Chapter 10

Building a Simple Web Application

**Create following files:**

**I.** Right Click crud-app and make new folder **models**

**Ii.** Right click **models** and create new file **itemModel.js** and paste following code

**//**The **Model** will be an in-memory array to hold items.

// models/itemModel.js

let items = [];

// Get all items

const getAllItems = () => items;

// Get item by ID

const getItemById = (id) => items.find(item => item.id === id);

// Add a new item

const addItem = (item) => {

    items.push(item);

};

// Update an existing item

const updateItem = (id, updatedItem) => {

    const index = items.findIndex(item => item.id === id);

    if (index !== -1) {

        items[index] = updatedItem;

    }

};

// Delete an item

const deleteItem = (id) => {

    items = items.filter(item => item.id !== id);

};

module.exports = {

    getAllItems,

    getItemById,

    addItem,

    updateItem,

    deleteItem

};

**5:**

**I.** Right Click **crud-app** and make new folder **controllers**

**II.** Right click **controllers** and create new file **itemController.js** and paste following code

// controllers/itemController.js

const itemModel = require('../models/itemModel');

// Display all items

exports.getAllItems = (req, res) => {

    const items = itemModel.getAllItems();

    res.render('index', { items });

};

// Render form to add a new item

exports.renderAddItemForm = (req, res) => {

    res.render('addItem');

};

// Add a new item

exports.addItem = (req, res) => {

    const newItem = {

        id: Date.now().toString(),

        name: req.body.name,

        description: req.body.description

    };

    itemModel.addItem(newItem);

    res.redirect('/');

};

// Render form to edit an item

exports.renderEditItemForm = (req, res) => {

    const item = itemModel.getItemById(req.params.id);

    res.render('editItem', { item });

};

// Update an existing item

exports.updateItem = (req, res) => {

    const updatedItem = {

        id: req.body.id,

        name: req.body.name,

        description: req.body.description

    };

    itemModel.updateItem(req.body.id, updatedItem);

    res.redirect('/');

};

// Delete an item

exports.deleteItem = (req, res) => {

    itemModel.deleteItem(req.params.id);

    res.redirect('/');

};

**6:**

**I.** Right Click **crud-app** and make new folder **routes**

**II.** Right click **routes** and create new file **itemRoutes.js** and paste following code

// routes/itemRoutes.js

const express = require('express');

const router = express.Router();

const itemController = require('../controllers/itemController');

// List all items

router.get('/', itemController.getAllItems);

// Render form to add a new item

router.get('/add', itemController.renderAddItemForm);

// Add a new item

router.post('/add', itemController.addItem);

// Render form to edit an item

router.get('/edit/:id', itemController.renderEditItemForm);

// Update an item

router.post('/update', itemController.updateItem);

// Delete an item

router.get('/delete/:id', itemController.deleteItem);

module.exports = router;

**7: Views (EJS Templates)**

We’ll use **EJS** to create dynamic HTML views.

1. **index.ejs**: Displays a list of items.

**I.** Right click **crud-app** and create new folder **views.**

**Ii.** Right click **views** and create new file **index.ejs** and paste following code

<h1>Item List</h1>

<a href="/add">Add New Item</a>

<ul>

  <% items.forEach(item => { %>

    <li>

      <%= item.name %> - <%= item.description %>

      <a href="/edit/<%= item.id %>">Edit</a>

      <a href="/delete/<%= item.id %>">Delete</a>

    </li>

  <% }) %>

</ul>

**addItem.ejs**: Form to add a new item.

**II. Right click views and create new file addItem.ejs and paste following code**

<h1>Add Details</h1>

<div class="container">

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

  <div class="form-group">

  <label>Key:</label>

  <input type="text" name="name" class="form-control" required >

</div>

<div class="form-group">

  <label>Value:</label>

  <textarea name="description" class="form-control" required></textarea>

  </div>

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

</form>

</div>

**editItem.ejs**: Form to edit an existing item.

**III.** Right click **views** and create new file **editItem.ejs** and paste following code

<h1>Edit Details:</h1>

<div class="container">

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

  <div class="form-group">

  <input type="hidden" name="id" value="<%= item.id %>" class="form-control">

  <label>Key:</label>

  <input type="text" name="name" value="<%= item.name %>" class="form-control" required>

</div>

<div class="form-group">

  <label>Value:</label>

  <textarea name="description" class="form-control" required><%= item.description %></textarea>

</div>

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

</form>

</div>

**8: Server (server.js)**

The **server.js** file sets up the Express app and configures the middleware, views, and routes.

**I.** Right Click **crud-app** and make new file **server.js** and paste following code.

// server.js

const express = require('express');

const bodyParser = require('body-parser');

const path = require('path');

const itemRoutes = require('./routes/itemRoutes');

const app = express();

// Middleware to parse form data

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

// Set EJS as the view engine

app.set('view engine', 'ejs');

app.set('views', path.join(\_\_dirname, 'views'));

// Static files (for CSS, images, etc.)

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

// Use item routes

app.use('/', itemRoutes);

// Start the server

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

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

});

**9: CSS (Optional)**

Add basic CSS for styling in

**I.** Right Click **crud-app** and make new folder **public**

**II.** Right click **public** and make new file **styles.css and paste following code:**

body {

    font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman', serif;

    font-size: larger;

    font-weight: 200;

  }

  h1 {

    color: #530101;

  }

  h2 {

    color: #270153;

  }

  ul {

    list-style-type: none;

    padding: 0;

  }

  ul li {

    padding: 5px 0;

  }

  .form-group {

    margin-bottom: 20px;

  }

  .container {

    max-width: 500px;

    margin: 50px auto;

    padding: 20px;

    border: 1px solid #ccc;

    border-radius: 5px;

    background-color: #f9f9f9;

  }

 /\* input, select {

    width: 100%;

    padding: 10px;

    margin-bottom: 10px;

    box-sizing: border-box;

}\*/

form {

  max-width: 1200px;

  margin: 0 auto;

}

label {

  display: block;

  margin: 10px 0 5px;

  color:blue;

}

.form-control {

  width: 100%;

  padding: 8px;

  font-size: 16px;

  border: 1px solid #4b0202;

  border-radius: 4px;

}

button {

  padding: 10px 20px;

  background-color: #008080;

  color: rgb(12, 0, 0);

  border: none;

  border-radius: 4px;

  cursor: pointer;

  font-size: 16px;

  display: flex;

  justify-content: center;

  align-items: center;

}

Ensure explorer in VS Code looks like this, as shown in Figure 10.1:

***Figure 10.1:*** *crud-app Folder*

**10: Run the Application**

1. Start the Node.js application by typing following command in terminal:

**node server.js**

***.***

**Writing HTML and CSS for the Front End**

**About HTML**

HTML (**HyperText Markup Language**) defines the structure and content of a webpage. It uses elements enclosed in tags to describe parts of a webpage like headings, paragraphs, images, links, forms, and more.

**Basic HTML Structure**

Every HTML document has a standard structure:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>My Web Application</title>

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

</head>

<body>

<!-- Content goes here -->

</body>

</html>

**Key HTML Elements**

Here are common HTML tags used for structuring content:

**a. Paragraphs: <p> for regular text.**

<h1>Welcome to My Website</h1>

<p>This is **Headings and Paragraphs**

**Headings: <h1> to <h6> for titles or subtitles.**

an introductory paragraph.</p>

**Lists**

**Ordered List (<ol>) and Unordered List (<ul>) are used for creating lists.**

<ul>

<li>HTML</li>

<li>CSS</li>

<li>JavaScript</li>

</ul>

**Links and Images**

* Links: <a> tag to create hyperlinks.
* Images: <img> tag to embed images.

<a href="https://example.com">Visit Example</a>

<img src="logo.png" alt="Company Logo">

**Forms**

Forms collect user input via fields like text boxes, checkboxes, and buttons.

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

<label for="username">Username:</label>

<input type="text" id="username" name="username">

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

</form>

**About CSS**

CSS (**Cascading Style Sheets**) is used to style HTML elements, controlling layout, colors, fonts, and overall design.

**CSS Syntax**

CSS consists of selectors and declarations:

selector {

property: value;

}

**Example:**

body {

background-color: #f0f0f0;

font-family: Arial, sans-serif;

}

**Key CSS Features**

**a. Selectors**

**Element Selector:** Styles all occurrences of an element.

p {

color: blue;

}

**Class Selector:** Styles elements with a specific class.

.highlight {

background-color: yellow;

}

**ID Selector:** Styles a single element with a unique ID.

#header {

text-align: center;

}

**The Box Model**

**Example:**

.container {

padding: 10px;

border: 2px solid black;

margin: 20px;

}

**Layout Techniques**

**Flexbox**: For flexible, one-dimensional layouts.

.flex-container {

display: flex;

justify-content: center;

}

**Grid**: For two-dimensional layouts.

.grid-container {

display: grid;

grid-template-columns: repeat(3, 1fr);

gap: 10px;

}

**Responsive Design**

Use media queries to adapt the layout to different screen sizes.

@media (max-width: 768px) {

body {

font-size: 14px;

}

}

**Combining HTML and CSS**

Here’s a simple example demonstrating HTML and CSS integration:

**HTML (index.html):**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Simple Web Page</title>

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

</head>

<body>

<header id="header">

<h1>My Website</h1>

<nav>

<ul class="menu">

<li><a href="#">Home</a></li>

<li><a href="#">About</a></li>

<li><a href="#">Contact</a></li>

</ul>

</nav>

</header>

<main>

<section class="content">

<h2>About Us</h2>

<p>This website demonstrates how HTML and CSS work together.</p>

</section>

</main>

<footer>

<p>&copy; 2025 My Website</p>

</footer>

</body>

</html>

**CSS (css-styles.css):**

body {

margin: 0;

font-family: Arial, sans-serif;

background-color: #f9f9f9;

color: #333;

}

#header {

background-color: #6f0206;

color: white;

padding: 20px;

text-align: center;

}

.menu {

list-style: none;

display: flex;

justify-content: center;

padding: 0;

margin: 0;

}

.menu li {

margin: 0 15px;

}

.menu a {

color: white;

text-decoration: none;

}

.content {

max-width: 800px;

margin: 20px auto;

background: white;

padding: 20px;

border-radius: 10px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

}

footer {

text-align: center;

padding: 10px;

background: #6f0206;

color: white;

}

**Inline JavaScript**: Directly within an HTML tag using the onclick, onchange, or other attributes.

<button onclick="alert('Button clicked!')">Click Me</button>

**Internal JavaScript**: Inside a <script> tag within the HTML file.

<script>

document.querySelector('button').addEventListener('click', () => {

alert('Button clicked!');

});

</script>

**External JavaScript**: Linking an external .js file using the <script> tag.

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

**Enhancing the DOM with JavaScript**

**DOM Manipulation**

**Example:**

<button id="changeColor">Change Background</button>

<script>

document.getElementById('changeColor').addEventListener('click', () => {

document.body.style.backgroundColor = 'lightblue';

});

</script>

**Event Handling**

**Example:**

<input type="text" id="inputField" placeholder="Type here">

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

<script>

document.getElementById('inputField').addEventListener('input', (event) => {

document.getElementById('output').textContent = `You typed: ${event.target.value}`;

});

</script>

**Adding Animations**

**Example:**

<button id="animate">Animate Box</button>

<div id="box" style="width: 100px; height: 100px; background: red; position: relative;"></div>

<script>

document.getElementById('animate').addEventListener('click', () => {

let box = document.getElementById('box');

let position = 0;

let interval = setInterval(() => {

if (position >= 200) clearInterval(interval);

else {

position++;

box.style.left = position + 'px';

}

}, 5);

});

</script>

**Form Validation**

JavaScript validates user input before sending data to the server, ensuring better UX and reduced server load (A detailed example of this has already been discussed in Chapter 7).

**Example:**

<form id="contactForm">

<input type="email" id="email" placeholder="Enter your email">

<span id="error" style="color: red;"></span>

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

</form>

<script>

document.getElementById('contactForm').addEventListener('submit', (event) => {

const email = document.getElementById('email').value;

const error = document.getElementById('error');

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

event.preventDefault();

error.textContent = 'Invalid email address';

} else {

error.textContent = '';

}

});

</script>

**Dynamic Content Loading**

JavaScript, in conjunction with APIs or AJAX, dynamically loads new content without refreshing the page.

**Example:**

<button id="loadData">Load Data</button>

<ul id="dataList"></ul>

<script>

document.getElementById('loadData').addEventListener('click', () => {

fetch('https://jsonplaceholder.typicode.com/posts')

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

.then((data) => {

const list = document.getElementById('dataList');

list.innerHTML = '';

data.slice(0, 5).forEach((post) => {

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

li.textContent = post.title;

list.appendChild(li);

});

});

});

</script>

**Advanced Techniques**

**Integrating Libraries and Frameworks**

**Example with jQuery:**

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

<script>

$(document).ready(() => {

$('#loadData').click(() => {

$('#dataList').append('<li>Loaded with jQuery</li>');

});

});

</script>

**Handling APIs and JSON**

JavaScript interacts with APIs to fetch or send data. Commonly, data is exchanged in JSON format.

**Example:**

fetch('https://api.example.com/data')

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

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

.catch(error => console.error('Error:', error));

**Understanding HTML Forms**

**Example Form**

<form id="userForm">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

<label for="email">Email:</label>

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

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

</form>

**Accessing Form Elements**

Use the DOM to interact with forms and their elements.

const form = document.getElementById('userForm');

const usernameField = form.username; // Access by 'name' attribute

const emailField = document.getElementById('email'); // Access by ID

**Submit Event**

form.addEventListener('submit', (event) => {

event.preventDefault(); // Prevent form from refreshing the page

console.log('Username:', usernameField.value);

console.log('Email:', emailField.value);

});

**Input Event**

Triggered when the user modifies a field's value.

usernameField.addEventListener('input', (event) => {

console.log('Current Value:', event.target.value);

});

**Form Validation with JavaScript**

**Basic Validation**

form.addEventListener('submit', (event) => {

if (usernameField.value.trim() === '') {

event.preventDefault();

alert('Username is required!');

}

});

**b. Advanced Validation**

Use regular expressions to validate complex patterns like email or phone numbers.

const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

form.addEventListener('submit', (event) => {

if (!emailRegex.test(emailField.value)) {

event.preventDefault();

alert('Invalid email format!');

}

});

**Dynamic Form Manipulation**

JavaScript enables dynamic interactions, such as adding, removing, or modifying fields.

**a. Adding Fields Dynamically**

const addFieldBtn = document.createElement('button');

addFieldBtn.textContent = 'Add Field';

document.body.appendChild(addFieldBtn);

addFieldBtn.addEventListener('click', () => {

const newField = document.createElement('input');

newField.type = 'text';

newField.placeholder = 'New Field';

form.appendChild(newField);

});

**b. Enabling/Disabling Fields**

const toggleBtn = document.createElement('button');

toggleBtn.textContent = 'Enable/Disable Email';

document.body.appendChild(toggleBtn);

toggleBtn.addEventListener('click', () => {

emailField.disabled = !emailField.disabled;

});

**Submitting Forms with JavaScript**

JavaScript can submit forms programmatically using the submit() method.

form.addEventListener('submit', (event) => {

event.preventDefault();

console.log('Form Submitted via JavaScript!');

});

const submitBtn = document.createElement('button');

submitBtn.textContent = 'Submit Programmatically';

document.body.appendChild(submitBtn);

submitBtn.addEventListener('click', () => {

form.submit(); // Submit without clicking the original submit button

});

**Fetching Form Data**

Retrieve and process form data using the FormData object.

form.addEventListener('submit', (event) => {

event.preventDefault();

const formData = new FormData(form);

console.log('Form Data Entries:', Array.from(formData.entries()));

});

**Example: Full Form Handling** (Refer to the detailed example of this in Chapter 7).

<form id="registrationForm">

<label for="fullName">Full Name:</label>

<input type="text" id="fullName" name="fullName" required>

<label for="password">Password:</label>

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

<label for="age">Age:</label>

<input type="number" id="age" name="age">

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

</form>

<script>

const regForm = document.getElementById('registrationForm');

regForm.addEventListener('submit', (event) => {

event.preventDefault();

const fullName = regForm.fullName.value;

const password = regForm.password.value;

const age = regForm.age.value;

if (password.length < 6) {

alert('Password must be at least 6 characters long.');

return;

}

alert(`Registration Successful!

Name: ${fullName}

Age: ${age}`);

});

</script>

**Example: Local Storage**

// Add data

localStorage.setItem('username', 'JohnDoe');

localStorage.setItem('age', '25');

// Retrieve data

console.log(localStorage.getItem('username')); // Output: JohnDoe

console.log(localStorage.getItem('age')); // Output: 25

// Remove a specific item

localStorage.removeItem('age');

// Clear all data

localStorage.clear();

**Example: Session Storage**

// Add data

sessionStorage.setItem('sessionId', '12345');

// Retrieve data

console.log(sessionStorage.getItem('sessionId')); // Output: 12345

// Clear session storage

sessionStorage.clear();

**Storing Objects**

const user = { name: 'Alice', age: 30 };

// Save the object

localStorage.setItem('user', JSON.stringify(user));

// Retrieve and parse the object

const storedUser = JSON.parse(localStorage.getItem('user'));

console.log(storedUser.name); // Output: Alice

**Example: Listening to Storage Changes**

window.addEventListener('storage', (event) => {

console.log('Key Changed:', event.key);

console.log('Old Value:', event.oldValue);

console.log('New Value:', event.newValue);

});

**Practical Example**

**Saving User Preferences**

// Save user preference

const saveThemePreference = (theme) => {

localStorage.setItem('theme', theme);

document.body.className = theme;

};

// Load user preference

const loadThemePreference = () => {

const theme = localStorage.getItem('theme');

if (theme) {

document.body.className = theme;

}

};

// Event listener for theme change

document.getElementById('themeSelector').addEventListener('change', (event) => {

saveThemePreference(event.target.value);

});

// Load theme preference on page load

loadThemePreference();

**HTML**

<select id="themeSelector">

<option value="light">Light</option>

<option value="dark">Dark</option>

</select>

**The fetch() API**

**Basic Syntax**

fetch(url, options)

.then(response => {

// Handle the response object

})

.catch(error => {

// Handle any errors

});

**Performing a GET Request**

A GET request is used to retrieve data.

**Example**

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

.then(response => {

if (!response.ok) {

throw new Error('Network response was not ok');

}

return response.json(); // Parse JSON response

})

.then(data => {

console.log('Data fetched:', data);

})

.catch(error => {

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

});

**Performing a POST Request**

A POST request is used to send data to a server.

**Example**

fetch('https://jsonplaceholder.typicode.com/posts', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

title: 'New Post',

body: 'This is a new post content',

userId: 1,

}),

})

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

.then(data => {

console.log('Post created:', data);

})

.catch(error => {

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

});

**Using Async/Await with fetch()**

To make the code cleaner and more synchronous, you can use async/await.

**Example**

async function fetchData() {

try {

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

if (!response.ok) {

throw new Error('Network response was not ok');

}

const data = await response.json();

console.log('Fetched Data:', data);

} catch (error) {

console.error('Error:', error);

}

}

fetchData();

**Handling Errors**

**Example**

fetch('https://jsonplaceholder.typicode.com/invalid-endpoint')

.then(response => {

if (!response.ok) {

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

}

return response.json();

})

.catch(error => {

console.error('Fetch failed:', error.message);

});

**Working with Headers**

Custom headers are often required for authentication or content negotiation.

**Example**

fetch('https://api.example.com/data', {

method: 'GET',

headers: {

'Authorization': 'Bearer YOUR\_ACCESS\_TOKEN',

'Accept': 'application/json',

},

})

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

.then(data => console.log('Data:', data))

.catch(error => console.error('Error:', error));

**Practical Example: Fetching Weather Data (A detailed example has already been discussed in Chapter 9)**

**HTML**

<div>

<h1>Weather App</h1>

<input type="text" id="city" placeholder="Enter city name">

<button id="fetchWeather">Get Weather</button>

<p id="weatherOutput"></p>

</div>

**JavaScript**

document.getElementById('fetchWeather').addEventListener('click', async () => {

const city = document.getElementById('city').value;

const apiKey = 'YOUR\_API\_KEY';

const url = `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}&units=metric`;

try {

const response = await fetch(url);

if (!response.ok) {

throw new Error('City not found');

}

const data = await response.json();

document.getElementById('weatherOutput').textContent =

`Weather in ${data.name}: ${data.main.temp}°C`;

} catch (error) {

document.getElementById('weatherOutput').textContent = error.message;

}

});

**Fetching Data for Dynamic Content**

**Example: Displaying API Data**

**HTML:**

<div id="userContainer">

<h3>Users</h3>

<ul id="userList"></ul>

</div>

**JavaScript:**

fetch('https://jsonplaceholder.typicode.com/users')

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

.then(users => {

const userList = document.getElementById('userList');

users.forEach(user => {

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

li.textContent = user.name;

userList.appendChild(li);

});

})

.catch(error => console.error('Error fetching users:', error));

**Reacting to User Actions**

JavaScript enables dynamic updates based on user interactions, such as button clicks or form submissions.

**Example: Updating Content on Button Click**

**HTML:**

<div id="messageContainer"></div>

<button id="updateButton">Click Me</button>

**JavaScript:**

document.getElementById('updateButton').addEventListener('click', () => {

document.getElementById('messageContainer').textContent = 'Button clicked! Dynamic content updated.';

});

**Dynamic Styling**

You can dynamically update styles based on user interactions or data.

**Example: Highlighting a Selected Item**

**HTML:**

<ul id="itemList">

<li>Item 1</li>

<li>Item 2</li>

<li>Item 3</li>

</ul>

**JavaScript:**

document.querySelectorAll('#itemList li').forEach(item => {

item.addEventListener('click', () => {

document.querySelectorAll('#itemList li').forEach(li => li.style.background = '');

item.style.background = 'yellow';

});

});

**Using Templates for Dynamic Content**

Templates allow you to reuse HTML structures for dynamically inserted content.

**Example: Using a Template Element**

**HTML:**

<template id="userTemplate">

<div class="user-card">

<h4 class="user-name"></h4>

<p class="user-email"></p>

</div>

</template>

<div id="userCards"></div>

**JavaScript:**

fetch('https://jsonplaceholder.typicode.com/users')

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

.then(users => {

const template = document.getElementById('userTemplate');

const container = document.getElementById('userCards');

users.forEach(user => {

const clone = template.content.cloneNode(true);

clone.querySelector('.user-name').textContent = user.name;

clone.querySelector('.user-email').textContent = user.email;

container.appendChild(clone);

});

});

**Writing Effective Tests**

**Unit Testing**

Test individual functions or methods for correctness.

**Example:**

function add(a, b) {

return a + b;

}

module.exports = add;

// Test using Jest

const add = require('./add');

test('adds 1 + 2 to equal 3', () => {

expect(add(1, 2)).toBe(3);

});

**Example using Cypress:**

describe('Login Test', () => {

it('should log in successfully', () => {

cy.visit('http://localhost:3000/login');

cy.get('#username').type('testuser');

cy.get('#password').type('password123');

cy.get('#loginButton').click();

cy.url().should('include', '/dashboard');

});

});