Team name: Infinite Loopers

M3: Formal Analysis and Architecture COSC 310

Team members

1. Noah Stasuik (44083343)
2. Trevor Winser (16216681)
3. Julie Flament (53203964)
4. Lakshay Dang (34951236)
5. Komal Singh (61837175)

Testing strategy and updates (5 marks)

1. Introduction
    a. Objective: Our main aim is to ensure our testing functionality works. Since five of us are working on one project, we want to ensure that our separate code components work after any integration.
    b. Scope: Our scope mainly focuses on meeting all main project requirements. Each page focuses on a main feature or has components of a feature integrated into it, and our scope testing goal is to ensure that our requirements are correctly integrated and stay working through any changes we implement throughout the project.
2. Testing Strategy
    a. Unit testing: After implementing any new code unit, we must develop a test file to ensure it works as intended. To test our backend/frontend connection, we have created GET / POST methods for each database table to ensure they work correctly with each page that makes class to the SQLite code. We also created local data that mocks how users interact with our application to ensure everything runs properly in production.
    b. Integration testing: Once code units are merged in our code base, we conduct black and white box testing to ensure the code works after integration. Whenever we merge code into the pull request, we will perform QA testing of all essential web app functions to ensure that everything works as intended. Automation will play a significant role in integration testing since many of our tests will run when we integrate our branch code into the main branch. Our future testing will incorporate various features such as alerts and predictions.
    c. System testing: Our system testing process goes a step further from integration testing. It involves writing tests with many units interacting with each other, which ensures working code. The number of tests written will be based on any typical user interaction that requires communication between

two or more code units. After completing each feature, we write tests to ensure all the work units function correctly.

   d. Risks: While different levels and types of testing each have their own risks, we will use multiple testing strategies to overcome the downsides of some testing methods. Also, we may run into consistency issues with five team members writing tests. To overcome these risks, we have set up testing automation and created team documentation, which allows for a knowledge transfer when testing is done with new libraries such as Selenium.

3. Testing Environment
   a. Software Requirements: Most of our testing will be in VS Code, Python, SQL Lite, and the live code environment. We can write test cases in Python and do QA work in the live environment to find and fix bugs throughout the development process. We have also started using pytest and selenium for our testing requirements. Our “requirements.txt” file includes these libraries to ensure all team members have the proper installations when testing the code.
   b. Managing Test Data: We will create test data in the backend throughout development. For SQL Lite, we will create test login data to test the login and other features with mock data. For Python testing, we will use assert statements and create mock user interactions that will allow testing to be most effective. Our test data will also focus on edge case testing, such as trying to register a user with an invalid email address.
4. Prioritization and Management
   a. Most of our teams' issue tracking occurs on our GitHub Kanban board. Each programming task will have a test file requirement to ensure that the code has been tested properly; the requirements will differ depending on the type of programming task undertaken (ex, front end = selenium, backend = pytest or other SQLite testing). Also, upon a pull request, the team will perform QA testing to ensure all requirements have been met, test the code

on their local machine, and run the test files themselves to ensure everything works as it should.

   b. Prioritization: Testing and pull requests will be FIFO, meaning that as they come in, we will test the team's code and ensure no backlog. As code is being written, we will follow test-driven development, so before any significant programming, we should have the proper tests to ensure that the code written will meet the task requirements. When tests are broken, we will find out what has changed from before and try to fix them at the time of discovery, reaching out to others in the group when their expertise surrounding a specific feature may be beneficial.

5. Test Automation
   a. Tools: We will use Github actions with a workflow involving PyPy so that our tests can be run automatically. GitHub can run all our tests remotely by installing the dependencies and running all the tests we created in the test folder. Our Pytest and Selenium tests will run automatically. Still, due to the complexities of test automation for SQLite, we will leave database testing as something done locally before creating a PR to merge a branch into the main branch.
6. Approvals and Reporting
   a. Approval Process: When creating a PR that merges into the main branch, we will ensure that at least 2 team members approve the merge when code changes affect application functionality, with at least one performing QA tasks to test the code. When the changes don’t directly change the functionality of the code, one approval with QA testing and running the tests in the test folder will suffice.
   b. Test reporting: We will communicate the testing process on the Kanban board and include testing instructions on all of our PRs to ensure the team is on the same page regarding our testing strategy.

7. Future + Current State of Testing
    a. Future: At the project's current state, we have test files associated with every screen, the Python files we have created, and our SQLite databases. The goal for our testing in future weeks is to create test files that focus on integrated and system-wide functionality so that our testing can prove our working product. Since we will finish implementing most of our features in the coming weeks, we will also focus on creating tests with more coverage and edge case testing. Lastly, when we get closer to finishing off testing as a team, we will update our group documentation to ensure alignment across the team regarding the testing strategy. Once we feel that our project is complete as a team, we will also run through the entire project and make sure the coverage of the implemented tests is acceptable, and whenever we feel that it is not, we will add tests to improve the project testing.
    b. Current State of Testing: Throughout the project, we ran into many issues involving testing because of knowledge gaps that existed for everyone with pytest, selenium, and SQLite testing. As we learnt more about this testing, we created test files with a test-driven development approach but may have missed coverage due to us learning about testing in the course after we had begun the coding process. Some errors in our tests outputting correctly led to us not discovering many broken tests in our sprint in W11-12. Our tests involving login and data cleaning are functioning, while the rest have minor errors that must be addressed. Other testing, such as those written for Twilio, have some VSCode-specific errors because of how the API interacts with the software, which we will solve by creating a separate location and testing process for these types of tests. Besides this, our database testing with SQLite and much of the front-end selenium testing must be fixed since many tests broke upon integration.

Automation implementation (3 marks)

For the project as a whole, the testing automation is being done using Github actions. Whenever a pull request is made to merge into the main branch, the PyPy and other test automation processes through GitHub will install our requirements folder and run our tests to ensure they all pass when new code is integrated into the code base. Because of our initial testing errors, the automation returns a failed state whenever we open a PR. Still, we see this as a good sign since it means that our automation is working and will indicate when the tests have been adequately fixed. More details on the automation are included in the above section.

Summary (2 marks)

Overall, our process as a team on the Hurriscan IoT sensor project has been great! Currently, a majority of our features, such as the dashboards, alerting systems, and login and registration functionalities, are completed. The stage we are presently in involves integrating much of what we have been building over the past two months. In the next two weeks, we plan to complete the navbar functionality so that switching between pages is seamless, and we want to complete our predictions model and combine it with the alerting functionality to send users an alert whenever a prediction is made that might affect them. After all of this is completed, we need to fix all of our tests and create new ones to ensure that our project as a whole is adequately completed. Our processes as a team have worked well; the documentation we use has made it easy to ensure alignment in how we code and work with the web app. Our communication has also been evident throughout, and reviewing PRs has caused most errors before they end up on the main branch. We feel that at this time, there is not much in our process that needs to be changed since

whenever issues have been identified, we implement the change and communicate it to the team through a group chat. Lastly, to improve the quality of the source code, we have been quite strict in reviewing the PRs, often requisition changes, and we have looked at and tested the code ourselves to ensure that it meets the requirements for good code. In the coming weeks, as a team, we will also review and update all code to ensure that the quality is good and that the design patterns and other intentions we originally had involving the code were followed properly.