Sprint 2 Testing Plan

This project documentation refers to the outlining of a comprehensive strategy for verifying and validating the software developed during the second sprint of the Condo Management System project. In order to meet the specific requirements of the project. It is crucial to incorporate a multi-level testing strategy that includes unit tests, integration tests, system tests, and a combination of automated and manual testing processes.

Testing Tool Selection:

Firstly, it is essential to decide on the tools that will be used at each level of testing.

As the stack of our Condo Management Systems project, we selected to use React for the front-end of the website application, alongside with HTML/CSS/ JavaScript. Moreover, we will be using Python for the back-end and MongoDB as our NoSQL database. The API for the back-end is still to be determined.

Unit Testing: It is crucial to use a tool that supports isolation and mocking capabilities. It will allow for testing individual methods or components in isolation from the rest of the system.

For the front-end of the application, Jest is a popular testing framework that works well with React and vanilla JavaScript. Jest is widely used for its simplicity and support for React applications. It can handle testing of JavaScript code, including asynchronous code, and integrates well with Babel.

For the Python back-end, pytest supports fixtures, which can be used to manage test dependencies, and it's known for its ease of use and ability to handle asynchronous code. There is also unittest, which is a unit testing framework that comes with Python's standard library. It's inspired by JUnit and has a similar feature set, including test organization, test fixtures, and test runners.

Integration Testing: In order to effectively test interactions between the Python backend, MongoDB database, and any external APIs that will be used, integration tools will be employed.

- Postman is an ideal tool for testing API endpoints, which will ensure that the backend correctly handles requests and responses. It can be used to automate tests for the RESTful services and to verify the integration between the frontend and backend.

- PyTest with pytest-mongodb and requests will be a testing framework for Python. With the pytest-mongodb plugin, we can mock or spin up a MongoDB instance for testing

purposes. The requests library or aiohttp (for asynchronous applications) can be used for testing API calls within Python tests, simulating integration between the backend and external services.

- MongoDB Memory Server is a Node.js package that spins up a real MongoDB server programmatically, in memory, for testing purposes. It is useful when for running tests that interact with the database without affecting the production or development databases.

System Testing: For system testing, where the goal is to validate the integrated system's functionality, performance, and behavior under production-like circumstances, there are several tools that can be effective.

- Selenium is a widely used tool for automating web browsers. It can simulate user interactions with the React application, testing the system as a whole from the user's perspective. It can work well for testing complex user flows and interactions across the application.

- Cypress is an all-in-one testing framework that's gained popularity for its ease of use and support for modern web development frameworks, including React. Cypress can be used for writing end-to-end tests, simulating real user interactions with the application. It provides a modern and developer-friendly alternative to Selenium, with features such as automatic waiting, real-time reloads, and video recording of test runs.

Automation: For Continuous Integration (CI), a service like Jenkins, Travis CI, or GitHub Actions to automate the execution of tests upon every commit or merge request will be employed. This will ensure that tests are run frequently and consistently.

- Jenkins is an open-source automation server that provides hundreds of plugins to support building, deploying, and automating any project. It is highly customizable for complex workflows, supports a wide range of plugins for integration with various development tools, and can be used for both CI and Continuous Deployment (CD).

- GitHub Actions is an automation tool that enables you to automate your build, test, and deployment pipeline right within your GitHub repository. It can be directly integrated with GitHub repositories, supports CI/CD and much more, including issue labeling, releasing software, and automated workflows based on GitHub events.

When choosing tools, we are considering the team's familiarity with the technologies, the complexity of setup and maintenance

Testing Approach:

Afterwards, the testing approach should be identified.

The team will focus on writing a high volume of unit tests due to their speed and efficiency. Mock dependencies to test individual functions or classes in isolation. Ensure you cover positive cases, negative cases, and edge cases. Moreover, as part of the integration tests, we will write tests for critical interactions between components, such as database access layers (DAOs) and external services. We will ensure that these components work together as expected under different scenarios. Additionally, in terms of system testing, we will conduct tests on the entire application to verify that it meets the requirements specified. This will include testing the interaction between all components and external systems in a production-like environment. Lastly, we will supplement automated tests with manual testing, especially for UI/UX aspects, accessibility, and other areas difficult to cover with automated tests. We might use UI automation tools like Selenium or tools for automated accessibility testing to reduce the effort.

Metrics and Coverage:

For Sprint 2, our team will set, once again, a target for code coverage, aiming for at least 80% with a focus on critical paths in the application. We will use coverage tools which are integrated with our testing frameworks to measure and report coverage.

Acceptance Tests:

For each user story in Sprint 2, an acceptance criteria should be developed, as well as automated and/or manual tests need to be created to verify that these criteria are met. Our acceptance tests will validate that the system performs as expected from an end-user perspective. For our website, acceptance tests will cover a wide range of scenarios, including user registration, property and financial management functionalities, and the use of common facilities, to simulate real-world use effectively. The emphasis will be placed on automating these tests to streamline the process, employing tools and frameworks that facilitate Behavior-Driven Development (BDD) for clearer, more understandable test scenarios. We might use BDD tools like Cucumber or SpecFlow to write acceptance tests in a language that is understandable.

Implementation Plan:

We will integrate testing into the development workflow by setting up a CI pipeline that runs tests automatically on every commit. This will ensure that the pipeline includes steps for running

unit tests, integration tests, and system tests. Additionally, we will schedule regular review sessions to assess test coverage and effectiveness, adjust testing strategies as needed, and ensure that testing keeps pace with development. Lastly, we will attempt to encourage collaboration between the teammates in charge of both development and testing to ensure that testing reflects the user's needs and project requirements.