5 Testing

5.1 Unit Testing

Frontend

Static Analysis - Eslint has been set up in the frontend application. Eslint will enforce consistent code styling, warns about unused code, removes undefined variables, and more. Adding Eslint to our project makes the code more readable and sets up a baseline of standards, allowing consistency between developers to adhere to.

Type Checking - By default, Javascript is an untyped language, meaning that it is possible to pass incorrect object types to functions that cannot process them at runtime. We added type checking to our project to ensure that the construct we are passing to a function matches what the function was designed to accept. This prevents unexpected errors and behaviors when running the application.

Jest Unit Tests - The frontend code is verified by unit tests, which cover each component. Jest Unit Tests are designed to ensure that each part of a component functions properly.

Firmware

Unit Tests - We will have unit tests in place for the different functionalities that we have including pH, temperature and WiFi. For the WiFi aspect, we will test possible situations that could happen such the WiFi going out then coming back on and verifying that our code will detect the disconnection and will connect once it comes back up. For temperature and pH, we will test this using various liquids of differing temperature and pH. We will verify that the output that we get from our device is similar to what we expected.

5.2 Interface Testing

Firmware

The hardware on the device itself creates an interface, through an RGB LED, that will be utilized by the user to quickly notice warnings or issues happening on the device. These warnings will be displayed by utilizing different colors set on the LED for each warning. This interface will be conducted through many different tests as there are many warnings the LED will display. These tests will need to be manually actuated for these issues to occur and to be tested.

- Wifi Disconnection
- Connect the device to a wifi source and then remove the power from the wifi system.
- Confirm the LED changed to the correct warning color
- Motor Jammed
- Start the feeding process on the device and hold the motor, disallowing it from being able to spin correctly. This would simulate jamming the motor.
- Confirm the LED changed to the correct warning color
- Doomsday Mode
- Connect the device to a wifi source and then remove the power from the wifi system.
- Remove the power source from the device and then reinsert the power source back into the device to power it back up
- Confirm the LED changed to the correct warning color

- Normal Operation
- Setup device as expected. (i.e. connected to wifi and power)
- Let device run for an extended period of time
- Ensure the device LED continues to display the correct operational color throughout the whole testing process

Frontend

The frontend will display components that are interactable to adjust variables such as scheduling, visual notices for pH levels and temperature. There will also be warnings that pop up or skeleton loaders that should replace components while waiting for data. These types of tests require manual intervention/data manipulation to occur and would allow us to check all cases.

- Main screens
- Should be the main default fallback page and easily accessible
- Go through various navigation steps and testing
- Loading data
- Components should load skeleton components for items whilst waiting for data/compilation
- Should not break the user interface if there is either no data or faulty data
- Notifications
- Should show up based on conditions set by either user or system and show relative warning/notification
- Show up as non-invasive and easy to dismiss

5.3 Integration Testing

Backend-Frontend Integration

Using google firebase we can interface with the frontend directly. Testing for the frontend to the backend will require making integration tests within the React frontend that will make calls to the firebase backend that will be able to retrieve data and return it to the frontend. Tests will be successful if the Backend-Frontend round trip receives the correct data that was sent to the backend.

Backend-Firmware Integration

Initial integration tests will be performed using a tool called Postman. This tool will allow us to independently test the REST API calls to the backend whilst not utilizing the backend to ensure that issues are isolated solely to the firmware on the device. Once this testing is completed we will integrate the firmware and backend by directing all API calls from the device to the backend service. Tests will be written to make a specific call to the backend and wait for a response from the backend service to confirm that the expected result was returned.

5.4 System Testing

Backend-Frontend Integration

For backend to frontend connection we will leverage a couple integration tests outlined in section 5.3 to make sure that full run-throughs of data are being correctly fired off. We can utilize test databases and separate connections through Firebase to give an accurate representation of what the system may actually do while also keeping important user data separate.

Backend-Firmware Integration

The integration tests used for the backend-firmware will be a vital part of the system tests, as the device needs to be integrated correctly into the backend for the whole system to be functional. The integration test is described in section *5.3*.

Frontend Unit Tests

Frontend unit tests will tell us there are components that are affecting functionality. In addition, snapshot tests will also be performed, allowing us to check if the components are rendered correctly. With unit testing and snapshot tests, debugging would be easier and verifies that the system is working as expected. These tests are described in sections 5.1, 5.2, and 5.5.

Firmware Units Tests

Firmware unit tests are important for system testing because these tests follow the requirements of the project for features available within the device itself. These will need to be tested to ensure compliance with the requirements and that when all information is received by the device it is able to handle the actions correctly and without error. These tests are described in section *5.1*.

5.5 Regression Testing

CI/CD Pipeline - Our CI/CD pipeline in Gitlab is configured to run all of our test code when making a pull request. This will ensure that when we add new code to our project, what we have added does not break existing features. New features in our application should build upon old functionality without causing it to fail.

Frontend

Snapshot Tests - Snapshot tests take a screenshot of each screen of the application and compare it to a new screenshot when you push to gitlab. This ensures that when you make changes to the UI, what gets merged to the main branch is what you expect. If a snapshot differs from what it was previously, it will warn the developer and the developer can update the snapshot if the changes were expected otherwise they will know that they have created a bug.

Firmware

Regression testing for the firmware is important as a lot of changes can have adverse effects on other parts of the code itself due to memory constraints and register crossovers. The firmware of our device will benefit from the use of the CI/CD Pipeline to ensure that each new code version uploaded is in compliance with all the previous release tests.

5.6 Acceptance Testing

Frontend

E2E Testing - End to end testing verifies that our application runs as expected from the users perspective. This style of testing is done by building the release configuration of the application and allowing full control of the UI through the testing interface. We will be using the Detox package to do this kind of testing in our app. E2E tests will confirm that all functionality such as button clicks, user input, authentication and more are working properly.

Backend

Data storage and passing must be fully functional and thoroughly tested in order to ensure that the backend as a whole functions properly. Acceptance testing would include adding sample data into our databases and simulating transfers to both the frontend and the firmware. Once the data is able to be properly stored and accurately received by the frontend and firmware, our acceptance tests for the backend will be complete. The client would most likely not be too involved in this portion of testing since it involves a lot of behind-the-scenes work that is not constrained by any requirements.

Firmware

Acceptance testing of the physical device features will be performed with physical tests as the device itself interacts with the world. We will perform tests to ensure that the device complies with the requirements documents. PH testing will be done through use of pH solutions at three different set pH levels. Temperature testing will be done using a store-bought thermometer and varying temperatures of water to confirm that our device is getting the accuracy that was required. The motor will be tested thoroughly within the enclosure of the device to ensure that each food dispense is in an accurate amount to what was set and is consistent with every dispense. LED will be tested as described in section 5.2.

5.7 Results

Our results would be considered successful in implementation if all unit tests, integration tests, interface tests, system tests, regression tests, and acceptance tests are completed successfully. As well, these tests should ensure that they cover the entire scope of the project to guarantee that each module is being tested thoroughly.

The results of our tests are the overall integration and compatibility measures of our components being built into a singular product. With these measurements giving insight into how our client wants their product to function and fulfill their needs we are able to tweak and change our product based on our tests to further satisfy the aspect of the product our client focuses on the most.

From all of our testing throughout the different aspects of this document we have found ways to improve our product but overall that our product continues to bridge the gap between our clients problem and our solution.