Testing Document

Edit New page

Jump to bottom

Lloyd Creighton edited this page 1 hour ago · 2 revisions

1. Introduction

This document outlines the testing policy for our project, including the automated testing framework, procedures, and tools used to ensure the quality and reliability of our software.

2. Testing Tools

We have chosen Playwright and Jest for our testing framework due to the following reasons:

2.1 Playwright

Cross-Browser Testing: Playwright supports multiple browsers, including Chromium, Firefox, and WebKit, enabling us to ensure that our application works seamlessly across different platforms. Automated Interaction: It allows us to automate user interactions with our application, making it easier to perform end-to-end testing. Fast and Reliable: Playwright is designed for speed and reliability, providing a stable testing environment.

2.2 Jest

Unit Testing: Jest is widely used for unit testing JavaScript applications, allowing us to write simple and effective tests for our components. Easy Integration: Jest integrates well with other tools, such as Playwright, and supports features like snapshot testing and mocking. Active Community: Being a widely adopted testing framework, Jest has a strong community and extensive documentation, making it easier to find support and resources.

3. Continuous Integration (CI)

To automate our testing and deployment process, we utilize GitHub Actions for several reasons:

Seamless Integration: GitHub Actions is built into GitHub, allowing for automatic triggers o events like pushes and pull requests.

Customization: We can create tailored workflows in YAML, easily modifying our CI/CD processes as needed.

Cost-Effective: It's free for public repositories and provides generous build minutes for private repositories.

Community Support: A vast marketplace of pre-built actions and strong documentation simplifies integration and troubleshooting.

Collaboration: CI results are accessible directly within our repository, improving visibility and team collaboration.

Scalability: Easily scalable workflows allow us to expand tests and deployment strategies as our project grows.

4. Testing Procedure

The following procedures outline our approach to testing:

4.1 Test Case Development

Test cases are developed in alignment with the project's requirements and specifications. Each feature or user story will have corresponding test cases to ensure high coverage.

4.2 Automated Testing

Automated tests are implemented using Playwright for end-to-end testing and Jest for unit testing. All tests are executed in the CI/CD pipeline on every push to the main branch.

4.3 Test Reports

Test reports are generated after each CI run, detailing the results of the automated tests. Reports are accessible via GitHub Actions for easy review.

4.4 Code Coverage

We utilize code coverage tools integrated with Jest to ensure that our tests cover a significant portion of the codebase. A coverage threshold is set to ensure that any new code maintains or improves coverage levels.

Link to Testing CI

Pages 11

Find a page...

+ Add a custom sidebar

Clone this wiki locally

```
https://github.com/COS301-SE-2024/IoT-DIRfram.wiki.git
```