# 1.0 Intro

The product requirements for sprint 1 have been modified as a result of timing constraints and newly found technical requirements which the developers were unaware of previously. The main functionality of the Sprint 1 has been modified to include a user registration and sign-in feature, whereby users can save their emails and unique IDs to a remote database. The user can also connect to a Firebase Database, in order to save/retrieve plant information. As planned, the Sprint 1 also included a hardware method of posting data to the database. Moreover, a method of notifying the user with information was also implemented.

The Sprint 2 however, we will focus on several improvements to existing features, and will strive to implement new necessary features. The application will be able to read and record data from the Firebase database, trigger push notifications to specific users, edit functions for the current plants, and send real-time data through sensors to the database.

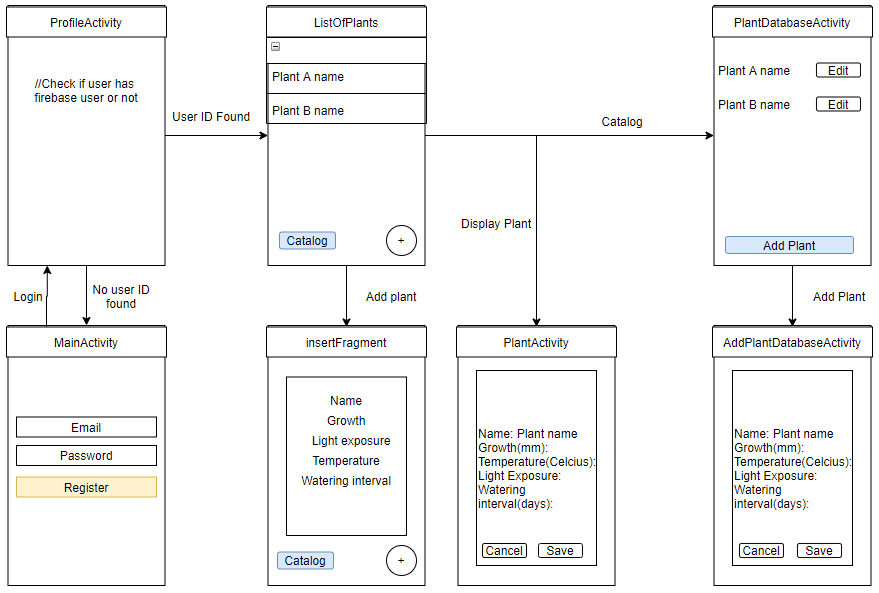
# 2.0 Requirements

See Annex for full table of product backlog and sprint backlogs.

# 3.0 Design Document

The current design document explores the Sprint 1 iteration of features included, giving the user the ability to register an account to the firebase database, create a plant profile, view its details and be able to receive notifications.

## 3.1 Android Application Wireframes



**Figure 1. Android application wireframe of Healthy Leaves user experience**

The application wireframe can be described with figure 1. As a user opens and starts the Healthy Leaves android application it will by default start the MainActivity due to Android Manifest declarations, it will directly go to the ProfileActivity which checks whether or not a userID can be found for this specific android device. If no userID was stored, the activity will be reverted to MainActivity where the user can create a user profile with the signup page. Else, if the userID is found, the activity ListOfPlants that displays a list of plants owned by the user is displayed (local database that will be changed in future sprints). It also gives access to expand the specific plant’s information (and graphs in the future) and the user can hit the plus circle button to add a new tuple to the local database of plants. The button catalog takes the user to PlantDatabaseActivity where a cloud based Firebase database of all plants entered by any user on any device is displayed with their information too. A button add plant can be pressed to be taken to AddPlantDatabaseActivity where the user can write to the firebase database to store new types of plants with their information.

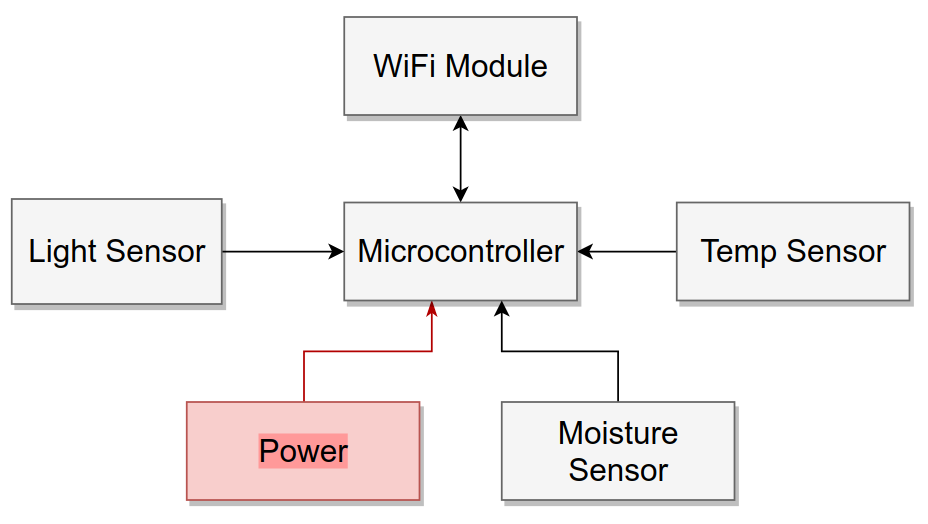
## 3.2 System Architecture

## 

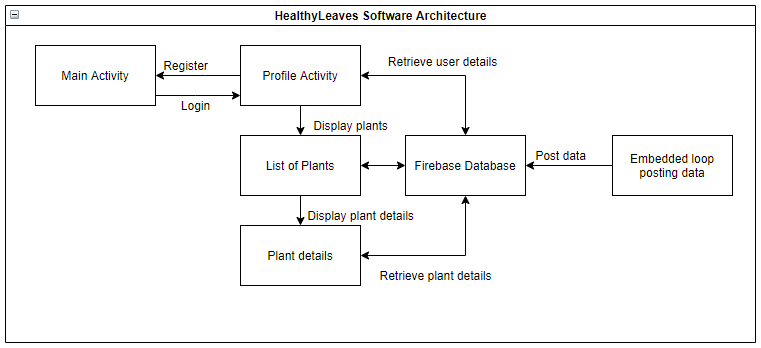
**Figure 2. System Architecture of firebase authentication**

The figure 2 given above describes the system architecture of firebase authentication. The Firebase database is connected to two main types of devices: the embedded systems on the plant and its pot (arduino) and the android client application. On the arduino side, only posts/write will be performed to the Firebase database as it measures the plant’s environment. On the client app, both read and write operations will be performed and will get authenticated by an encrypted Token (using HTTPS) stored on the android phone client side which is used to determine if the database read and write operations requested by the android client are done by a signed-in user on the server. Therefore the token is a security measure used to verify the integrity and authenticity of the request and the server can retrieve the userID from it.

## 3.3 Hardware Architecture

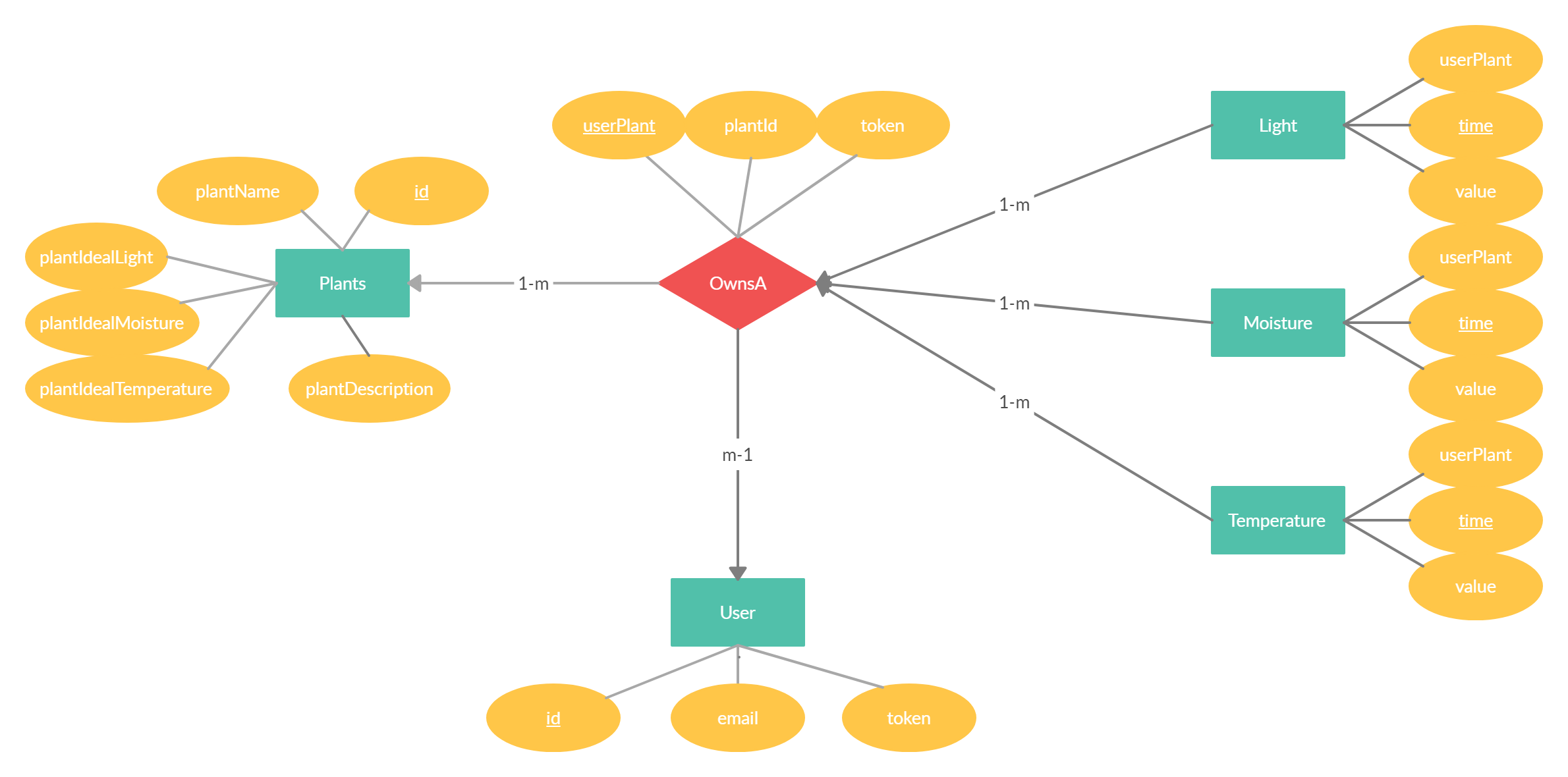


## 3.4 Software Architecture



**Figure 3. Software Architecture of Healthy Leaves project**

The ProfileActivity is the activity that the user will first open when the application starts running. If the user has never logged in before, he/she will be taken to the MainActivity, which will prompt the user to register with an account email and password. These credentials will then be saved into the Firebase database, and the user will be given a token. The MainActivity will then redirect the user back to the ProfileActivity, where it will verify if the user has a token once again. Given that a token is now saved for that user, he/she will be redirected back to ListOfPlants. The ListOfPlants contains the list of plants that the user has already added to his account. If the Addition button is pressed in the bottom right corner of the screen, he/she will be redirected to the PlantActivity screen, where the user will be prompted to fill in the name, growth, light exposure, temperature and watering interval fields to add that respective plant. The information above is currently being hard coded, and will eventually be pulled from a database of plants that contains all the relevant information for a plant name. If the user clicks on a plant name from the ListOfPlants, the user will be redirected to the PlantProfile, where he/she will be able to see the relevant information to the plant that was clicked on. Given the user’s token ID and plant details, a relevant cloud function will send a notification to the respective user, advising on how to manage his/her plant. The user will also have the ability to add data to the Plant catalog, which is a table containing all plants, independent of users. Future sprints will allow users to input their local plants according to the plants available in the database.



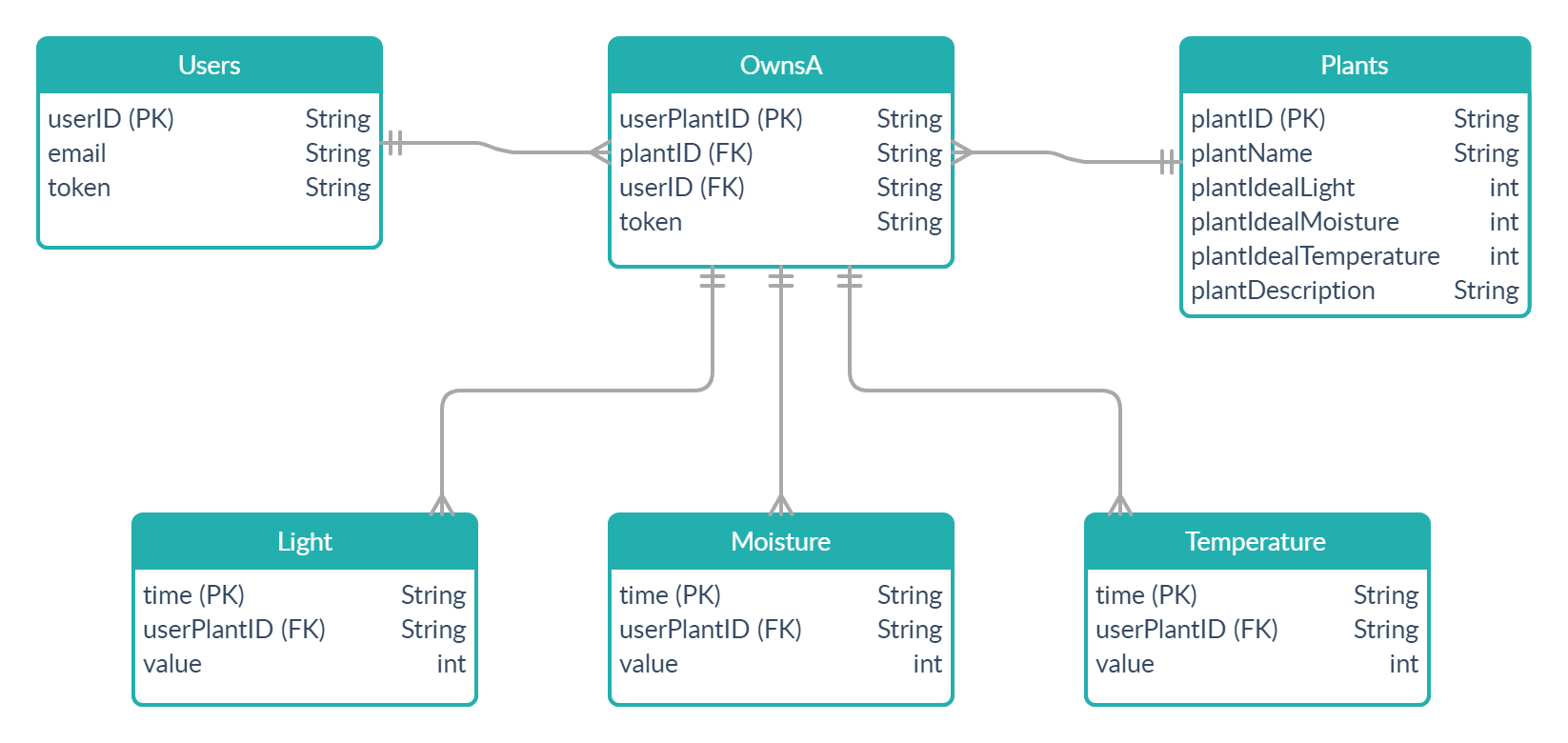
**Figure 4. Entity Relationship diagram (ER-diagram) of Healthy Leaves project**

The figure above illustrates the entity relationship diagram that will structure the firebase database of the HealthyLeaves project. These database entity relationships will be used both on the android and arduino as the whole information being written and read by both hardware entities will partition the collected data in these uniform relationship structures. In fact, the Plants, OwnsA and Users tuples will be created and written from the android side of the project while the Light, Moisture, and Temperature will be created and written from the Arduino side of the project.

The Plants will have a unique primary key given by their id and hold general information on the specific plant such as its name, its ideal light, moisture, and temperature level, and a short description of the plant. The Users will have a unique primary key given by their id and hold authentication attributes such as their email and token. In android studio firebase implementation, the token can be used to get the current user logged in profile the application is currently running on.

OwnsA is a relationship that describes a User tuple owns a plant. A user can own multiple times the same plant type from Plants, therefore plantID and token can’t form a primary key. Therefore, OwnsA has its own unique non null primary key userPlantID. Note the many to one relationship where a user can own multiple plants and a plant can be owned by multiple users. However, the relationship OwnsA can only describe the relationship between a single user and a single plant.

Finally, the light, moisture, and temperature entities describe each data measured by the arduino module and its sensors. Each measurement by the different 3 types of measurements will collect an integer type measurement and will note the time at which it was taken at. Because no two measurements by the same sensor can be done at the same time (using epoch time, how many seconds have elapsed since January 1st 1970), the attribute time can be used as the primary key of the light, moisture and temperature primary keys. They also hold a foreigh key of the OwnsA userPlantID to know to which specific plant owned by a user those measurements are referring to. Every measurement is only associated to one OwnsA relationship while an OwnsA relationship will have multiple measurements. In terms of optimal database design, these light, moisture and temperature tables will be extremely large and may lead to long response time.



**Figure 5. Database diagram of Healthy Leaves project**

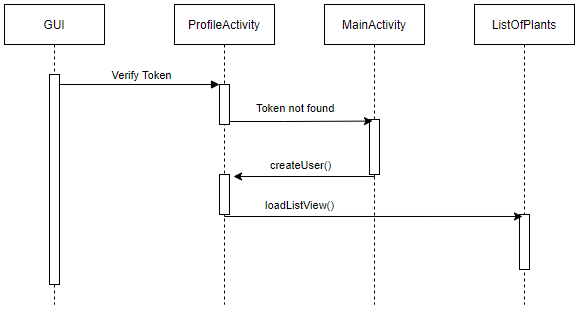
The figure above illustrates the database diagram that will be used to implement the firebase database of the Healthy Leaves project. It follows the logical relationships designed in the ER-diagram of figure 4.

## 3.5 Use Cases and Sequence Diagrams

The following Use Cases, given the current iteration of the product, will be tested:

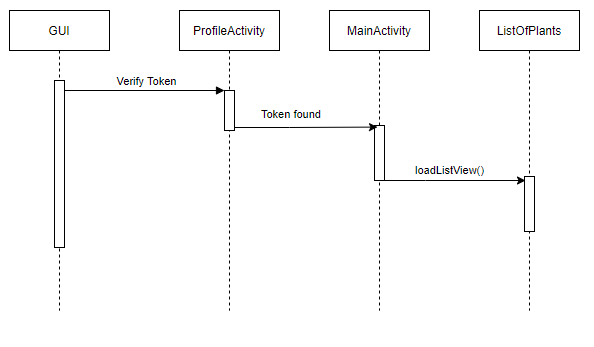
1. The user registers to the app for the first time
2. The user is already registered and logs in
3. The user adds a plant
4. The user opens plant details

Use Case #1



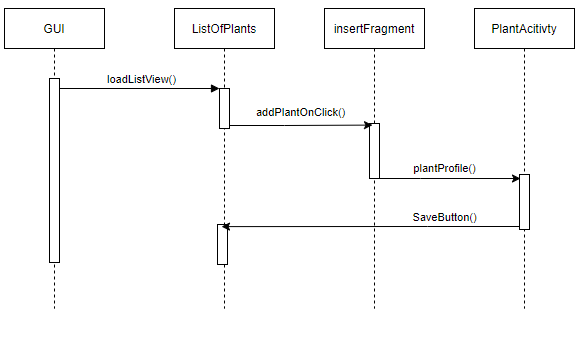
The ProfileActivity will begin by verifying the current token and determining if it is null or not. Given that the user is registering for the first time, it will be. The token is then considered not found, and prompts the user to the MainActivity. The MainActivity will force the user to create an account given an email address and password. The account creation will create a token for the user, save it locally and redirect the user to the ProfileActivity. From there, the token will be found, and redirect the user to the ListOfPlants, where his respective plants will be displayed.

Use Case #2



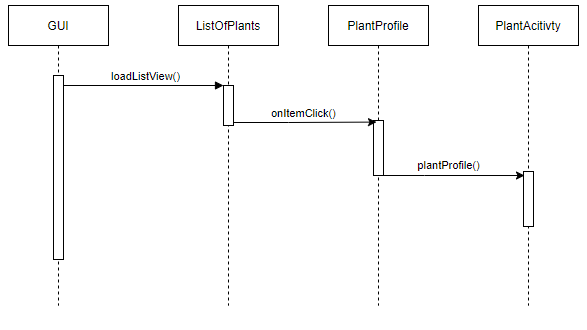
The second use case assumes that the user is already registered and has a token saved locally. Upon opening the app, it will find the respective token, and immediately prompt him/her to the ListOfPlants.

Use Case #3



Given that the user is logged in, upon pressing the button on the bottom right corner of the screen, it will open a fragment, prompting the user to input text fields relating to the plant creation process. As of sprint 1, the plant creation process is done manually by the user. Upon completion, the PlantAcitivty will redirect the user to the ListOfPlants if the button “Save” is clicked.

Use Case #4



The use case where the user is viewing the plant details assumes the user is already logged in and viewing his own plants. Upon clicking a plant, the application will redirect the user to PlantActivity, where the relevant data attributed to the plant will be displayed.

## 

# 4.0 Testing

4.1 Test Plan 1: Firebase Connection

4.1.1 Summary

Requirement ID: S-10

In this section, we are trying to verify the connection with the Firebase Database, in order to ensure that the user can connect to the database and properly receive notifications, regardless of platform or application activity (on or off).

|  |  |  |
| --- | --- | --- |
| Test Case S-10.1 |  |  |
| Pre-Condition: Notifying user with application closed |  |  |
| Steps | Expected Results | Actual Results |
| 1. Close the phone application | The phone application should close | The phone application closes |
| 2. Turn on Wifi | Wifi should turn off | Wifi turns off |
| 3. Wait for notification | Notification should still arrive | Notification does arrive |
| Result: Pass |  |  |

|  |  |  |
| --- | --- | --- |
| Test Case S-10.2 |  |  |
| Pre-Condition: Notifying user application on |  |  |
| Steps | Expected Results | Actual Results |
| 1. Open application | The phone application should open | The phone application opens |
| 2. View List of Plants | The list of plants should appear | The list of plants does appear |
| 3. Wait for notification | Notification should still arrive | Notification does arrive |
| Result: Pass |  |  |

|  |  |  |
| --- | --- | --- |
| Test Case S-10.3 |  |  |
| Pre-Condition: User tries to avoid registering |  |  |
| Steps | Expected Results | Actual Results |
| 1. Open application | The phone application should open | The phone application opens |
| 2. Press back multiple times | The application should redirect to ProfileActivity, and MainActivity again | The application does redirect the user to MainActivity |
| Result: Pass |  |  |

|  |  |  |
| --- | --- | --- |
| Test Case S-10.4 |  |  |
| Pre-Condition: User logs in to different phone with same credentials |  |  |
| Steps | Expected Results | Actual Results |
| 1. Open application | The phone application should open | The phone application opens |
| 2. Open new phone and enters credentials | The application should not register the same account again, it allow the user to log in | The application does allow the user to log in without register the same account again |
| 3. View List of Plants | The application should redirect the user to his list of plants | The application does redirect the user to his list of plants |
| Result: Pass |  |  |

4.2 Test Plan 2: Plant profile and display

4.2.2 Summary

Requirement ID: S-14

In this section, we are creating a listview display in the main activity to view all the plants in the database. Furthermore, we want to ensure that users can add additional plants with their corresponding information in the listview.

|  |  |  |
| --- | --- | --- |
| Test Case S-14.1 |  |  |
| Pre-Condition: Adding new plants |  |  |
| Steps | Expected Results | Actual Results |
| 1. User opens dialog fragment in main activity by pressing floating action button | Dialog fragment opens to allow user to add new plant and corresponding details | Dialog fragment opens to allow user to add new plant and corresponding details |
| 2. Entering plant profile information in the dialog fragment | Text boxes are filled with user inputs | Text boxes are filled with user inputs |
| 3. Pressing save button | Once filled correctly, user input is saved and the dialog fragment closes. New plant added to listview in main activity. | Once filled correctly, user input is saved and the dialog fragment closes. New plant added to listview in main activity. |
| Result: Pass |  |  |

|  |  |  |
| --- | --- | --- |
| Test Case S-14.2 |  |  |
| Pre-Condition: Leaving blanks in dialog fragment |  |  |
| Steps | Expected Results | Actual Results |
| 1. User opens dialog fragment in main activity by pressing floating action button | Dialog fragment opens to allow user to add new plant and corresponding details | Dialog fragment opens to allow user to new plant and corresponding details |
| 2. Filling text boxes with numbers and letters, as well as leaving them blank | Text boxes are filled with user inputs | Text boxes are filled with user inputs |
| 3. Pressing save button | If any textboxes are left blank, the app prompts the user to fill them out. Cannot proceed to save before doing so. Once filled, user input is saved and the dialog fragment closes. New plant added to listview in main activity. | Application crashes |
| Result: Pass |  |  |

|  |  |  |
| --- | --- | --- |
| Test Case S-14.3 |  |  |
| Pre-Condition: Entering incorrect information in fragment |  |  |
| Steps | Expected Results | Actual Results |
| 1. User opens dialog fragment in main activity by pressing floating action button | Dialog fragment opens to allow user to add new plant and corresponding details | Dialog fragment opens to allow user to add new plant and corresponding details |
| Filling plant description textboxes with invalid inputs (ex: watering interval to 0, etc) | Text boxes are filled with user inputs | Text boxes are filled with user inputs |
| 3. Pressing save button | Upon clicking the save button, the application warns the user about invalid inputs in the fragment and prompts them to change them. Cannot proceed to save before doing so. Once filled correctly, user input is saved and the dialog fragment closes. New plant added to listview in main activity. | No warnings given to users. Invalid inputs are saved in the plant profile |
| Result: Pass |  |  |

|  |  |  |
| --- | --- | --- |
| Test Case S-14.4 |  |  |
| Pre-Condition: Entering already used plant name |  |  |
| Steps | Expected Results | Actual Results |
| 1. User opens dialog fragment in main activity by pressing floating action button | Dialog fragment opens to allow user to add new plant and corresponding details | Dialog fragment opens to allow user to add new plant and corresponding details |
| Entering name for plant that is already used by another plant in the database | Text boxes are filled with user inputs | Text boxes are filled with user inputs |
| 3. Pressing save button | Upon clicking the save button, the application warns the user that the plant name is already being used by another plant in the database and prompts the user to change the name. Cannot proceed before doing so. Once filled correctly, user input is saved and the dialog fragment closes. New plant added to listview in main activity. | plant name is saved and there are now multiple plants in the database with identical names |
| Result: Pass |  |  |

4.3 Test Plan 3: Database

4.3.2 Summary

Requirement ID: S-16

In this section, we are creating a connection to the firebase database and saving plant profiles to it. The user should be able to add plants and retrieve from the firebase database.

|  |  |  |
| --- | --- | --- |
| Test Case S-16.1 |  |  |
| Pre-Condition: User inputs all empty fields |  |  |
| Steps | Expected Results | Actual Results |
| 1. Open application | The phone application should open | The phone application opens |
| 2. Click on Catalog | The phone application should redirect the user to the catalog list of plants | The phone application does redirect the user to the catalog list of plants |
| 3. Click on add plant | The phone application should redirect the user to the add plant to database page, where the user can fill in the inputs | The phone application does redirect the user to the add plant to database |
| 4. Leave all inputs blank and press save | The phone application should alert the user and focus on empty inputs to fill out | App alerts the user and focuses on inputs. |
| Result: Pass |  |  |

|  |  |  |
| --- | --- | --- |
| Test Case S-16.2 |  |  |
| Pre-Condition: User inputs some empty fields |  |  |
| Steps | Expected Results | Actual Results |
| 1. Open application | The phone application should open | The phone application opens |
| 2. Click on Catalog | The phone application should redirect the user to the catalog list of plants | The phone application does redirect the user to the catalog list of plants |
| 3. Click on add plant | The phone application should redirect the user to the add plant to database page, where the user can fill in the inputs | The phone application does redirect the user to the add plant to database |
| 4. Leave some inputs blank and press save | The phone application should alert the user and focus on empty inputs to fill out | App alerts the user and focuses on inputs. |
| Result: Pass |  |  |

|  |  |  |
| --- | --- | --- |
| Test Case S-16.3 |  |  |
| Pre-Condition: User inputs fields in every input |  |  |
| Steps | Expected Results | Actual Results |
| 1. Open application | The phone application should open | The phone application opens |
| 2. Click on Catalog | The phone application should redirect the user to the catalog list of plants | The phone application does redirect the user to the catalog list of plants |
| 3. Click on add plant | The phone application should redirect the user to the add plant to database page, where the user can fill in the inputs | The phone application does redirect the user to the add plant to database |
| 4. Fill out all fields | The phone application should save a plant to the database | The phone application does save a plant to the database |
| Result: Pass |  |  |

|  |  |  |
| --- | --- | --- |
| Test Case S-16.4 |  |  |
| Pre-Condition: Plant database loads when going to plant database |  |  |
| Steps | Expected Results | Actual Results |
| 1. Open application | The phone application should open | The phone application opens |
| 2. Click on Catalog | The phone application should redirect the user to the catalog list of plants and display the plants | The phone application displays the plants |
| Result: Pass |  |  |

|  |  |  |
| --- | --- | --- |
| Test Case S-16.5 |  |  |
| Pre-Condition: Plant database loads when going to plant database after adding plant |  |  |
| Steps | Expected Results | Actual Results |
| 1. Open application | The phone application should open | The phone application opens |
| 2. Click on Catalog | The phone application should redirect the user to the catalog list of plants and display the plants | The phone application does not display the plants |
| 3. Add plant | The phone application should add a plant given the user input | The phone application does add a plant |
| 4. View plants | The phone application should retrieve plants and display newly added plant at the bottom of the recyclerview | The phone application does display the recyclerview and the newly added plant |
| Result: Pass |  |  |

4.4 Test Plan 4: ESP32 Microcontroller

4.4.2 Summary

Requirement ID: C-1

In this section, we are creating a connection between the microcontroller and the firebase database. This enables sending data to and from the firebase database.

|  |  |  |
| --- | --- | --- |
| Test Case C-1.1 |  |  |
| Pre-Condition: Microcontroller is powered |  |  |
| Steps | Expected Results | Actual Results |
| 1. Setup ESP32 library in ArduinoIDE | Be able to successfully compile code to the Sparkfun ESP32 | Successfully compiled code to the Sparkfun ESP32 |
| Result: Pass |  |  |

|  |  |  |
| --- | --- | --- |
| Test Case C-1.2 |  |  |
| Pre-Condition: Microcontroller is powered |  |  |
| Steps | Expected Results | Actual Results |
| 1. Connect microcontroller to WiFi network | The microcontroller successfully connects to WiFi and return no error code | The microcontroller does successfully connect to WiFi with no errors |
| 2. Send data to Firebase | The microcontroller must successfully stores data on Firebase | The microcontroller does successfully store data on Firebase |
| Result: Pass |  |  |

|  |  |  |
| --- | --- | --- |
| Test Case C-1.3 |  |  |
| Pre-Condition: Microcontroller is powered |  |  |
| Steps | Expected Results | Actual Results |
| 1. Connect microcontroller to WiFi network | The microcontroller successfully connects to WiFi and return no error code | The microcontroller does successfully connect to WiFi with no errors |
| 2. Receive data from Firebase | The microcontroller must successfully fetch data on Firebase | The microcontroller does successfully fetch data from Firebase |
| Result: Pass |  |  |

## Definition of done

|  |  |  |  |
| --- | --- | --- | --- |
| **Story ID** | **DoD** | **Confirmation** | **Validation** |
| S-10 | Tested | Can the microcontroller connect to the Firebase Database?  Can the user register to the Firebase Database?  Can the application retrieve the user token? | Complete |
| S-16 | Tested | Do we have an accessible database  of various plants with their quantifiable  information?  Do we have an accessible database of the user owned plants and their quantifiable accessible data log?  Can we access the database?  Can we post to the database? | Incomplete, the database doesn't establish relationships between plants and users and therefore also don't hold quantifiable data log. |
| S-14 | Tested | Can the plant profile be saved locally?  Can the plant profile be saved remotely? | Complete |
| A-1 | Deployable | Does each team member knows  what to do in Sprint 2?  Is the sprint 2 document complete? | Complete |
| A-2 | Deployable | Can my microcontroller connect to the WIFI given this module? | Complete |
| A-4 | Deployable | Can we verify all the requirements?  Can we break down the app  somehow? | Complete |
| A-5 | Deployable | Does the design document contain  all the information about the project?  Is it submitted? | Complete |
| C-1 | Have had a code review | Can the microcontroller read moisture levels?  Is the data saved to the microcontroller? | Complete |

## 