Test Documents:

Unit Testing

Unit tests are essential to validate the functionality of the JavaScript functions in isolation. For the JavaScript in out application, we used jest to create tests for the functions like identifyPlant, checking file uploads and functions in the login page.

* Test **identifyPlant** function by mocking the HTTP requests to the plant identification API and verifying that it handles both success and error responses correctly.
* Test for the load indicator function to check if it properly dis plays load indicator when wating for the response from the API

Test code: // \_\_tests\_\_/app.test.js

require('jest-fetch-mock').enableMocks();

const fs = require('fs');

const path = require('path');

const html = fs.readFileSync(path.resolve(\_\_dirname, './plant.html'), 'utf8');

global.fetch = require('jest-fetch-mock');

function setupFileReaderMock() {

    class MockFileReader {

        onload = null;

        readAsDataURL(blob) {

            if (this.onload) {

                this.onload({ target: { result: 'data:image/jpeg;base64,' + btoa('fake data') } });

            }

        }

        addEventListener(event, handler) {

            this.onload = handler;

        }

    }

    global.FileReader = MockFileReader;

}

describe('Plant Identification Page', () => {

    beforeEach(() => {

        document.documentElement.innerHTML = html.toString();

        setupFileReaderMock();

        require('./script');  // Load your script after the DOM setup

        document.getElementById("loadingIndicator").style.display = 'none';

    });

    afterEach(() => {

        jest.resetModules();

        fetch.resetMocks();

    });

    it('shows loading indicator during plant identification', async () => {

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

        fetch.mockResponseOnce(JSON.stringify({ success: true }));  // Mocking a successful API call

        document.getElementById('uploadInput').dispatchEvent(new Event('change', { bubbles: true }));

        expect(loadingIndicator.style.display).toBe('block');

    });

    it('handles API error correctly', async () => {

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

        fetch.mockRejectOnce(new Error('Failed to fetch'));  // Mocking a fetch failure

        await new Promise(resolve => {

            document.getElementById('uploadInput').addEventListener('change', resolve);

            document.getElementById('uploadInput').dispatchEvent(new Event('change', { bubbles: true }));

        });

        // Ensure that the error handling completes

        await new Promise(resolve => setTimeout(resolve, 0));

        // Check for correct error handling and UI updates

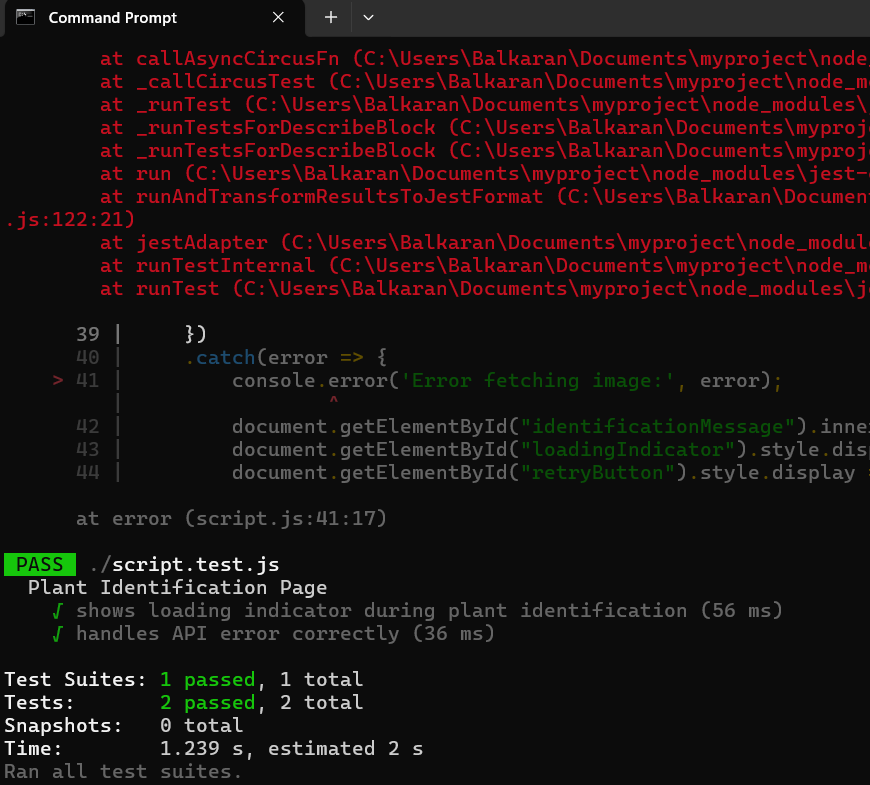
        expect(identificationMessage.innerText).toContain('Error fetching image: Failed to fetch');

        expect(document.getElementById("loadingIndicator").style.display).toBe("none");

        expect(document.getElementById("retryButton").style.display).toBe("block");

    });

});

Result: 

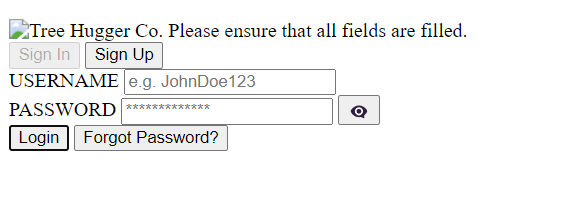
Unit Testing for login page to check if functions in login page are working properly

This following code is testing the functionality of an authentication login page.

1.**Switches to Sign In**:

* + Verifies that clicking the "Sign In" button displays the sign-in form and hides the sign-up form.

1. **Switches to Sign Up**:
   * Verifies that clicking the "Sign Up" button displays the sign-up form and hides the sign-in form.
2. **Handles login submission with empty fields**:
   * Simulates a scenario where the user tries to log in without filling in any fields.
   * Ensures that an appropriate message is displayed when attempting to submit empty fields.



1. **Handles successful login**:
   * Mocks a successful response from a server after attempting a login.
   * Sets up input values for username and password.
   * Simulates a login button click.
   * Checks if the **window.location.assign** method is called with the correct URL after successful login.
   * Uses a spy to monitor the **window.location.assign** method to ensure it's called correctly.
   * Restores the spy after the test to maintain test isolation.
2. describe('Authentication Form Tests', () => {
3. beforeAll(() => {
4. // Setup HTML elements in the document
5. document.documentElement.innerHTML = `
6. <div id="signInForm" style="display: none;"></div>
7. <div id="signUpForm" style="display: none;"></div>
8. <div id="mainResponse"></div>
9. <button id="signInBtn">Sign In</button>
10. <button id="signUpBtn">Sign Up</button>
11. <input id="signInUsernameInput">
12. <input id="signInPasswordInput">
13. <input id="signUpUsernameInput">
14. <input id="signUpPasswordInput">
15. <input id="signUpEmailInput">
16. <button id="login">Login</button>
17. `;
18. require('./login-source.js');
19. });
20. beforeEach(() => {
21. fetch.resetMocks();
22. });
23. test('Switches to Sign In', () => {
24. document.getElementById('signInBtn').click();
25. expect(document.getElementById('signInForm').style.display).toBe('block');
26. expect(document.getElementById('signUpForm').style.display).toBe('none');
27. });
28. test('Switches to Sign Up', () => {
29. document.getElementById('signUpBtn').click();
30. expect(document.getElementById('signUpForm').style.display).toBe('block');
31. expect(document.getElementById('signInForm').style.display).toBe('none');
32. });
33. test('Handles login submission with empty fields', () => {
34. document.getElementById('signInBtn').click(); // Ensure sign-in form is visible
35. document.getElementById('login').click(); // Simulate login button click
36. expect(document.getElementById('mainResponse').textContent).toContain('Please ensure that all fields are filled.');
37. });
38. test('Handles successful login', async () => {
39. fetch.mockResponseOnce(JSON.stringify({ success: true }), { status: 200 });
40. document.getElementById('signInUsernameInput').value = 'user';
41. document.getElementById('signInPasswordInput').value = 'pass';
43. // Spy on window.location.assign method
44. const assignSpy = jest.spyOn(window.location, 'assign');
46. // Assuming the login button has an event listener that triggers an async operation
47. document.getElementById('login').click();
48. await new Promise(r => setTimeout(r, 50)); // Wait for any asynchronous operations to complete
50. // Check if window.location.assign is called with the correct URL
51. expect(assignSpy).toHaveBeenCalledWith('http://localhost:8080/index.html');
53. // Restore the spy
54. assignSpy.mockRestore();
55. });
56. });

Accuracy Testing:

The main test was to test the accuracy of the identified plant given by the API.

For this testing, uploaded the image of 50 different plants and out of it API correctly identified 44 plant images giving the accuracy of 88%.