await client.close(); // Close the MongoDB connection

  }

});

// READ: Get all registered users

app.get('/users', async (req, res) => {

    try {

      await client.connect();

      const database = client.db('testdb'); // Replace with your database name

      const usersCollection = database.collection('users');

      const users = await usersCollection.find().toArray();

      res.json(users);

    } catch (err) {

      console.error('Error fetching users:', err);

      res.status(500).send('Error fetching users');

    }finally {

      await client.close(); // Close the MongoDB connection

    }

  });

    // UPDATE: Update user information

  app.post('/update', async (req, res) => {

    const filter = { email: req.body.email };

    const update = { $set: { name: req.body.newName, password: req.body.newPassword } };

    try {

      await client.connect(); // Reconnect to MongoDB

    const database = client.db('testdb'); // Replace with your database name

    const usersCollection = database.collection('users'); // Replace with your collection name

      const result = await usersCollection.updateOne(filter, update);

      if (result.matchedCount > 0) {

        res.send(`User with email ${req.body.email} updated successfully`);

      } else {

        res.send('No user found with the provided email');

      }

    } catch (err) {

      console.error('Error updating user:', err);

      res.status(500).send('Error updating user');

    }finally {

      await client.close(); // Close the MongoDB connection

    }

  });

  // DELETE: Delete a user

  app.post('/delete', async (req, res) => {

    const filter = { email: req.body.email };

    try {

      await client.connect(); // Reconnect to MongoDB

      const database = client.db('testdb'); // Replace with your database name

      const usersCollection = database.collection('users'); // Replace with your collection name

      const result = await usersCollection.deleteOne(filter);

      if (result.deletedCount > 0) {

        res.send(`User with email ${req.body.email} deleted successfully`);

      } else {

        res.send('No user found with the provided email');

      }

    } catch (err) {

      console.error('Error deleting user:', err);

      res.status(500).send('Error deleting user');

    }finally {

      await client.close(); // Close the MongoDB connection

    }

  });

// Error handling middleware

app.use((err, req, res, next) => {

  console.error(err.stack);

  res.status(500).send('Something went wrong!');

});

// Start the server

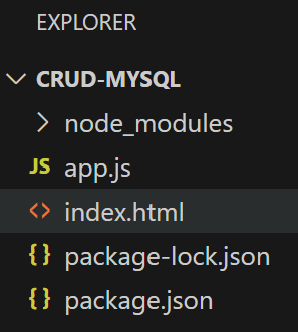
app.listen(port, () => {

  console.log(`Server running at http://localhost:${port}`);

});

**Example for MySQL Connectivity:**

Project Structure:



***Figure 14.8:*** *Project Structure- MySQL*

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>CRUD with MySQL and Node.js</title>

  <style>

    table, th, td {

      border: 1px solid black;

      border-collapse: collapse;

      padding: 10px;

    }

    th, td {

      text-align: left;

    }

  </style>

</head>

<body>

  <h1><p style="font-family:cambria"></p><p style="color:blue"> Users List </p></h1>

  <table id="usersTable" bgcolor="Silver">

    <thead>

      <tr>

        <th><p style="color:maroon">Name</p></th>

        <th><p style="color:maroon">Email</p></th>

        <th><p style="color:maroon">Actions</p></th>

      </tr>

    </thead>

    <tbody>

      <!-- Data will be injected here -->

    </tbody>

  </table>

  <h2><p style="font-family:cambria"></p><p style="color:blue">Add User</p></h2>

<form method="POST" action="/add"> <!-- Make sure method="POST" and action="/add" are correct -->

  <label><b><p style="color:maroon">Name:</p></b></label>

  <input type="text" name="name" required><br>

  <label><b><p style="color:maroon">Email:</p></b></label>

  <input type="email" name="email" required> <br><br>

  <button type="submit">Add User</button>

</form>

  <h2><p style="font-family:cambria"></p><p style="color:blue">Edit User</p></h2>

  <form id="editUserForm" style="display:none;">

    <input type="hidden" name="id">

    <label>Name:</label><br>

    <input type="text" name="name" required><br>

    <label>Email:</label><br>

    <input type="email" name="email" required><br>

    <button type="submit">Update User</button>

  </form>

  <script>

    document.addEventListener('DOMContentLoaded', function() {

      loadUsers();

      // Fetch and display users

      function loadUsers() {

        fetch('/users')

          .then(response => response.json())

          .then(data => {

            const tableBody = document.querySelector('#usersTable tbody');

            tableBody.innerHTML = '';

            data.forEach(user => {

              const row = document.createElement('tr');

              row.innerHTML = `

                <td>${user.name}</td>

                <td>${user.email}</td>

                <td>

                  <button onclick="editUser(${user.id}, '${user.name}', '${user.email}')">Edit</button>

                  <form style="display:inline;" action="/delete" method="POST">

                    <input type="hidden" name="id" value="${user.id}">

                    <button type="submit">Delete</button>

                  </form>

                </td>

              `;

              tableBody.appendChild(row);

            });

          });

      }

      // Add user

      document.getElementById('addUserForm').addEventListener('submit', function(e) {

        e.preventDefault();

        const formData = new FormData(this);

        fetch('/add', {

          method: 'POST',

          body: formData

        }).then(() => {

          this.reset();

          loadUsers();

        });

      });

      // Edit user

      window.editUser = function(id, name, email) {

        const editForm = document.getElementById('editUserForm');

        editForm.style.display = 'block';

        editForm.name.value = name;

        editForm.email.value = email;

        editForm.id.value = id;

      };

      // Update user

      document.getElementById('editUserForm').addEventListener('submit', function(e) {

        e.preventDefault();

        const formData = new FormData(this);

        fetch('/edit', {

          method: 'POST',

          body: formData

        }).then(() => {

          this.style.display = 'none';

          loadUsers();

        });

      });

    });

  </script>

</body>

</html>

**Create a file named app.js to handle the backend logic. Here is an example backend that connects to MySQL and performs CRUD operations:   
Paste the following code**

const express = require('express');

const mysql = require('mysql2');

const path = require('path'); // To serve the static HTML file

const app = express();

// Middleware to parse form data

app.use(express.urlencoded({ extended: true }));

// MySQL connection setup

const db = mysql.createConnection({

  host: 'localhost',

  user: 'root',

  password: 'rupali123',    //change as required

  database: 'mydatabase'

});

db.connect((err) => {

  if (err) throw err;

  console.log('MySQL Connected...');

});

// Route to serve index.html

app.get('/', (req, res) => {

  res.sendFile(path.join(\_\_dirname, 'index.html')); // Make sure index.html is in the same folder as app.js

});

// Read: Fetch all users

app.get('/users', (req, res) => {

    const sql = 'SELECT \* FROM users';

    db.query(sql, (err, results) => {

      if (err) throw err;

      res.json(results);

    });

  });

// Route to handle form submission

app.post('/add', (req, res) => {

  console.log('Received form data:', req.body);

  const { name, email } = req.body;

  if (!name || !email) {

    console.log('Name or email is missing');

    return res.status(400).send('Name and email are required');

  }

  const sql = 'INSERT INTO users (name, email) VALUES (?, ?)';

  db.query(sql, [name, email], (err, result) => {

    if (err) throw err;

    console.log('User added successfully, result:', result);

    res.redirect('/');

  });

});

// Update: Edit a user

app.post('/edit', (req, res) => {

    const { id, name, email } = req.body;

    const sql = 'UPDATE users SET name = ?, email = ? WHERE id = ?';

    db.query(sql, [name, email, id], (err) => {

      if (err) throw err;

      res.redirect('/');

    });

  });

  // Delete: Remove a user

  app.post('/delete', (req, res) => {

    const { id } = req.body;

    const sql = 'DELETE FROM users WHERE id = ?';

    db.query(sql, [id], (err) => {

      if (err) throw err;

      res.redirect('/');

    });

  });

// Start the server

app.listen(3000, () => {

  console.log('Server started on port 3000');

});

**Run the Application**

**Start MySQL**:

Ensure MySQL is running on your machine

**MySQL Database and Table Creation:**

Create database mydatabase;

use mydatabase;

Create table users(id int auto\_increment primary key, name varchar(20), email

varchar(25));

Start your Node.js server:  
node app.js

Server running at <http://localhost:3000>

Click the above link and check the result.

**Debugging JavaScript Code**

**Common Debugging Methods**

**Using console.log() (Basic Debugging)**

let num = 5;

console.log("Value of num:", num); // Debugging output

**Using debugger Statement**

function testFunction() {

let x = 10;

debugger; // Execution will pause here in the browser’s DevTools

x += 5;

console.log(x);

}

testFunction();

**Using console.log() in Node.js**

console.log("Server is running on port", process.env.PORT);

**Unit Testing in JavaScript (Jest)**

**Setting Up Jest**

Install **Jest** globally or in a project:

npm install --save-dev jest

Add a test script in **package.json**:

"scripts": {

"test": "jest"

}

**Writing a Simple Unit Test**

**📂 Project Structure**

backend-api/

│── src/

│ ├── math.js

│── tests/

│ ├── math.test.js

│── package.json

**📂 src/math.js (Function to Test)**

function add(a, b) {

return a + b;

}

function multiply(a, b) {

return a \* b;

}

module.exports = { add, multiply };

**📂 tests/math.test.js (Jest Test File)**

const { add, multiply } = require("../src/math");

test("adds 2 + 3 to equal 5", () => {

expect(add(2, 3)).toBe(5);

});

test("multiplies 3 \* 4 to equal 12", () => {

expect(multiply(3, 4)).toBe(12);

});

**Integration Testing with Supertest**

**Setting Up Supertest**

Install **Supertest** and **Jest**:

npm install --save-dev supertest jest

**Testing an Express API**

**📂 server.js (Sample Express API)**

const express = require("express");

const app = express();

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

res.json({ message: "pong" });

});

module.exports = app;

**📂 tests/server.test.js**

const request = require("supertest");

const app = require("../server");

test("GET /ping should return pong", async () => {

const response = await request(app).get("/ping");

expect(response.status).toBe(200);

expect(response.body.message).toBe("pong");

});

**Running the Test**

npm test

**End-to-End (E2E) Testing with Cypress**

**Installing Cypress**

npm install --save-dev cypress

**Writing a Cypress Test**

**📂 cypress/integration/sample.spec.js**

describe("Visit Homepage", () => {

it("should load homepage", () => {

cy.visit("http://localhost:3000");

cy.contains("Welcome");

});

});

**Running Cypress**

npx cypress open

**Remove unnecessary console logs**:

console.log("Debug message"); // Remove before deploying

**Use a linter** (ESLint) to fix syntax issues:

npx eslint . --fix

**Minify CSS & JavaScript**:

Use tools like **Terser (JS)** and **CSSNano (CSS)**.

npm install terser -g

terser script.js -o script.min.js

**UI/UX Enhancements**

**Add a Loading Indicator** for API calls:

function showLoader() {

document.getElementById("loader").style.display = "block";

}

function hideLoader() {

document.getElementById("loader").style.display = "none";

}

**Make the App Responsive** (Use CSS Media Queries):

@media (max-width: 600px) {

body { font-size: 14px; }

}

**Enhance Form Validation**:

function validateForm() {

let email = document.getElementById("email").value;

if (!email.includes("@")) {

alert("Invalid email address!");

return false;

}

return true;

}

**Performance Optimization**

**Enable Compression** in Express.js:

npm install compression

const compression = require("compression");

app.use(compression());

**Lazy Load Images in React/Vue**:

<img src="image.jpg" loading="lazy" alt="Image">

**Security Enhancements**

**Use Helmet.js for Security Headers (Express.js)**

npm install helmet

const helmet = require("helmet");

app.use(helmet());

**Sanitize User Input to Prevent XSS**

const sanitizeHtml = require("sanitize-html");

let cleanData = sanitizeHtml(req.body.comment);

**Use HTTPS for Secure Requests**:

Use **SSL/TLS certificates** on production servers.

**Error Handling and Logging**

**Catch Unhandled Errors in Express.js**

app.use((err, req, res, next) => {

console.error(err.stack);

res.status(500).send("Something went wrong!");

});

**Log Requests using Morgan**

npm install morgan

const morgan = require("morgan");

app.use(morgan("dev"));

**Deploying a Frontend Application**

**Deploying on Vercel (React/Vue/Static Sites)**

Install Vercel CLI:

npm install -g vercel

Login to Vercel:

vercel login

Deploy the app:

vercel --prod

Your app is now live at **https://your-project.vercel.app**.

**Deploying on Netlify**

Install Netlify CLI:

npm install -g netlify-cli

Login:

netlify login

Deploy:

netlify deploy --prod

The site will be hosted at **https://your-project.netlify.app**.

**Deploying on Heroku**

Install the **Heroku CLI**:

npm install -g heroku

Login:

heroku login

Create a new Heroku project:

heroku create my-api

Add a **Git Remote**:

git remote add heroku https://git.heroku.com/my-api.git

Deploy:

git push heroku main

Open the API:

heroku open

API is live at **https://my-api.herokuapp.com**.

**Testing Production API**

Use **Postman** or fetch() to test:

fetch("https://backend-api.onrender.com/tasks")

.then(res => res.json())

.then(data => console.log(data));