HANDLING CODE REVIEW FEEDBACK:

ADDING FEEDBACK AS COMMENTS:

**CODE:**

const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {

event.preventDefault();

try {

const response = await fetch(`${API\_ENDPOINT}/organisations`, {

method: 'POST',

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

body: JSON.stringify({ name: organisationName, user\_name: userName, email: userEmail, password: userPassword }),

});

if (!response.ok) {

throw new Error(`Sign-up failed with status ${response.status}`);

// Feedback: Consider providing a more specific error message for failed sign-up.

}

console.log('Sign-up successful');

const data = await response.json();

localStorage.setItem('authToken', data.token);

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

navigate("/dashboard");

// Feedback: Consider adding comments to explain the purpose of each step in the code.

} catch (error) {

console.error('Sign-up failed:', error);

// Feedback: Consider displaying a user-friendly error message for failed sign-up.

// Feedback: Consider handling different types of errors separately for better error handling.

}

};

**UPDATED CODE:**

const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {

event.preventDefault();

try {

const response = await fetch(`${API\_ENDPOINT}/organisations`, {

method: 'POST',

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

body: JSON.stringify({ name: organisationName, user\_name: userName, email: userEmail, password: userPassword }),

});

if (!response.ok) {

// Feedback: Provide a more specific error message for failed sign-up.

throw new Error(`Sign-up failed. Please try again later. Status: ${response.status}`);

}

console.log('Sign-up successful');

const data = await response.json();

localStorage.setItem('authToken', data.token);

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

// Feedback: Add comments to explain the purpose of each step.

navigate("/dashboard");

} catch (error) {

console.error('Sign-up failed:', error);

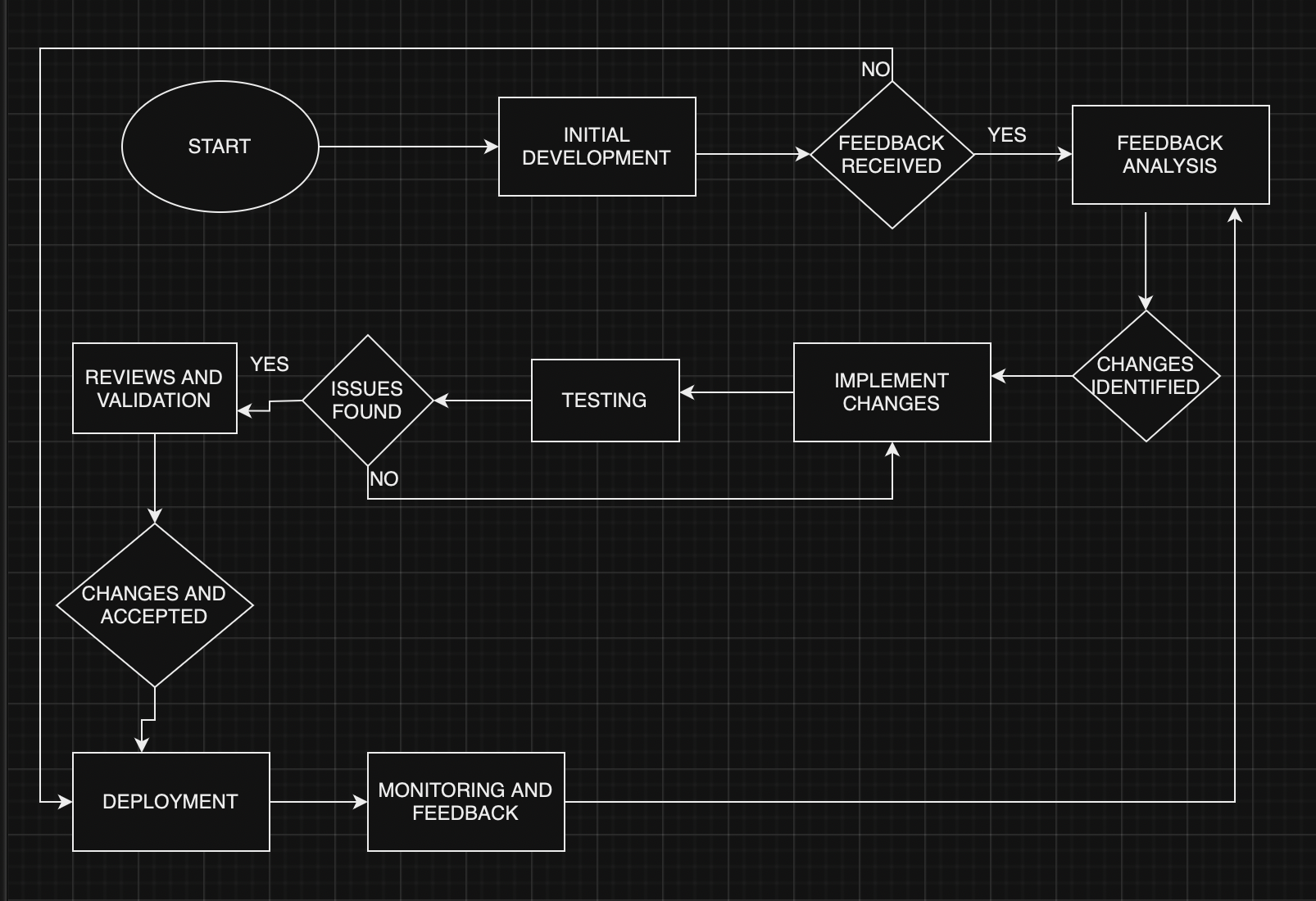
// Feedback: Display a user-friendly error message for failed sign-up.

alert('Sign-up failed. Please try again.');

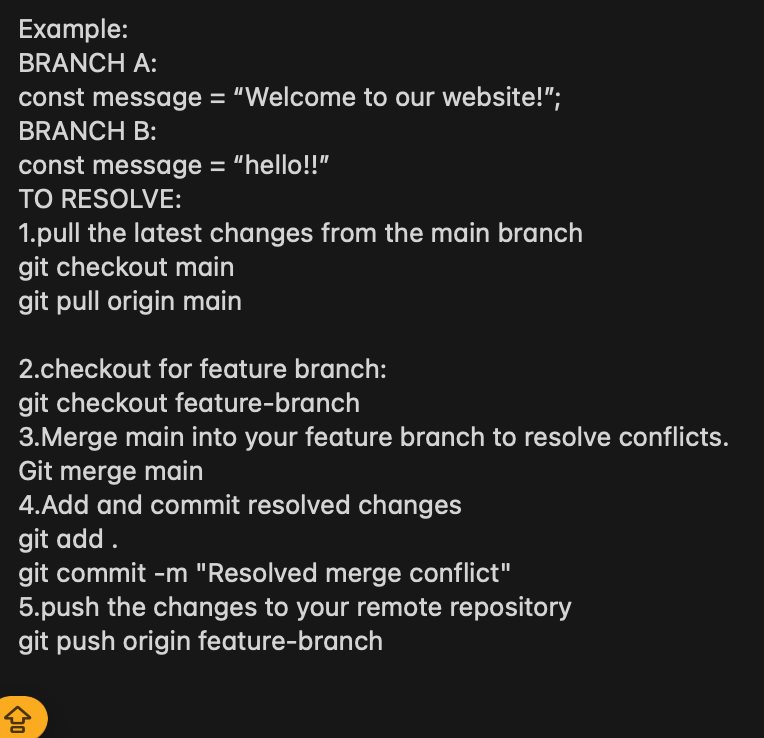
// Feedback: Handle different types of errors separately for better error handling.

}

};

ITERATIVE DEVELOPMENT PROCESS:

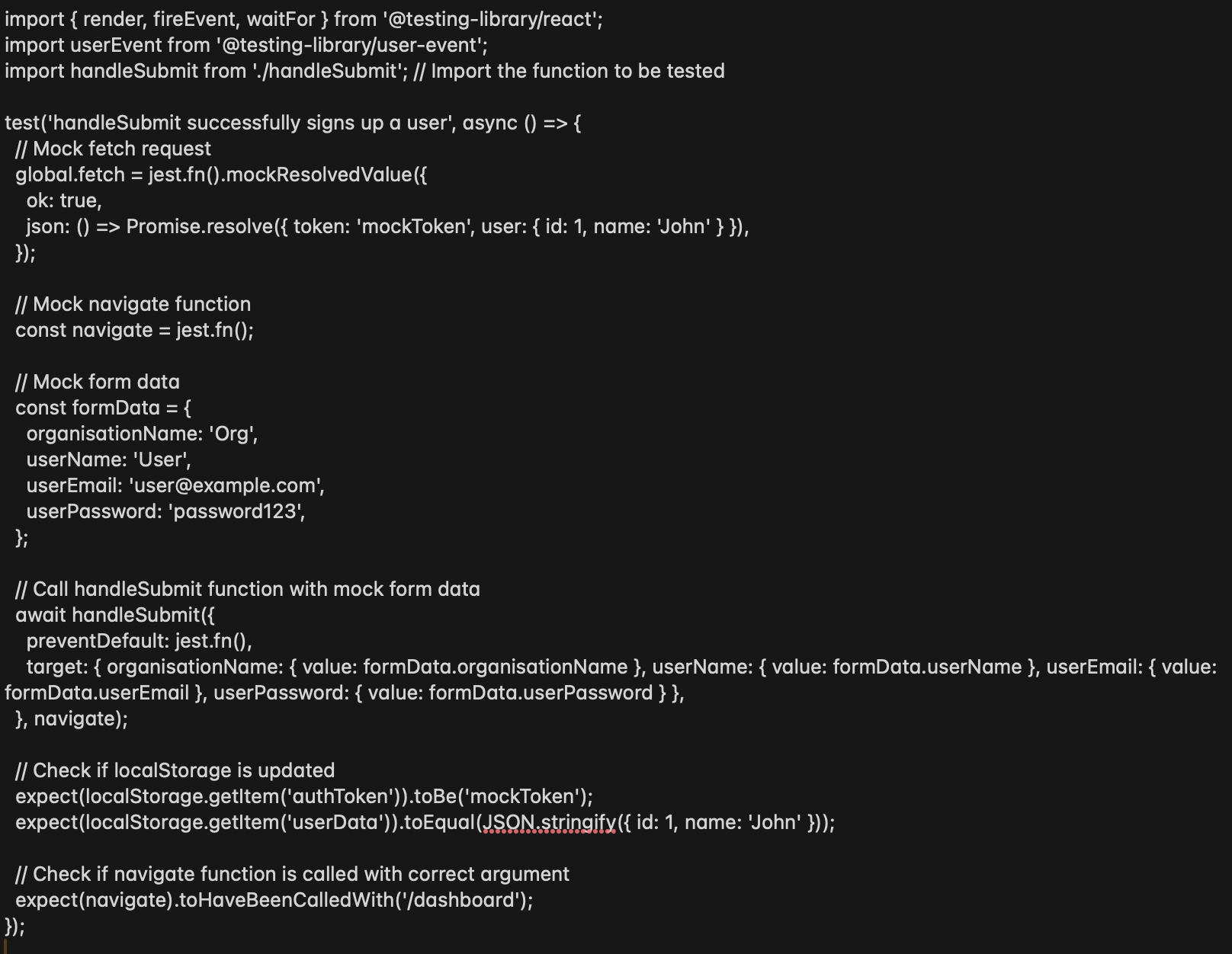
RESOLVING MERGE CONFLICTS:

If two branches modify the same lines of code, a merge conflict can occur.

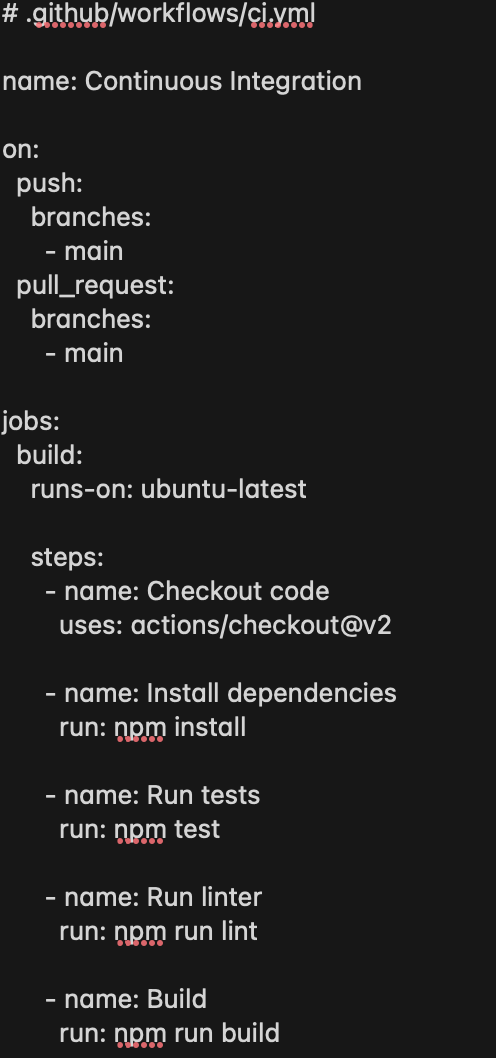
CI/CD Integration:

AUTOMATED TESTS:

To ensure code quality and prevent regressions, automated tests are crucial. In the context of React applications like this one, unit tests, integration tests, and end-to-end tests can be implemented.

CODE:

Quality checks also need to be performed.Quality checks can include linting, code formatting, and code coverage analysis. Tools like ESLint, Prettier, and Jest can be integrated into the CI/CD pipeline to enforce code quality standards.



EXAMPLE: