National College of Ireland

BSHC4

Software Development

2021/2022

Karl Tallon

X18752469

X18752469@student.ncirl.ie

Pocket Botanist – An Android App

Technical Report

Contents

Annotations

App	Application (i.e. on Android)
Project	Entire project relating to both the app & device
Device	Pocket Botanist's proprietary Soil Sensor Device
->	Indicates moving between app pages
Firebase	Google Firebase Storage & Google Firebase Realtime Database

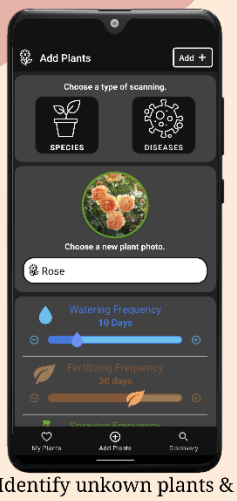
Showcase Poster

National College of Ireland

Pocket Botanist

Karl Tallon - BSHCE (Hons) in Computer Science

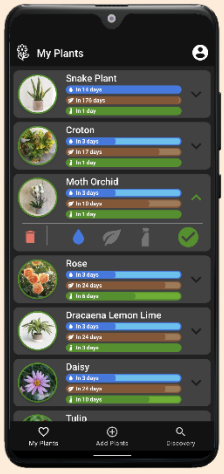
Project Description:
Having grown up with an interest in botany, I wanted to create an app which would help users, even if untrained in horticulture, to properly care for their plant collections. The result of this project is the android app named Pocket Botanist, & our proprietary device, which enables users to more easily take care of their beloved plants at home.



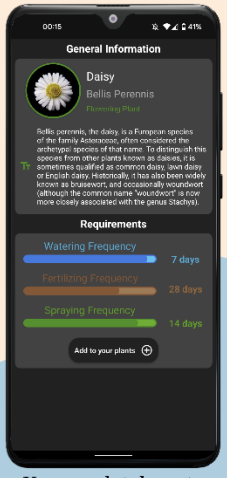
Identify unkown plants & ailments, add new plants to your collection.

Don't know what plant you're looking at? Easily scan it to identify it!

Remove the hassle from plant care with the handy Pocket Botanist device, automatically detect when your plants need watering!



Manage your favourite plants & track all of their essential needs!



Use our database to research plants & their individual needs.

Executive Summary

This project's core goal was to build a polished android mobile app using Android Studio & Firebase. The overall purpose of this app is to provide users with a Plant Care app which can facilitate a multitude of features to help aid in the caretaking of the user's plants, even for those who are untrained in the field of horticulture. Users can utilise this app to easily take full care for their plants. The app, named "Pocket Botanist", includes the following main functionalities: Account Registration, Account Login, Catalogue your plant into your own collection, Calendar feature to track your plant's needs (Watering, Fertilizing, Spraying), Search the Pocket Botanist Database for new plants, Using image classification to identify unknown plant species, Using image classification to identify plant diseases & infections. These image classification methods compared the uploaded image of the plant with the TensorFlow Lite models I created. There is also proprietary hardware which I designed specifically to accompany the Pocket Botanist app, this hardware enables users to easily monitor plant soil water levels. A smart, Bluetooth-enabled soil monitoring device is being planned for development in the future, with in-app integration also being planned. The main purpose of this project is to give user's full control over their plant care routine & empower them with the necessary tools to become confident in their caretaking.

In this technical report I will be discussing four major topics including the Introduction, the System & it's requirements, the Conclusion & then finally any Future Development or Research. The purpose of this report will be to demonstrate the Pocket Botanist from multiple aspects as to help audiences gain a better understanding of the project from several considerations.

The result of this report focuses more from the standpoint of conclusions, reflections, recommendations & the future plans more than purely a visual demonstration of the app itself & its accompanying hardware. As a result of this, this report will discuss the project not only from a demonstrative perspective, but also an explanatory standpoint too while making future plans to improve upon features. The initial version of Pocket Botanist is planned as a free of charge app with future plans for payment plan incorporations as the features & reputation of Pocket Botanist is bolstered, as well as future plans to commercialise the accompanying hardware.

The conclusion to this project is the first release-ready version of Pocket Botanist & the proof-of-concept hardware to accompany it, both enabling users to more easily take care of their plant collections.

This project is viewable at https://github.com/Tallon1/PocketBotanist.

1.0 Introduction

1.1. Background

I conceptualised this project with two main goals in mind: To produce an interesting app with legitimate real world & commercial possibilities, but also to challenge myself with the prospect of learning cutting-edge technology which is relevant in today's industries. It is because of these overarching goals that I have chosen this project & the difficult features contained within it. I believe through this project's design & proprietary hardware, there is commercial viability to this endeavour which I plan to follow through with.

1.2. Aims

This project's primary aim is to produce a polished android app with the purpose of being used as a Plant Care app, while also producing proprietary hardware to complement its functionality. This current version of Pocket Botanist requires an internet connection to be used but in future there are plans for offline usage, this is because all information is stored through Firebase so Pocket Botanist currently requires a stable connection. The four main features which are at the core of this app are:

- Registration & Login – The Pocket Botanist uses Firebase to allow for users to register an account with our app, log into the app & recover lost passwords.
- Cataloguing – This feature will allow users to add their plants to their collection, customise the caretaking requirements as they see fit for their environment & also view their plant's individual information for caretaking e.g. Watering, Fertilizing.
- Requirement Tracking – This feature allows for in-app tracking of the individual plant needs such as Watering, Fertilizing & Spraying needs. Alongside this, the handy push notifications of Pocket Botanist will help ensure users never miss that important reminder.
- Discovery – Allow users to search the Pocket Botanist Database for specific plants & learn about their information. This information includes their Latin name, plant type, a description of the plant, an image of the plant & their caretaking requirements.
- Plant Type Classification – This feature utilises TensorFlow image classification & machine learning to identify the species of plant. The user can upload a photo, through their camera or gallery, then identify the plant species by comparing the image against my trained TensorFlow model & providing a confidence rating.
- Plant Disease & Infection Classification – Much like the previous feature, this feature makes use of TensorFlow image classification & machine learning to identify the diseases or infections causing harm to user's plants. The user can upload a photo, through their camera or gallery, then identify the ailment by comparing the image against my trained TensorFlow model & providing a confidence rating for the result.

Additionally, the accompanying hardware I've designed can be placed in a plant's soil & the two metal probes will detect the water content of the soil. The soil's moisture will be indicated by the LED light on top of the device, if the LED is on then the soil is too dry & if the LED is off then the soil has enough water.

1.3. Technology

After many hours of research & deliberation over which would suit my project's requirements best, these are the four main technologies being utilised in this project, they are as follows:

- Android Studio (Java) – Android Studio is the integrated development environment which I will be using for my app. Having been built upon JetBrains' IntelliJ, Android Studio is the official IDE used for developing Google's Android OS & it is specifically designed for Android development making it a fantastic platform for my project *(Android Developers, 2019).*
- Google Firebase – Firebase is another Google flagship platform utilised in the development of mobile & web apps. Firebase is typically sought after for its excellent tools for analytics tracking, reporting & fixing app crashes *(Firebase, 2019),* but for my project I will be focusing on making use of their free-to-use Realtime Database, Storage & Authentication services. These services will provide the necessary tools for me to have fluid Register & Login features but also provide storage for both user & plant data.
- TensorFlow – TensorFlow, in short, is a free & open-source library for artificial intelligence & machine learning. Typically, TensorFlow is used with a focus on deep neural networks with an emphasis on classification, perception, understanding, prediction, discovering & creation *(TensorFlow, 2020).* Regarding my project, I will be utilising TensorFlow's ability for classification & confidence rating for the identifying of plant types or diseases. The TensorFlow models were both created using the Google Cloud Platform to train my own models specifically for Species & Disease classification.
- GitHub – GitHub is an Internet Hosting provider for version control & software development *(GitHub, 2019).* I have utilised this fantastic service to host my application & make its APK available for any user to download. This is viewable at my GitHub.

1.4. Structure

In this report, the high-level structure can be divided into four main sections: the Introduction, the System, the Conclusion & then Future Development or Research. In each of these sections, I will be presenting my project from multiple aspects as to give the audience a better understanding of the logic behind my project. This technical report will make use of both visual & textual demonstrations as to provide a more rounded insight into the context of this project. Each of the aforementioned sections will be further dissected into subtopics which will provide key information regarding those respective topics.

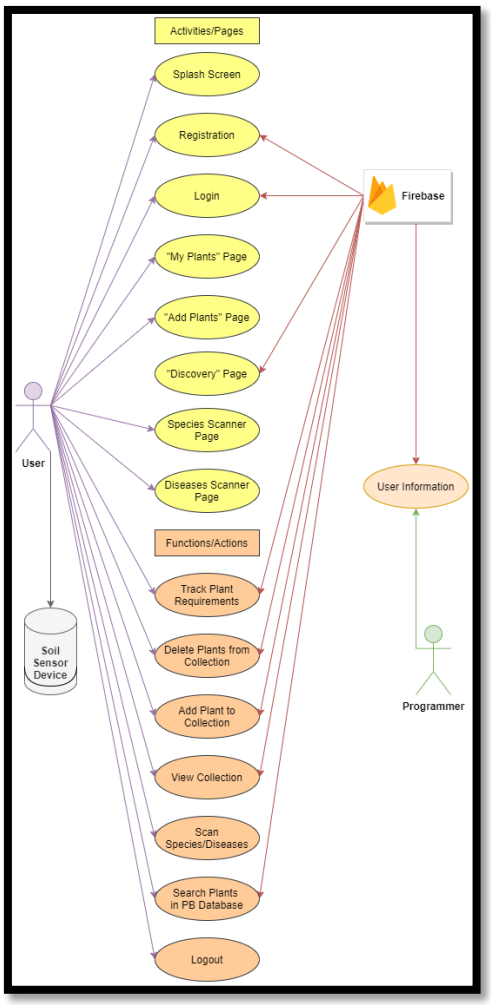
2.0 System

2.1. Requirements

2.1.1. Functional Requirements

- This app is used for cataloguing, identifying & caring for plants.
- Users must use an Android mobile phone.
- This app requires users to Register, Login, recover passwords & Logout.
- This app must record Registration & Login data in the Firebase Database.
- This app must allow users to add plants to their collection.
- This app must allow users to view their catalogued plants & their respective information.
- This app must allow users to track their plant's needs through in-app tracking & push notifications.
- This app must allow users to identify both unknown plant species & diseases using their camera or gallery.
- This app must allow users to search for plants through the Pocket Botanist database.

2.1.1.1. Use Case Diagram



Draw.io Diagram 1

2.1.1.2. Requirement 1 <Splash Screen>

The first requirement is the Splash Screen.

2.1.1.2.1. Description & Priority

The Splash Screen requirement displays the app's logo when the app is first launched. It shows for 1000 milliseconds prior to showing the Main Activity or Login Activity.

2.1.1.2.2. Use Case 1: Splash Screen

Scope

The scope of this particular use case is to let the users recognise the app's logo & attempt to make it more memorable.

Description

This use case describes the initial screen of the app displaying its logo.

Flow Description

Precondition

- The user starts the app.
- The device is connected to the internet.

Activation

This use case begins when an actor (user) has started the app.

Main flow

1. The system identifies the app's splash screen.

Alternative flow

A1: <No internet on device>

1. The device has no internet connection & the system cannot establish a connection to Firebase.
2. The system shows "Connection Error!\n Check your network settings."
3. (External: The user turns on the Wi-Fi &/or mobile internet)
4. The actor (user) clicks "Refresh" button to establish the connection.
5. <Returns to number 1 in the Main Flow>

Exceptional flow

E1: <System errors cause the splash screen to fail>

1. The system shows the failure of the app's splash screen.
2. The actor (user) must restart the app.
3. The use case continues at *position 1* of the main flow. (See E2)

E2: <System errors cause the app's splash screen to fail after restarting>

1. The system shows the failure of the app's splash screen.

2. The actor (user) must report this failure to the end user & await their response.
3. The use case continues at *position 1* of the main flow.

Termination

The system presents the apps Login Page (Start next Use Case: User Login).

Post condition

The system goes into a wait state.

2.1.1.3. Requirement 2 <User Registration>

The second requirement is User Registration.

2.1.1.3.1. Description & Priority

The Registration requirement enables users to create an account by inputting their email & password. This will allow users to log into their account & access other functions which can be utilised after log in. The User Registration requirement is the most vital of the system, each user is required to register in order to use the app.

2.1.1.3.2. Use Case 2: Registration

Scope

The scope of this use case is to allow users to create an account in the app, subsequently allowing Firebase to store the relevant information for the individual user.

Description

This use case details the Registration process of the system. Users are required to register before using the app. The Firebase Realtime Database will store all information relevant to the user once they are registered. This will enable them to start the following use case: Logging into their account & activating other functions.

Flow Description

Precondition

- The user uses a mobile phone & has downloaded the app.
- The user starts the app.
- The device is connected to the internet.
- The app is defined as the system.

Main flow

1. The system recognises the Registration of the app.
2. The actor (user) starts the app.
3. The actor (user) clicks the “Register” button.
4. The system directs the user to the Registration Page.
5. The actor (user) inputs their username, email & password into the editable text box.
6. The actor (user) clicks the “Register” button, the actor (user) sees the loading wheel as information is stored on the Firebase database. (See A1) (See A2)
7. The system shows “User successfully registered” once the information has been collected & store into the Firebase database. (See E1)

8. The actor (user) has successfully created an account in the system & is returned to the Login page.

Alternative flow

A1: <One of the editable text boxes are empty: email & password>
1. The actor (user) leaves one of the two editable text boxes empty: username, email & password.
2. The system shows "This field can't be empty", depending on which text box is empty.
3. The actor (user) is required to fill out all details in order to register.
4. This use case continues at *position 4* of the main flow.

A2: <The user has already been registered>
1. The system shows "This email address is already in use by another account.".
2. The actor (user) must use the Back Button & return to the Login page.
3. This use case continues at *position 1* in the main flow of the following use case: Login.

A3: <No internet on device>
1. The device has no internet connection & the system cannot establish a connection to Firebase.
2. The system shows "Connection Error!\n Check your network settings."
3. (External: The user turns on the Wi-Fi &/or mobile internet)
4. The actor (user) clicks "Refresh" button to establish the connection.
5. <Returns to number 1 in the Main Flow>

Exceptional flow

E1: <System errors cause the Registration process to fail>
1. The system shows the Registration has failed.
2. The actor (user) must restart the app.
3. The use case continues at *position 3* of the main flow. (See E2)

E2: <System errors cause the Registration process to fail after restarting>
1. The system shows the failure of the app's Registration process.
2. The actor (user) must report this failure to the end user & await their response.
3. The use case continues at *position 1* of the main flow.

Termination

The system presents the apps Login Page (Start next Use Case: Login).

Post condition

The system goes into a wait state.

2.1.1.4. Requirement 3 <User Login>

The third requirement is User Login.

2.1.1.4.1. Description & Priority

The Login requirement enables users to login to the account they created in Requirement 2. Users logging into their account grants access to the app's features. This requirement is the second most vital in this app. Each user must login before they can use the app's full capabilities. These users will be stored in the Firebase Realtime Database.

2.1.1.4.2. Use Case 3: Login

Scope

The scope of this use case is to allow users to login to their accounts in the app, subsequently Firebase will cross-check their information & direct users to the "My Plants" Page.

Description

This use case details the Login process of the system. Users are required to login before using the app. Firebase will match all information relevant to the user once they are logged in. This will enable them to start the following use case: "My Plants" & activating other functions.

Flow Description

Precondition

- The user uses a mobile phone & has downloaded the app.
- The user starts the app.
- The user has an account in the system.
- The device is connected to the internet.
- The app is defined as the system.

Activation

This use case begins when an actor (user) has started the app & has already registered.

Main flow

1. The system recognises the Login of the app.
2. The actor (user) starts the app.
3. The actor (user) inputs their email & password into the editable text box.
4. The actor (user) clicks the "Login" button & stands by while the system matches his/her data against the Firebase database.
5. The system directs the user to the "My Plants" Page.

Alternative flow

A1: <One of the editable text boxes are empty: email & password>

1. The actor (user) leaves one of the two editable text boxes empty: email & password.
2. The system shows “This field can’t be empty”, depending on which text box is empty.
3. The actor (user) is required to fill out all details in order to login.
4. This use case continues at *position 3* of the main flow.

A2: <Password or Email is incorrect>

1. The actor (user) enters their login details incorrectly.
2. The system shows “The password is invalid or the user does not have a password”, depending on which text box is empty.
3. This use case continues at *position 3* of the main flow.

A3: <No internet on device>

1. The device has no internet connection & the system cannot establish a connection to Firebase.
2. The system shows “Connection Error!\n Check your network settings.”
3. (External: The user turns on the Wi-Fi &/or mobile internet)
4. The actor (user) clicks “Refresh” button to establish the connection.
5. <Returns to number 1 in the Main Flow>

Exceptional flow

E1: <System errors cause the Login process to fail>

1. The system shows the Login has failed.
2. The actor (user) must restart the app.
3. The use case continues at *position 3* of the main flow. (See E2)

E2: <System errors cause the Login process to fail after restarting>

1. The system shows the failure of the app’s Login process.
2. The actor (user) must report this failure to the developer & await their response.
3. The use case continues at *position 1* of the main flow.

Termination

The system presents the apps “My Plants” Page (Start next Use Case: “My Plants” Page).

Post condition

The system goes into a wait state.

2.1.1.5. Requirement 4 <Forgot Password>

The fourth requirement is Forgot Password.

2.1.1.5.1. Description & Priority

The Forgot Password requirement enables users to recover their forgotten passwords. Users will submit their email, then receive an email link to change their password & the change will be registered in the Firebase database.

2.1.1.5.2. Use Case 4: Forgot Password

Scope

The scope of this use case is to allow users to recover their lost passwords.

Description

This use case details the Forgot Password process of the system. Firebase will send an email to the user containing a link, here they can resubmit a new password & the updated password will be registered to their account.

Flow Description

Precondition

- The user uses a mobile phone & has downloaded the app.
- The user starts the app.
- The user has an account in the system.
- The device is connected to the internet.
- The app is defined as the system.

Activation

This use case begins when an actor (user) has started the app & has already registered.

Main flow

1. The actor (user) starts the app.
2. The actor (user) clicks the “Forgot Password?” button & the “Reset Password” dialog box appears.
3. The actor (user) enters their email into the “Reset Password” prompt.
4. The system sends an email containing a Firebase link to the user.
5. The actor (user) follows the Firebase link & enters their new password.
6. The Firebase system registers the updated password.
7. The actor (user) returns to the Login Page & enters their updated password.
8. The system directs the user to the “My Plants” Page.

Alternative flow

A1: <The editable text box is empty: email>

1. The actor (user) leaves the editable text boxes empty: email.
2. The system shows “This field can’t be empty”, depending on which text box is empty.
3. The actor (user) is required to fill out all details in order to register.
4. This use case continues at *position 4* of the main flow.

A2: <No internet on device>

1. The device has no internet connection & the system cannot establish a connection to Firebase.
2. The system shows “Connection Error!\n Check your network settings.”
3. (External: The user turns on the Wi-Fi &/or mobile internet)
4. The actor (user) clicks “Refresh” button to establish the connection.
5. <Returns to number 1 in the Main Flow>

Exceptional flow

E1: <System errors cause the Forgot Password process to fail>

1. The system shows the Forgot Password has failed.
2. The actor (user) must restart the app.
3. The use case continues at *position 2* of the main flow. (See E2)

E2: <System errors cause the Forgot Password process to fail after restarting>

1. The system shows the failure of the app’s Forgot Password process.
2. The actor (user) must report this failure to the developer & await their response.
3. The use case continues at *position 1* of the main flow.

Termination

The system presents the apps “Login” Page (Start next Use Case: “Login” Page).

Post condition

The system goes into a wait state.

2.1.1.6. Requirement 5 <Logout>

The fifth requirement is Logout.

2.1.1.6.1. Description & Priority

The Logout requirement enables users to logout of their account & return to the Login Page.

2.1.1.6.2. Use Case 5: Logout

Scope

The scope of this use case is to allow users to logout of their account.

Description

This use case details the Logout process of the system. This simply signs the user out of their Pocket Botanist account & returns them to the Login Page.

Flow Description

Precondition

- The user uses a mobile phone & has downloaded the app.
- The user starts the app.
- The user has an account in the system.
- The device is connected to the internet.
- The app is defined as the system.
- The user is logged into their account.
- The user is on the “My Plants” page.

Activation

This use case begins when an actor (user) has started the app, has already registered an account & logged in.

Main flow

1. The actor (user) clicks the Profile icon in the top right.
2. The system opens the sidebar displaying the username, email & “Logout” button.
3. The actor (user) clicks the “Logout” button.
4. The system logs the user out of their account & returns them to the “Login” page.

Exceptional flow

E1: <System errors cause the Logout process to fail>

1. The system shows the Logout has failed.
2. The actor (user) must restart the app.
3. The use case continues at *position 1* of the main flow. (See E2)

E2: <System errors cause the Logout process to fail after restarting>

1. The system shows the failure of the app's Logout process.
2. The actor (user) must report this failure to the developer & await their response.
3. The use case continues at *position 1* of the main flow.

Termination

The system presents the apps "Login" Page (Start next Use Case: "Login" Page).

Post condition

The system goes into a wait state.

2.1.1.7. Requirement 6 <My Plants>

The sixth requirement is “My Plants” Page.

2.1.1.7.1. Description & Priority

The use case describes how the user will interact with the “My Plants” Page as they view their plant collection & track their needs within the Pocket Botanist app.

2.1.1.7.2. Use Case 6: My Plants

Scope

The scope of this use case is to show how the user interacts with the “My Plants” Page.

Description

This use case details the “My Plants” Page of the app & how the user will interact with this page as they view their plant collection & track their needs. These plant collections & their individual needs will be stored in the Firebase Realtime Database.

Flow Description

Precondition

- The user uses a mobile phone & has downloaded the app.
- The user starts the app.
- The user has an account in the system.
- The device is connected to the internet.
- The user has logged into the app.
- The app is defined as the system.

Activation

This use case begins when an actor (user) has started the app, has already registered an account & logged in.

Main flow

1. The system displays the Main Activity, the “My Plants” Page.
2. The actor (user) is greeted with the “My Plants” Page.
3. The system displays the user’s plant collection if they have any.
4. The actor (user) selects a plant.
5. The actor (user) checks one of the plant’s caretaking requirements (Watering, Fertilizing or Spraying).

6. The actor (user) clicks the checkmark & updates the plant to state they have taken care of that specific requirement.
7. The system recognises this update & the change is stored in the Firebase Realtime Database.

Alternative flow

A1: <The "My Plants" Page is empty>

1. The actor (user) accesses "My Plants" Page & has no plants in their collection.
2. The system shows "No plants add yet.\n Begin your collection in 'Add Plants'.".
3. The actor (user) may proceed to the next Use Case 7: "Add Plants" Page to begin their own plant collection.
4. This use case continues in Use Case 7.

A2: <No internet on device>

1. The device has no internet connection & the system cannot establish a connection to Firebase.
2. The system shows "Connection Error!\n Check your network settings."
3. (External: The user turns on the Wi-Fi &/or mobile internet)
4. The actor (user) clicks "Refresh" button to establish the connection.
5. <Returns to number 1 in the Main Flow>

Exceptional flow

E1: <System errors cause the "My Plants" process to fail>

1. The system shows the "My Plants" has failed.
2. The actor (user) must restart the app.
3. The use case continues at *position 1* of the main flow. (See E2)

E2: <System errors cause the "My Plants" process to fail after restarting>

1. The system shows the failure of the app's "My Plants" process.
2. The actor (user) must report this failure to the developer & await their response.
3. The use case continues at *position 1* of the main flow.

Termination

The system presents the apps "Add Plants" Page (Start next Use Case: "Add Plants" Page).

Post condition

The system goes into a wait state.

2.1.1.8. Requirement 7 <Add Plants>

The seventh requirement is "Add Plants" Page.

2.1.1.8.1. Description & Priority

The use case describes how the user will interact with the "Add Plants" Page as they add plants to their collections, input its requirements or access the Species & Diseases scanners.

2.1.1.8.2. Use Case 7: Add Plants

Scope

The scope of this use case is to show how the user interacts with the "Add Plants" Page.

Description

This use case details the "Add Plants" Page of the app & how the user will interact with this page as add plants to their collection & input their individual needs. These additions to their plant collections will be stored in the Firebase Realtime Database. This page is also the access point for the Species & Diseases scanners.

Flow Description

Precondition

- The user uses a mobile phone & has downloaded the app.
- The user starts the app.
- The user has an account in the system.
- The device is connected to the internet.
- The user has logged into the app.
- The app is defined as the system.
- The user has clicked into the "Add Plants" Page.

Activation

This use case begins when an actor (user) has started the app, has already registered an account, logged in & clicked the "Add Plants" tab.

Main flow

1. The system displays the "Add Plants" Page.
2. The actor (user) is greeted with the "Add Plants" Page.
3. The actor (user) selects the upload image option.
4. The system displays the choice dialog box for "Take Photo", "Choose from Gallery" or "Cancel" options.

5. The actor (user) chooses their preference & uploads an image for their plant.
6. The system loads the selected images & readies it for upload to Firebase.
7. The actor (user) enters the name of their plant.
8. The actor (user) inputs the individual caretaking requirements of their plant.
9. The actor (user) clicks the “Add +” button.
10. The system uploads their new plant to Firebase which assigns this data to UserPlants within the Users data in the Firebase Realtime Database.

Alternative flow

A1: <No internet on device>

1. The device has no internet connection & the system cannot establish a connection to Firebase.
2. The system shows “Connection Error!\n Check your network settings.”
3. (External: The user turns on the Wi-Fi &/or mobile internet)
4. The actor (user) clicks “Refresh” button to establish the connection.
5. <Returns to number 1 in the Main Flow>

A2: <The actor (user) selects Species Scan button>

1. The actor (user) selects the Species Scan button.
2. The use case continues in Use Case 8: Species Scanner.

A3: <The actor (user) selects Diseases Scan button>

1. The actor (user) selects the Species Scan button.
2. The use case continues in Use Case 9: Diseases Scanner.

Exceptional flow

E1: <System errors cause the “Add Plants” process to fail>

1. The system shows the “Add Plants” has failed.
2. The actor (user) must restart the app.
3. The use case continues at *position 1* of the main flow. (See E2)

E2: <System errors cause the “Add Plants” process to fail after restarting>

1. The system shows the failure of the app’s “Add Plants” process.
2. The actor (user) must report this failure to the developer & await their response.
3. The use case continues at *position 1* of the main flow.

Termination

The system presents the apps “Species Scanner” Page (Start next Use Case: “Species Scanner” Page).

Post condition

The system goes into a wait state.

2.1.1.9. Requirement 8 <Species Scanner>

The eighth requirement is “Species Scanner” Page.

2.1.1.9.1. Description & Priority

The use case describes how the user will interact with the “Species Scanner” Page as they scan an image, from their camera or gallery, to identify the plant’s species.

2.1.1.9.2. Use Case 8: Species Scanner

Scope

The scope of this use case is to show how the user interacts with the “Species Scanner” Page.

Description

This use case details the “Species Scanner” Page of the app & how the user will interact with this page as they scan an image of a plant to identify its species. The app will return the suspected result of the name of that plant’s species & its confidence rating as a percentage (e.g. Rose\n Confidence = 0.9647059).

Flow Description

Precondition

- The user uses a mobile phone & has downloaded the app.
- The user starts the app.
- The user has an account in the system.
- The device is connected to the internet.
- The user has logged into the app.
- The app is defined as the system.
- The user has clicked into the “Add Plants” then the “Species Scanner” pages.

Activation

This use case begins when an actor (user) has started the app, has already registered an account, logged in, accessed the “Add Plants” page & clicked the “Species Scanner” button.

Main flow

1. The system displays the “Species Scanner” Page.
2. The actor (user) is greeted with the “Species Scanner” Page.
3. The actor (user) selects the upload image option.
4. The system opens the user’s gallery.
5. The actor (user) chooses the image they want to scan.

6. The system loads this image into the ImageView.
7. The actor (user) selects the “Classify” button.
8. The system returns the classification result for this image.

Alternative flow

A1: <No internet on device>

1. The device has no internet connection & the system cannot establish a connection to Firebase.
2. The system shows “Connection Error!\n Check your network settings.”
3. (External: The user turns on the Wi-Fi &/or mobile internet)
4. The actor (user) clicks “Refresh” button to establish the connection.
5. <Returns to number 1 in the Main Flow>

A2: <The actor (user) returns to the “Add Plants” page>

1. The actor (user) presses the Back button.
2. The use case continues in Use Case 7: Add Plants.

Exceptional flow

E1: <System errors cause the “Species Scanner” process to fail>

1. The system shows the “Species Scanner” has failed.
2. The actor (user) must restart the app.
3. The use case continues at *position 1* of the main flow. (See E2)

E2: <System errors cause the “Species Scanner” process to fail after restarting>

1. The system shows the failure of the app’s “Species Scanner” process.
2. The actor (user) must report this failure to the developer & await their response.
3. The use case continues at *position 1* of the main flow.

Termination

The system presents the apps “Add Plants” Page (Start next Use Case: “Add Plants” Page).

Post condition

The system goes into a wait state.

2.1.1.10. Requirement 9 <Diseases Scanner>

The ninth requirement is “Diseases Scanner” Page.

2.1.1.10.1. Description & Priority

The use case describes how the user will interact with the “Diseases Scanner” Page as they scan an image, from their camera or gallery, to identify the plant’s species.

2.1.1.10.2. Use Case 9: Diseases Scanner

Scope

The scope of this use case is to show how the user interacts with the “Diseases Scanner” Page.

Description

This use case details the “Diseases Scanner” Page of the app & how the user will interact with this page as they scan an image of a plants to identify the diseases affecting it. The app will return the suspected result of the name of the disease which the plant may have & its confidence rating as a percentage (e.g. Tomato – Early Blight\n Confidence = 0.90588236).

Flow Description

Precondition

- The user uses a mobile phone & has downloaded the app.
- The user starts the app.
- The user has an account in the system.
- The device is connected to the internet.
- The user has logged into the app.
- The app is defined as the system.
- The user has clicked into the “Add Plants” then the “Diseases Scanner” pages.

Activation

This use case begins when an actor (user) has started the app, has already registered an account, logged in, accessed the “Add Plants” page & clicked the “Diseases Scanner” button.

Main flow

1. The system displays the “Diseases Scanner” Page.
2. The actor (user) is greeted with the “Diseases Scanner” Page.
3. The actor (user) selects the upload image option.
4. The system opens the user’s gallery.

5. The actor (user) chooses the image they want to scan.
6. The system loads this image into the ImageView.
7. The actor (user) selects the “Classify” button.
8. The system returns the classification result for this image.

Alternative flow

A1: <No internet on device>

1. The device has no internet connection & the system cannot establish a connection to Firebase.
2. The system shows “Connection Error!\n Check your network settings.”
3. (External: The user turns on the Wi-Fi &/or mobile internet)
4. The actor (user) clicks “Refresh” button to establish the connection.
5. <Returns to number 1 in the Main Flow>

A2: <The actor (user) returns to the “Add Plants” page>

1. The actor (user) presses the Back button.
2. The use case continues in Use Case 7: Add Plants.

Exceptional flow

E1: <System errors cause the “Diseases Scanner” process to fail>

1. The system shows the “Diseases Scanner” has failed.
2. The actor (user) must restart the app.
3. The use case continues at *position 1* of the main flow. (See E2)

E2: <System errors cause the “Diseases Scanner” process to fail after restarting>

1. The system shows the failure of the app’s “Diseases Scanner” process.
2. The actor (user) must report this failure to the developer & await their response.
3. The use case continues at *position 1* of the main flow.

Termination

The system presents the apps “Add Plants” Page (Start next Use Case: “Add Plants” Page).

Post condition

The system goes into a wait state.

2.1.1.11. Requirement 10 <Discovery>

The tenth requirement is “Discovery” Page.

2.1.1.11.1. Description & Priority

The use case describes how the user will interact with the “Discovery” Page as they can search for plants inside the Pocket Botanist database & research information pertaining to them.

2.1.1.11.2. Use Case 10: Discovery

Scope

The scope of this use case is to show how the user interacts with the “Discovery” Page.

Description

This use case details the “Discovery” Page of the app & how the user will interact with this page as they can search for specific plants, while also being able to research what they typically look like, some information about them & their typical caretaking requirements.

Flow Description

Precondition

- The user uses a mobile phone & has downloaded the app.
- The user starts the app.
- The user has an account in the system.
- The device is connected to the internet.
- The user has logged into the app.
- The app is defined as the system.
- The user has clicked into the “Discovery” Page.

Activation

This use case begins when an actor (user) has started the app, has already registered an account, logged in & clicked the “Discovery” tab.

Main flow

1. The system displays the “Discovery” Page.
2. The actor (user) is greeted with the “Discovery” Page.
3. The actor (user) selects a plant to learn more about.
4. The system displays brings the user to “Check Plants” page.
5. This use case continues in Use Case 11: Check Plants.

Alternative flow

A1: <No internet on device>

1. The device has no internet connection & the system cannot establish a connection to Firebase.
2. The system shows “Connection Error!\n Check your network settings.”
3. (External: The user turns on the Wi-Fi &/or mobile internet)
4. The actor (user) clicks “Refresh” button to establish the connection.
5. <Returns to number 1 in the Main Flow>

A2: <The actor (user) selects Search function>

1. The actor (user) selects the Search function at the top of the screen.
3. The actor (user) enters the name of the plant they want to find.
4. The actor (user) clicks the Search button, the button with a search icon.
5. The system displays the closest matches to the search characters.
6. <Returns to number 3 in the Main Flow>

Exceptional flow

E1: <System errors cause the “Discovery” process to fail>

1. The system shows the “Discovery” has failed.
2. The actor (user) must restart the app.
3. The use case continues at *position 1* of the main flow. (See E2)

E2: <System errors cause the “Discovery” process to fail after restarting>

1. The system shows the failure of the app’s “Discovery” process.
2. The actor (user) must report this failure to the developer & await their response.
3. The use case continues at *position 1* of the main flow.

Termination

The system presents the apps “Check Plants” Page (Start next Use Case: “Check Plants” Page).

Post condition

The system goes into a wait state.

2.1.1.12. Requirement 11 <Check Plants>

The eleventh requirement is “Check Plants” Page.

2.1.1.12.1. Description & Priority

The use case describes how the user will interact with the “Check Plants” Page as they can search for plants inside the Pocket Botanist database & research information pertaining to them.

2.1.1.12.2. Use Case 11: Check Plants

Scope

The scope of this use case is to show how the user interacts with the “Check Plants” Page.

Description

This use case details the “Check Plants” Page of the app & how the user will interact with this page as they can search for specific plants, while also being able to research what they typically look like, some information about them & their typical caretaking requirements.

Flow Description

Precondition

- The user uses a mobile phone & has downloaded the app.
- The user starts the app.
- The user has an account in the system.
- The device is connected to the internet.
- The user has logged into the app.
- The app is defined as the system.
- The user has clicked into “Discovery” then into the “Check Plants” pages.

Activation

This use case begins when an actor (user) has started the app, has already registered an account, logged in, clicked the “Discovery” tab & clicked into the “Check Plants” page.

Main flow

1. The system displays the “Check Plants” Page.
2. The actor (user) is greeted with the “Check Plants” Page.
3. The system displays the respective plant’s information from the Firebase Realtime Database.
4. The actor (user) reads through the information provided to them.

Alternative flow

A1: <No internet on device>

1. The device has no internet connection & the system cannot establish a connection to Firebase.
2. The system shows “Connection Error!\n Check your network settings.”
3. (External: The user turns on the Wi-Fi &/or mobile internet)
4. The actor (user) clicks “Refresh” button to establish the connection.
5. <Returns to number 1 in the Main Flow>

A2: <The actor (user) clicks the “Add to your plants” button>

1. The actor (user) clicks the “Add to your plants” button.
2. The system brings the user to the “Add Plants” page.
3. The actor (user) adds the plant to their collection.
4. This continues in Use Case 7: Add Plants.

Exceptional flow

E1: <System errors cause the “Check Plants” process to fail>

1. The system shows the “Check Plants” has failed.
2. The actor (user) must restart the app.
3. The use case continues at *position 1* of the main flow. (See E2)

E2: <System errors cause the “Check Plants” process to fail after restarting>

1. The system shows the failure of the app’s “Check Plants” process.
2. The actor (user) must report this failure to the developer & await their response.
3. The use case continues at *position 1* of the main flow.

Termination

This use case is terminated the system successfully displays the correct information to the user before they return to Use Case 10: Discovery.

Post condition

The system goes into a wait state.

2.1.1.13. Requirement 12 <Soil Sensor Device>

The twelfth requirement is Soil Sensor Device.

2.1.1.13.1. Description & Priority

The use case describes how the user will interact with the Soil Sensor Device as they use it to easily monitor the water content of their plant's soil. This product was designed with the main purpose of empowering users to become better caretakers for their plants, without the needed expertise to understand soil moisture principles.

2.1.1.13.2. Use Case 12: Soil Sensor Device

Scope

The scope of this use case is to show how the user interacts with the Soil Sensor Device.

Description

This use case details the Soil Sensor Device which accompanies the app & how the user will interact with this device as they use it to easily monitor the water content of their plant's soil.

Flow Description

Precondition

- The user has purchased the Soil Sensor Device.
- The Soil Sensor Device will be referred to as the device.
- The user has ensured there is a usable battery for the device.

Activation

This use case begins when an actor (user) has acquired the device.

Main flow

1. The actor (user) places a new battery into the device.
2. The device turns on, turning on the LED.
3. The actor (user) places the device into the soil of their plant.
4. The device's LED light turns off, indicating the soil has adequate water content.

Alternative flow

A1: <Device runs out of battery>

1. The device's battery runs out.

2. The device's LED turns off.
3. <Return to number 1 of the Main Flow>

A2: <The device's LED turns on while in the soil>

1. The device's LED turns on while in the plant's soil.
2. The actor (user) notices the LED light has turned on.
3. The actor (user) provides their plant with adequate water.
4. The device's LED light turns off, indicating the soil has adequate water content.
5. <Return to A1 of Alternative Flow>

Exceptional flow

E1: <The device encounters a fundamental error>

1. The device stops working.
2. The actor (user) must contact the Pocket Botanist help email for assistance.

Termination

The device provides the monitoring results to the user, enabling them to better understand their plant's individual caretaking needs.

Post condition

The system goes into a wait state.

2.1.2. Data Requirements

From the perspective of a tester (The User):

- This app's Registration & Login processes requires the user's personal details to be completed.
- This app requires the user to provide their username, email & password.
- This app may require users to grant Camera & Gallery permissions if they intend to use those features which require them.
- This app requires user input for images, titles, & content.

From the perspective of the database (Firebase):

- Firebase stores all the information relevant to users, only the Programmer is privy to this information. As the project creator, they have access.
- The user passwords must be a minimum of 6 characters in length.
- The Programmer must ensure all user provided information is secure. Also, all Firebase Databases are encrypted as to ensure user information is secure.
- All data in the app is sourced from public legal resources e.g. Google Search Engine.
- This app does not require any "special user data" to be accessed.

2.1.3. User Requirements

Based on inputs during testing from users (testers), the following User Requirements are:

- Users are required to have a mobile phone, or android emulators installed on their laptop.
- Users are required to use android OS on their mobile phones.
- Users are required to input their username, email & password during the Registration process in the app.
- Users are required to input an email & password which match their login details for the Login process in the app.

2.1.4. Environmental Requirements

From the perspective of the testers (The User):

- Users are required to have a mobile phone with android OS, or an equivalent emulator installed on their laptop.
- Users should be familiar with how to operate an android mobile device.
- Users are required to operate the app while connected to internet, whether it is Cellular Data or Wi-Fi.

2.1.5. Usability Requirements

Usability Testing:

- Create a “more than sufficient” amount of usability tests prior to allowing users to access the app. In short, the app should be straightforward & simple to use for any user.
- Ensure there is a minimum testing group size of at least 5 individual participants to take the usability tests prior to release, ideally all with different android phone models.

From the retrospective opinion of the testers (The Users):

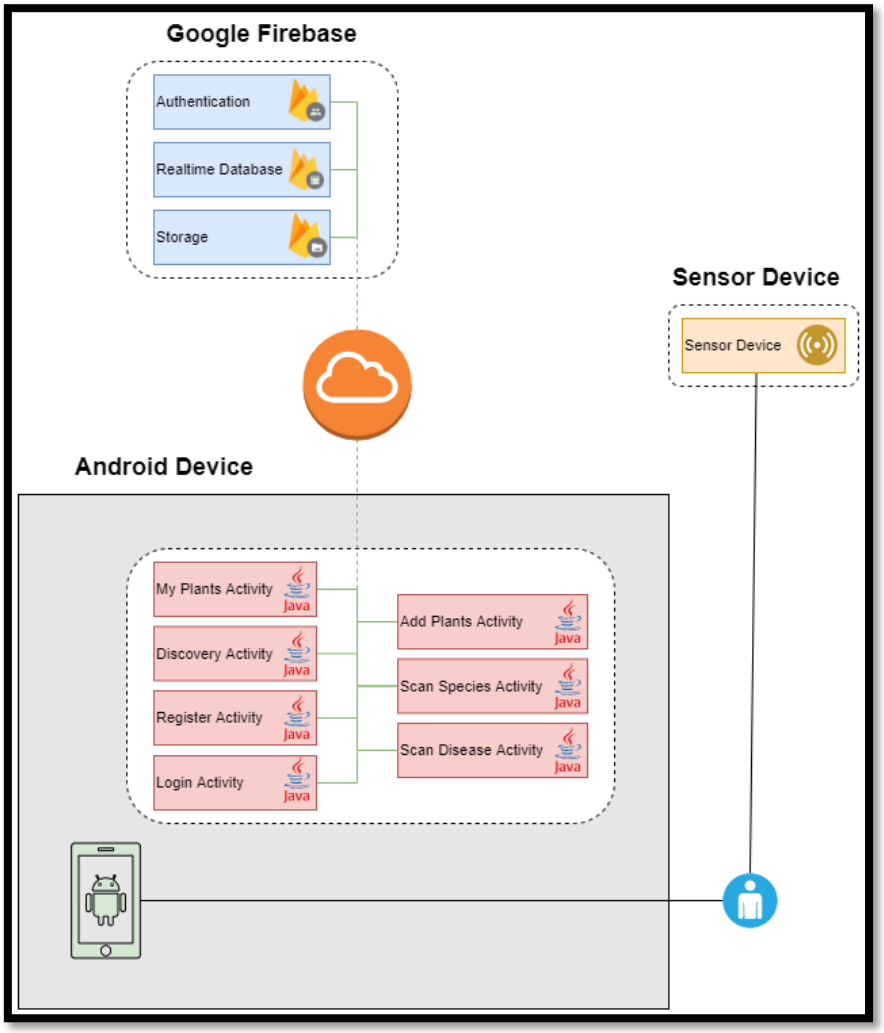
This app must achieve four requirements if it’s to be considered usable: **Easy to learn, Easy to use, Easy to operate & aesthetically appealing, Integrity over Complexity**.

- Ensure the system is easy to learn for new users. This will help improve both user satisfaction & usability of the app.
- Ensure the system is easy to navigate, create the app’s layout & UI with logical reasoning to ensure intuitive operation for new users.
- Ensure the UI is aesthetically appealing to the eye, conducting test groups with colour schemes to discover the most favourable options.
- Ensure the app is downloaded on as many android OS devices as possible.
- Ensure the integrity of the app by focusing on producing a polished product, rather than a project which is more complex but not as polished.

2.2. Design & Architecture

In my project, the main language I used was Java. As a result, I have several Java classes for each function & their respective XML files for their corresponding layouts. The overall Design & Architecture I decided upon for this project is a minimalist one, choosing more of a "Dark Mode" colour scheme for my universal colours. Technically, the first class the user encounters is the Splash Screen, but this only lasts 1000 milliseconds as it shows the logo. For the overall theme of the project, I have selected a series of "Dark Mode" hex colours to utilise in my colour scheme, this is in order to present an aesthetically pleasing atmosphere along with green accents to reflect the plant nature of the app. I strived to create an app which would be enjoyed by users & commercially viable.

In this project, I have fully created this app with the use of Android Studio while also testing the app on my own personal mobile phone, which is a Pixel 5 running the latest version of Android OS. Android Studio was used to develop the app itself, but Firebase has been utilised for its Authentication, Realtime Database & Storage services. This will enable users to register accounts & store information inside of Pocket Botanist. All of this data is encrypted within the Firebase service & is accessible by the Programmer.



Draw.io Diagram 2

2.3. Implementation

In this section I will explain in detail, with screenshots, how I created each function which are included in the Pocket Botanist app. In my project the three main corner stones are the implementations for Firebase, Glide & TensorFlow. TensorFlow was chosen due to its wide variety in tools & library resources for image processing technology, while Firebase was chosen for its unique combination of complexity & usability as it's easily implementable. Glide was chosen for its fast & efficient media management capabilities. To implement these, I needed to include the relevant dependencies, as show below:

```
dependencies {
    // Base
    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
    implementation 'androidx.navigation:navigation-fragment:2.4.2'
    implementation 'androidx.navigation:navigation-ui:2.4.2'
    implementation "com.android.support:support-annotations:28.0.0"
    annotationProcessor "com.android.support:support-annotations:28.0.0"

    // Firebase
    implementation platform('com.google.firebase:firebase-bom:29.3.1')
    implementation 'com.google.firebase:firebase-database:20.0.4'
    implementation 'com.google.firebase:firebase-auth:21.0.3'
    implementation 'com.google.firebase:firebase-storage:20.0.1'
    implementation 'com.firebaseui:firebase-ui-storage:6.2.1'
    implementation 'com.google.firebase:firebase-analytics'

    // Glide
    implementation 'com.github.bumptech.glide:glide:4.13.0'
    annotationProcessor 'com.github.bumptech.glide:compiler:4.13.0'

    // TensorFlow
    implementation('org.tensorflow:tensorflow-lite:0.0.0-nightly') { changing = true }
    implementation('org.tensorflow:tensorflow-lite-gpu:0.0.0-nightly') { changing = true }
    implementation('org.tensorflow:tensorflow-lite-support:0.0.0-nightly') { changing = true }

    // Testing
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

Next, I will separate each feature into their own individual sections:

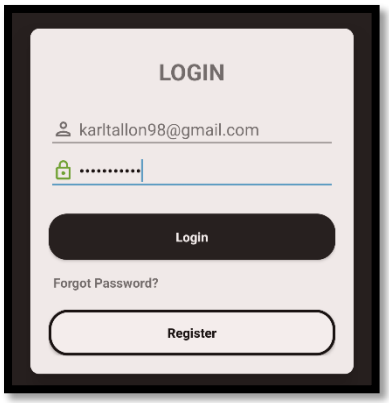
2.3.1. Firebase

Over the past 4 years in NCI, I have used Firebase on multiple occasions within my Android applications. The Firebase platform provides developers with some amazing tools for no cost when used in smaller scales projects. In my project, I utilised Firebase's Authentication, Realtime Database & Storage services to help bolster the functionality of Pocket Botanist. Android Studio allows for easy integration of Firebase through their Firebase Assistant. With this assistant, developers can easily integrate a multitude of Firebase services, whether its authenticating user login information or storing user data such as Plant entries.

2.3.2. Login

Through the usage of Firebase Authentication, I was able to more efficiently create an encrypted login service, helping ensure more safe usage for users. The Login page, as shown below, consists of 2 TextFields for the email & password then 3 buttons for "Login", "Forgot Password?" & "Register". There are error handlers for ensuring users enter their details correctly, as shown below:

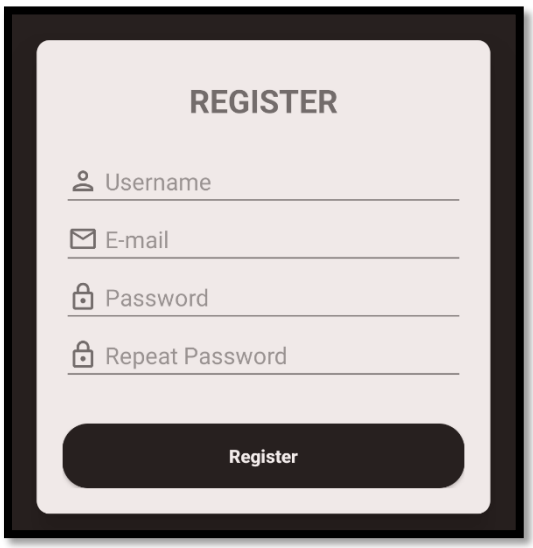
```
private boolean checkLoginRequirements(String email, String password) {
    boolean errorsFree = true;
    if (email.isEmpty()) {
        loginView.setEmailError("This field can't be empty.");
        errorsFree = false;
    } else if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
        loginView.setEmailError("This field can't be empty.");
        errorsFree = false;
    }
    if (password.isEmpty()) {
        loginView.setPasswordError("This field can't be empty.");
        errorsFree = false;
    }
    return errorsFree;
}
```



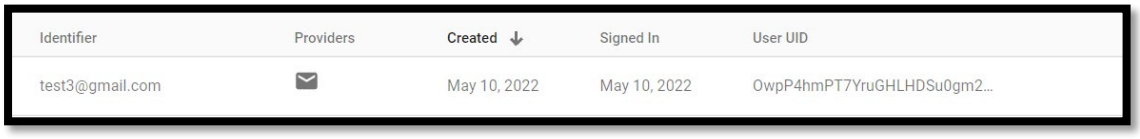
2.3.3. Registration

For the Registration process, I am again utilising Firebase Authentication to allow users to register with Pocket Botanist. The Register Page, as shown below, consists of 4 TextFields for the username, email, password & confirm password then 1 button for "Register". There are error handlers for ensuring users enter their details correctly, as shown below:

```
public boolean checkRegisterRequirements(
        String username, String email,
        String password, String repeatedPassword) {
    boolean isCorrect = true;

    if (username.isEmpty()) {
        registerView.setUsernameError("This field can't be empty.");
        isCorrect = false;
    }
    if (email.isEmpty()) {
        registerView.setEmailError("This field can't be empty.");
        isCorrect = false;
    } else if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
        registerView.setEmailError("This is not a valid email.");
        isCorrect = false;
    }
    if (password.isEmpty()) {
        registerView.setPasswordError("This field can't be empty.");
        isCorrect = false;
    } else if (password.length() < 6) {
        registerView.setPasswordError("Password too short! At least 6 char.");
        isCorrect = false;

    } else if (!password.equals(repeatedPassword)) {
        registerView.setRepeatedPasswordError("Passwords didn't match");
        isCorrect = false;
    }
    return isCorrect;
}
```



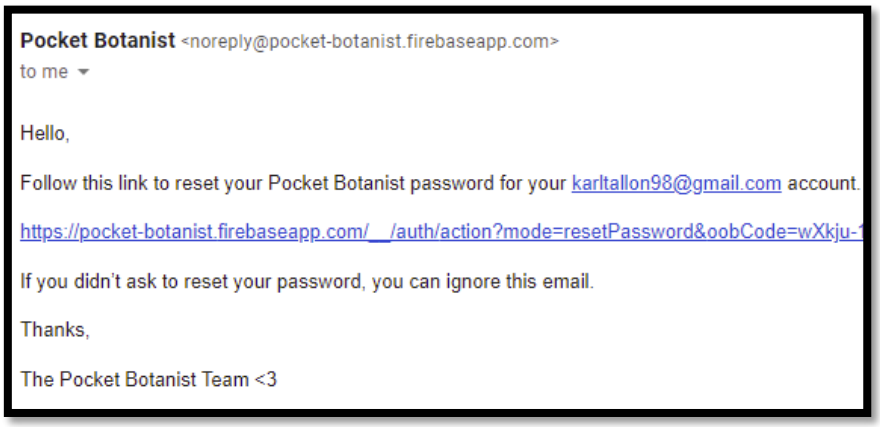
Once users have successfully registered through the Pocket Botanist app, the app states "User successfully registered", then their information is stored within the Firebase Realtime Database:



Identifier	Providers	Created ↓	Signed In	User UID
test3@gmail.com	✉	May 10, 2022	May 10, 2022	OwpP4hmPT7YruGHLHDSu0gm2...

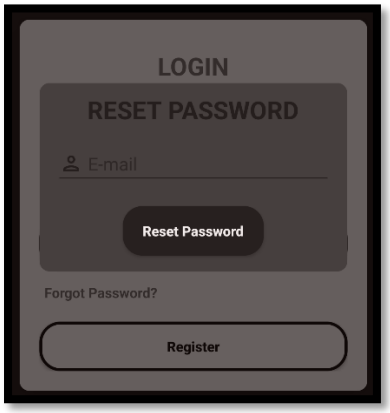
2.3.4. Forgot Password

The Forgot Password feature once again utilises Firebase & allows for easy changing of passwords through emails which are sent directly to users, ensuring a more secure password recovery process. Below is an example of the email which users receive:



The code snippet below shows how the process is performed. The app handles sending the request for an email to be sent, then it displays a message states "Email sent" to inform the customer of the successful request. Next, the user receives the above email & they're directed to a Firebase webpage when they enter a new password which is then updated within the Pocket Botanist database.

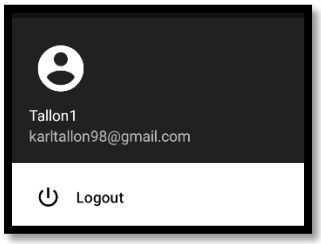
```
public void performResetPassword(String email) {
    forgotListener.onStart();
    FirebaseAuth.getInstance().sendPasswordResetEmail(email)
            .addOnSuccessListener(task -> {
                    forgotListener.onEnd();
                    forgotListener.onSuccess("Email sent.");
            })
            .addOnFailureListener(error -> {
                forgotListener.onEnd();
                forgotListener.onFailure(error.getMessage());
            });
}
```



2.3.5. Logout

The Logout feature is simple yet effective as it only requires a small bit of code. Below we see the code used to perform this function. It once again utilises the Firebase framework to properly log users out of their account once they click the "Logout" button.

```
public void performLogout() {
    mainListener.onEnd();
    FirebaseAuth.getInstance().signOut();
    mainListener.onFailure();
}
```



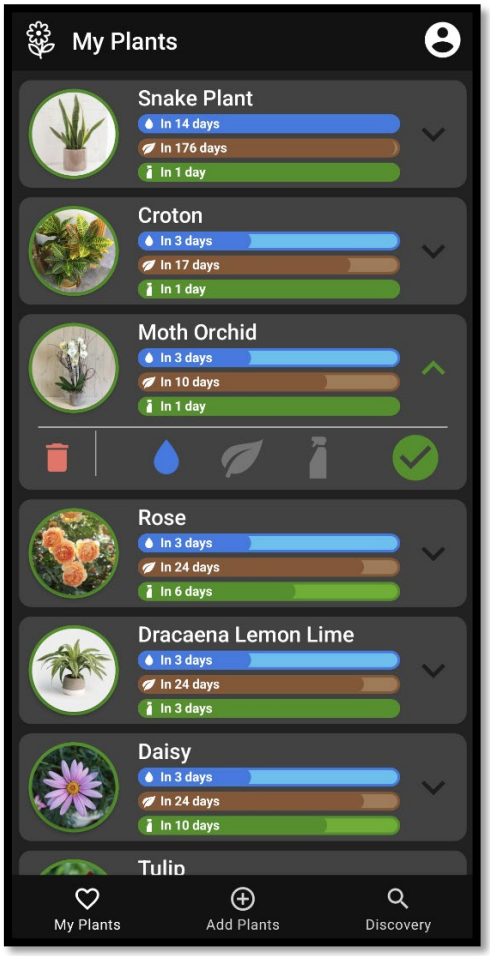
2.3.6. My Plants

The “My Plants” Fragment also makes use of Firebase as it utilises their Realtime Database service. Once the user has added their first plant through the “Add Plants” page, their plant will be added with their own UserPlants collection which is stored on Firebase. An example of this can be seen below:

Once these plants have been logged into the Firebase Realtime Database, the app will display these within the “My Plants” page. This is performed by the fetchMyPlantList function, this function checks the user ID, pulls the corresponding UserPlants array from Firebase & displays it for the user to see.

```
public void fetchMyPlantList() {
    myPlantsListener.onStart();
    List<UserPlant> plantList = new ArrayList();

    FirebaseDatabase.getInstance().getReference( path: "Users")
            .child(Objects.requireNonNull(FirebaseAuth.getInstance().getUid())).child("UserPlants")
            .addValueEventListener(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot snapshot) {
                    plantList.clear();
                    for (DataSnapshot ds : snapshot.getChildren()) {
                        UserPlant plant = ds.getValue(UserPlant.class);
                        plantList.add(plant);
                    }
                    myPlantsListener.onEnd();
                    myPlantsListener.onSuccess(plantList);
                }

                @Override
                public void onCancelled(@NonNull DatabaseError error) {
                    myPlantsListener.onEnd();
                    myPlantsListener.onFailure(error.getMessage());
                }
            });
}
```

Below is an example use case of a user who has multiple has plants, all with their own individual needs & dates at which they need to be met. The red exclamation mark beside the Sunflower plant means one of its requirements need to be taken care of. The user is able to select one of the following options: Delete, Water, Fertilize, Spray & Confirm.



An example of one of the above button's codes can be seen below. The Delete button allows users to remove unwanted plants from their own collections & that change is reflected within the Firebase Realtime Database.

```
public void performDeletePlant(UserPlant plant) {
    myPlantsListener.onStart();
    DatabaseReference ref = FirebaseDatabase.getInstance().getReference()
            .child("Users").child(FirebaseAuth.getInstance().getUid()).child("UserPlants");
    ref.child(plant.getId()).removeValue()
            .addOnSuccessListener(task -> {
                myPlantsListener.onEnd();
                myPlantsListener.onPlantDeleted(plant.getId());
            })
            .addOnFailureListener(error -> {
                myPlantsListener.onEnd();
                myPlantsListener.onFailure(error.toString());
            });
}
```

For the “Update” buttons, the below code is responsible for publishing the updated data to the Firebase database. These requirements are tracked based on the data which they were last updated. Once the requirements reach the date at which they need to be taken care of, Pocket Botanist will send a push notification to the user’s mobile phone to alert them of the plant’s pressing needs.

```
public void performUpdatePlantNeeds(UserPlant plant, boolean isWatered, boolean isFertilized, boolean isSprayed) {
    myPlantsListener.onStart();
    DatabaseReference ref = FirebaseDatabase.getInstance().getReference()
            .child("Users").child(FirebaseAuth.getInstance().getUid())
            .child("UserPlants").child(plant.getId());
    String now = getCurrentDate();

    List<Task> taskList = new ArrayList<~>();

    if (isWatered) {
        taskList.add(ref.child("lastWatering").setValue(now));
    }
    if (isFertilized) {
        taskList.add(ref.child("lastFertilizing").setValue(now));

    }
    if (isSprayed) {
        taskList.add(ref.child("lastSpraying").setValue(now));

    }

    Tasks.whenAllSuccess(taskList.toArray(new Task[0]))
            .addOnSuccessListener(task -> {
                myPlantsListener.onEnd();
                plant.setLastWatering(now);
                plant.setLastFertilizing(now);
                plant.setLastSpraying(now);
                myPlantsListener.onSuccess(plant);
            })
            .addOnFailureListener(error -> {
                myPlantsListener.onEnd();
                myPlantsListener.onFailure(error.getMessage());
            });

}
```

Below is an example of the aforementioned screenshot which the Pocket Botanist app will send to its users to give them a gentle reminder.

2.3.7. Add Plants

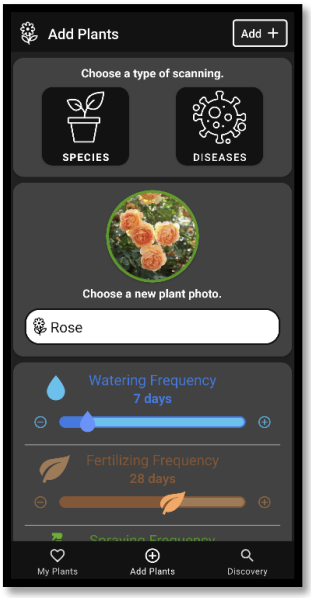
The “Add Plants” Fragment also makes use of Firebase as it utilises their Realtime Database service but also uses the Glide framework to enable more effective media management within the Pocket Botanist app. Once the user has inputted all of the required information, they may press the “Add +” button which then submits that new plant data to the Firebase Realtime Database which can then be viewed on the “My Plants” page. The below code allows for the new data to be submitted into that individual user’s UserPlants array list. The app will also display a message informing the user than the plant has been successfully added.

```
private void addPlant(UserPlant plant) {
    DatabaseReference databaseReference = FirebaseDatabase.getInstance() FirebaseDatabase
            .getReference( path: "Users") DatabaseReference
            .child(Objects.requireNonNull(FirebaseAuth.getInstance().getUid()))
            .child("UserPlants");

    databaseReference.child(plant.getId()).setValue(plant)
            .addOnSuccessListener(task -> {
                addListener.onEnd();
                addListener.onSuccess("The plant has been added", plant);
            })
            .addOnFailureListener(error -> {
                addListener.onEnd();
                addListener.onFailure(error.getMessage());
            });
}
```

In the case of plants that are added with a photo attached, these images will be stored within the Firebase Storage service under the folder “plant_images”. This means those plant images will always be available for display even if the user deletes the image from their own mobile phone storage.

```
private void addPlantWithImage(UserPlant plant) {
    String filePathAndName = "plant_images/" + plant.getId();
    StorageReference storageReference = FirebaseStorage.getInstance().getReference(filePathAndName);
    storageReference.putFile(Uri.parse(plant.getImage()))
            .addOnSuccessListener(task -> {
                addPlant(plant);
            })
            .addOnFailureListener(error -> {
                addListener.onEnd();
                addListener.onFailure(error.getMessage());
            });
}
```

Below is an example use case of a user who is inputting a Rose to their personal collection. They have inputted the image, name & frequency for its individual requirements. At the top of the screen, we can see the buttons for the Scanner pages, which will be discussed in the next sections.

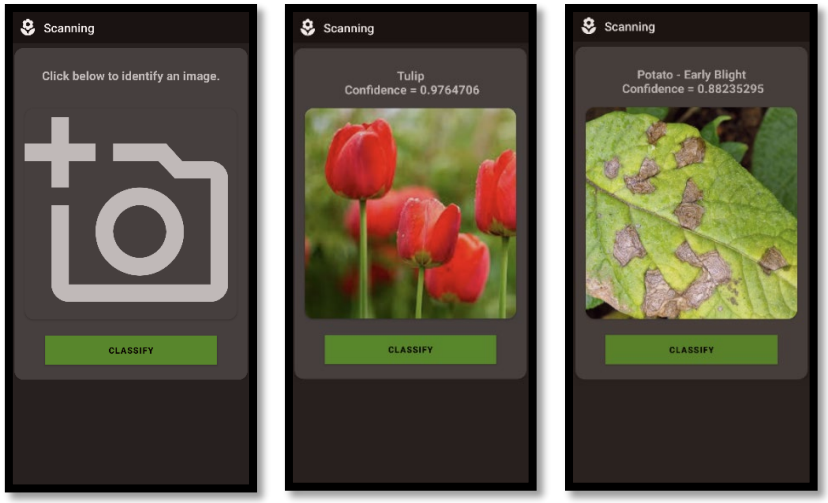


Below shows the permissions requirements which Pocket Botanist follows according to the user's choice of which image uploading method they select.

```
private void showChoosePhotoDialog() {
    final CharSequence[] options = getResources().getStringArray(R.array.photo_options);

    AlertDialog.Builder builder = new AlertDialog.Builder(requireContext());
    builder.setTitle("Choose plant photo");

    builder.setItems(options, (dialog, item) -> {

        if (options[item].equals("Take Photo")) {
            tryPickPhotoFromCamera();

        } else if (options[item].equals("Choose from Gallery")) {
            tryPickPhotoFromGallery();

        } else if (options[item].equals("Cancel")) {
            dialog.dismiss();
        }
    });
    builder.show();
}

private void tryPickPhotoFromCamera() {
    if (getActivity().checkSelfPermission(Manifest.permission.CAMERA) ==
            PackageManager.PERMISSION_DENIED) {
        String[] permission = {Manifest.permission.CAMERA};
        requestPermissions(permission, PERMISSION_CAMERA);
    } else {
        pickPhotoFromCamera();
    }
}

private void tryPickPhotoFromGallery() {
    if (getActivity().checkSelfPermission(Manifest.permission.READ_EXTERNAL_STORAGE) ==
            PackageManager.PERMISSION_DENIED) {
        String[] permission = {Manifest.permission.READ_EXTERNAL_STORAGE};
        requestPermissions(permission, PERMISSION_STORAGE);
    } else {
        pickPhotoFromGallery();
    }
}
```

2.3.8. Species & Diseases Scanners

The below screenshots show the Scanners in operation. On the left, we have the starting page which is awaiting the user's input of an image for analysis. In the next two, we have the results for those inputted images. The app takes the user's inputted image, compares it against thousands of images before returning the result which can be seen at the top. The name of the plant will be displayed along with the Confidence rating, which is a percentage result. Taking the middle use case as our example, the Pocket Botanist app has classified that image as being of Tulips & it is 97% confident that this is the case. The Species & Disease features are split into their own pages with their own individual TensorFlow models as to reduce analysis error. As of the current version of Pocket Botanist, the datasets for each function contain just a few examples so as to act as proof-of-concept for the app.



Taking the Species Scanner function as an example, we can dissect the code as seen in the below screenshots. The below screenshot shows the loadImage function which is responsible for properly loading the user's inputted image into the correct dimension before being processed then displayed in the app alongside the results. The loadModelFile is responsible for accessing the TensorFlow model, which in this case is the Species model, so that the loaded image may be analysed against this model.

```
private TensorImage loadImage(final Bitmap bitmap) {
    // Loads bitmap into a TensorImage.
    inputImageBuffer.load(bitmap);

    // Creates processor for the TensorImage.
    int cropSize = Math.min(bitmap.getWidth(), bitmap.getHeight());
    ImageProcessor imageProcessor = new ImageProcessor.Builder()
                    .add(new ResizeWithCropOrPadOp(cropSize, cropSize))
                    .add(new ResizeOp(imageSizeX, imageSizeY, ResizeOp.ResizeMethod.NEAREST_NEIGHBOR))
                    .add(getPreprocessNormalizeOp())
                    .build();
    return imageProcessor.process(inputImageBuffer);
}

private MappedByteBuffer loadModelFile(Activity activity) throws IOException {
    AssetFileDescriptor fileDescriptor = activity.getAssets().openFd( fileName: "speciesModel.tflite");
    FileInputStream inputStream = new FileInputStream(fileDescriptor.getFileDescriptor());
    FileChannel fileChannel=inputStream.getChannel();
    long startOffSet = fileDescriptor.getStartOffset();
    long declaredLength = fileDescriptor.getDeclaredLength();
    return fileChannel.map(FileChannel.MapMode.READ_ONLY,startOffSet,declaredLength);
}
```

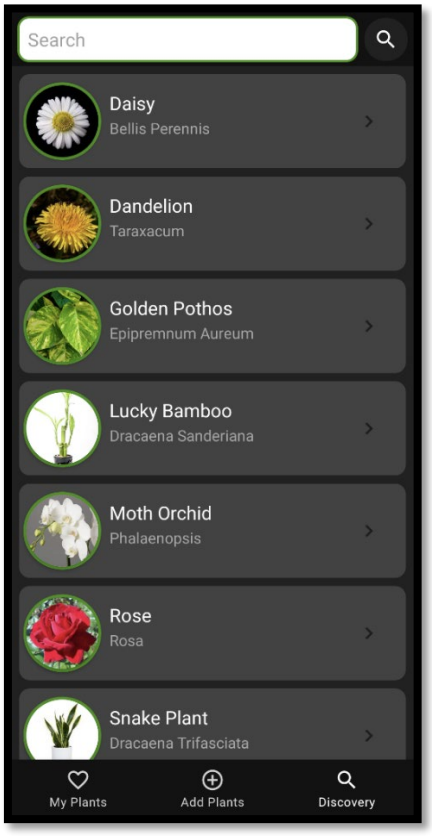
The below screenshot shows the showResult function within the Species Scanner class. This function is responsible for printing the result for the analysed image, the result will contain the species of the plant & how confident the app is in its analysis of the user's image.

```
private void showResult() {

    try {
        labels = FileUtil.loadLabels( context: this,  filePath: "speciesDict.txt");
    } catch (Exception e){
        e.printStackTrace();
    }
    Map<String, Float> labeledProbability = new TensorLabel(labels, probabilityProcessor.process(outputProbabilityBuffer)).getMapWithFloatValue();
    float maxValueInMap = (Collections.max(labeledProbability.values()));

    for (Map.Entry<String, Float> entry : labeledProbability.entrySet()) {
        if (entry.getValue() == maxValueInMap) {
            classifyText.setText(entry.getKey() + "\n Confidence = " + maxValueInMap);
        }
    }
}
```

The datasets used to train my Species (TensorFlow, 2020) & Diseases *(J and Gopal, 2019)* TensorFlow models can be found in my references.

2.3.9. Discovery

The screenshot below shows the Discovery page of the app. On this page users can search for new plants which are contained within the Pocket Botanist database. Much like the "My Plants" page, these plants are all stored within the Firebase Realtime Database. The Discovery page only displays a portion of the information stored within the Plants array. This includes the plant's image, common name & Latin name. The user is able to click into each of these plant entries & also can search for any plant using the search function at the top of the screen.



The below function operates much like the “My Plants” function of “fetchMyPlantList”, this function retrieves all of the data contained within the “Plants” array list & displays it accordingly.

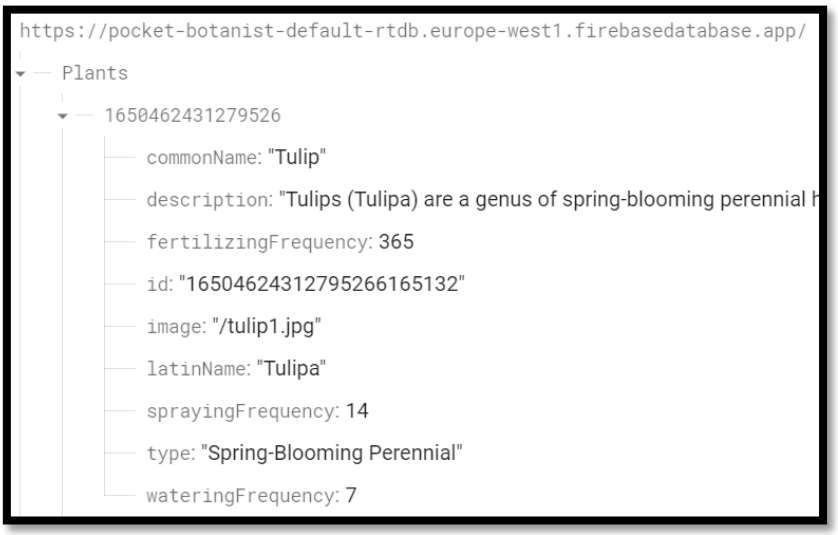
```
public void performGetAllPlants() {
    discoverListener.onStart();
    List<Plant> plantList = new ArrayList<>();
    FirebaseDatabase.getInstance().getReference( path: "Plants")
            .limitToFirst(50)
            .addValueEventListener(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot snapshot) {
                    for (DataSnapshot ds : snapshot.getChildren()) {
                        Plant plant = ds.getValue(Plant.class);
                        plantList.add(plant);
                    }
                    discoverListener.onEnd();
                    discoverListener.onSuccess(plantList);
                }

                @Override
                public void onCancelled(@NonNull DatabaseError error) {
                    discoverListener.onEnd();
                    discoverListener.onFailure(error.getMessage());
                }
            });
}
```

The below function operates as the code behind the aforementioned search function. It displays the search results, ordered in descending alphabetical order.

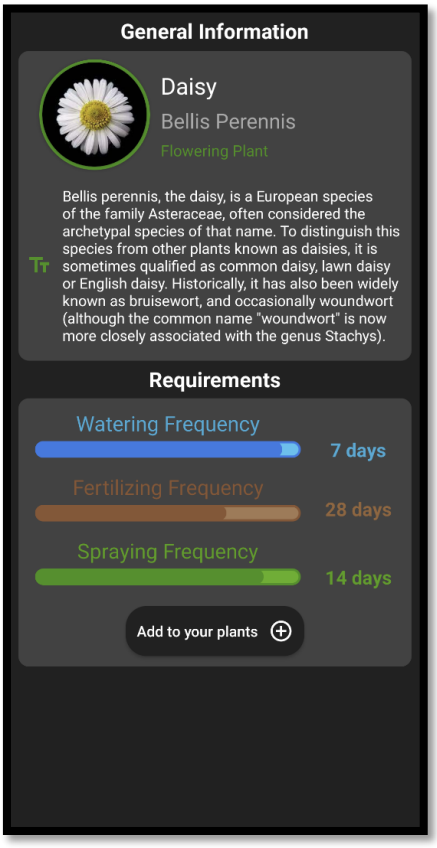
```
public void performGetMatchingPlants(String regex) {
    discoverListener.onStart();
    List<Plant> plantList = new ArrayList<>();
    FirebaseDatabase.getInstance().getReference( path: "Plants") DatabaseReference
            .orderByChild("commonName") Query
            .startAt(regex)
            .endAt(regex + "\uf8ff")
            .addValueEventListener(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot snapshot) {
                    for (DataSnapshot ds : snapshot.getChildren()) {
                        Plant plant = ds.getValue(Plant.class);
                        plantList.add(plant);
                    }
                    discoverListener.onEnd();
                    discoverListener.onSuccess(plantList);
                }

                @Override
                public void onCancelled(@NonNull DatabaseError error) {
                    discoverListener.onEnd();
                    discoverListener.onFailure(error.getMessage());
                }
            });
}
```

2.3.10. Check Plants

The “Check Plants” page is what the user will find when they click into one of the plant entries from the Discovery page. The below screenshot shows the full information for the plants shown on these pages.

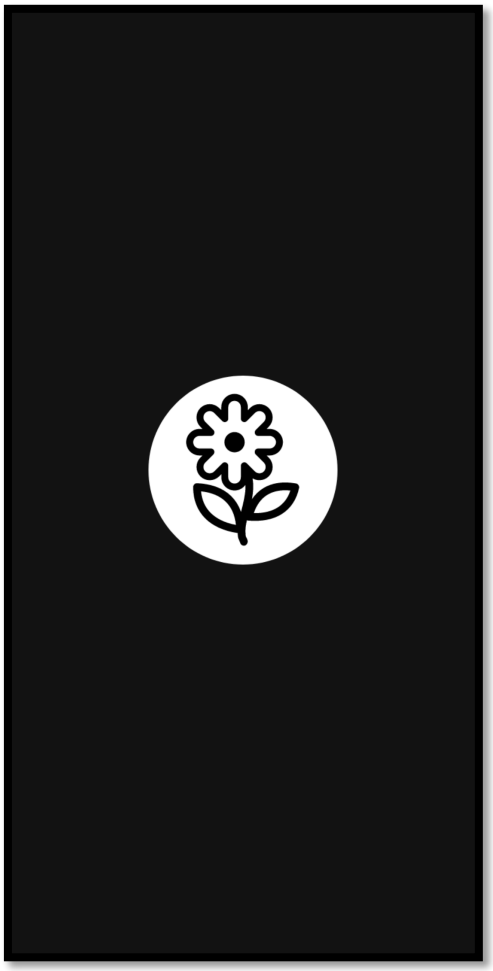


While the Discovery page only displays a portion of the plant information, this page displays all of the relevant information for the given plant entry. These plants are manually entered into the Firebase Realtime Database while their images are hosted in the Firebase Storage service. The user can use the “Add to your plants” button to be brought back to the “My Plants” section to add this plant for themselves.



2.3.11. Splash Screen

The below screenshot shows the splash screen of Pocket Botanist, this screen welcomes users while displaying the app's logo for 1000 milliseconds before the user is brought to either the Login page or the "My Plants" page, depending on whether this is their first time in the app or not.



2.3.12. Soil Sensor Device

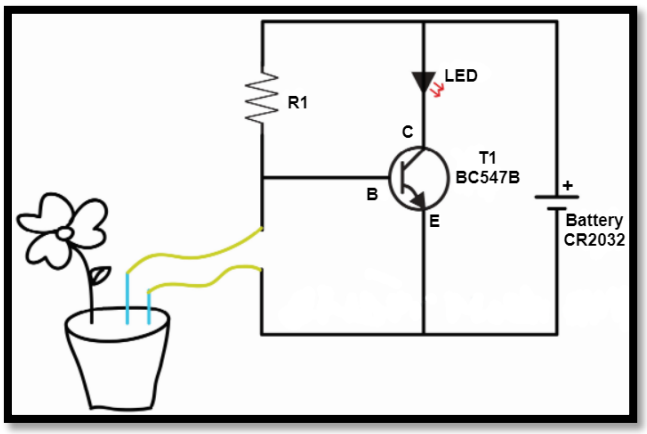
The accompanying hardware which I created to complement the Pocket Botanist app has been create by me through the process of soldering together my own device. All of my parts were sourced from within Ireland from a wonderful company called https://irishelectronics.ie/. This was a lengthy & difficult endeavour as I learnt about electronics prototyping but it was extremely enjoyable & I am very pleased with it.

Below are the required components & price per unit (€):

1. 470K TruOhm 0.25 Watt Resistor (13 cent)
2. BC547B NPN General Purpose Transistor (30 cent).
3. 3v CR2032 Battery (€1.2)
4. Battery Holder for LI-Cell Ø 19mm (CR2032) (€1.15).
5. LED Lamp Holder Ø5mm (€1.29).
6. 5mm LED (20 cent).

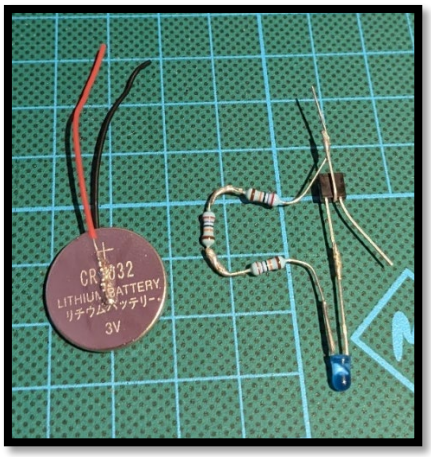
Using the above prices, the total cost for components is €4.27. This price could be drastically reduced from economy of scale if we made use of bulk purchasing from a wholesaler. The additional cost of the 3D printing could be greatly reduced by purchasing our own for prototyping then employing a factory, in the likes of China, to produce a more refined version of the product. With these low projected costs for production, I believe there is massive potential for lucrative turnover.

The below image is the device's circuit diagram which depicts the actual wire connections & components that are used. The principle behind this device is that there is an open electrical circuit which allows the LED to stay lit, once both metal probes make contact with something conductive at the same time, the LED will turn off. It is through testing this principle that I have been able to successfully use this device in plants & monitor their soil moisture levels.

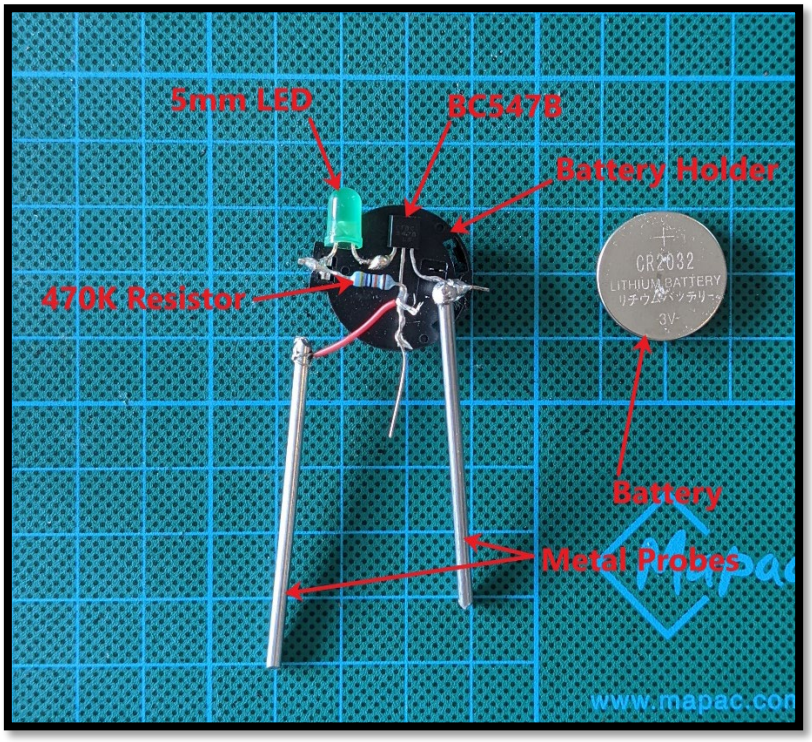


Draw.io Diagram 3

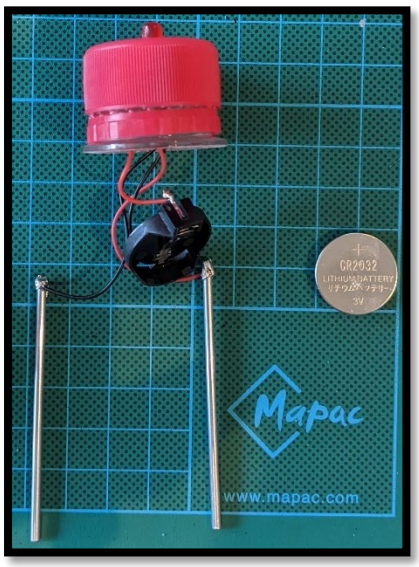
The early stages of prototyping began as a basic configuration while I figured out how to properly connect the components & ensure they worked together. Below is an image of the first prototype I created, I initially used the wrong Resistors, using three 270K Resistors joined together & instead I swapped to a single 470K Resistor for the next version. On top of this, I was soldering directly onto the battery, but I quickly corrected such an amateur mistake. I disconnected the battery as this was not a safe iteration of my sensor. During this phase, my two metal probes were two 10cm wood screws.

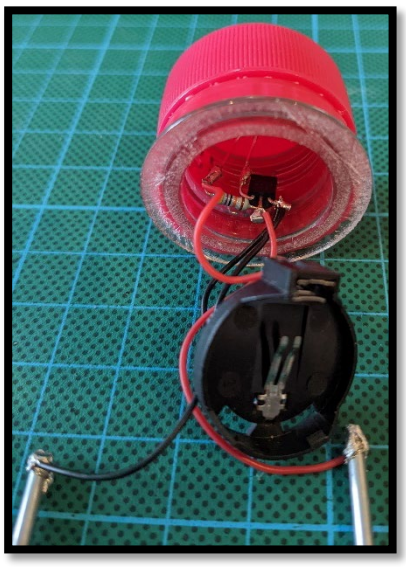


Below is the next iteration of the device, I had ordered some additional pieces like the battery holder & proper resistors. This was a much safer version of the device as I was able to safely hold the battery, which also meant I am able to replace it in case the battery runs out. This is the general layout of my device's components & connections as of right now, in future versions I only extended the connections with some 26-gauge wire.

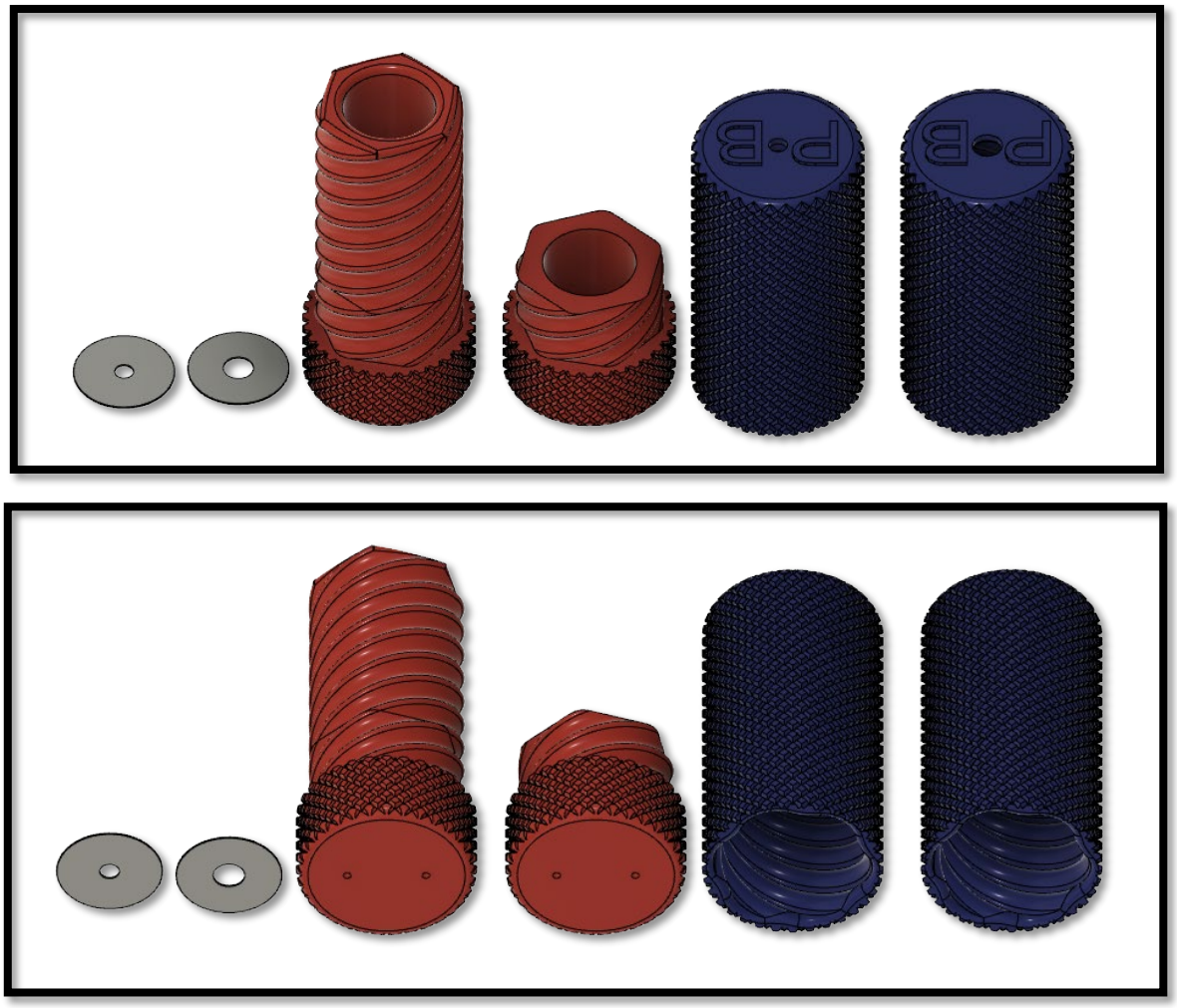


The below images show the testing I conducted for placing the device within a housing. The housing consisted of two Coke bottle tops which I sawed off, sanded down & drilled the appropriately sized holes into each end. Initially I was using screws as the metal probes, but after further testing I found trimmable lengths of solid & conductive wire to use. In the rightmost image we can see the housing fully closed, containing all of the components, with its LED turned on.





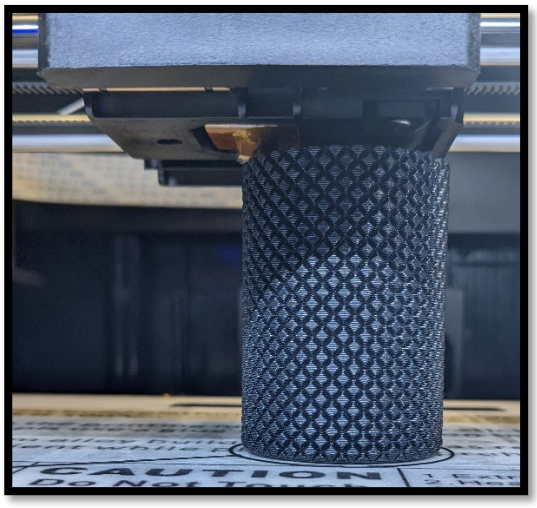
With this testing completed, I moved onto designing a real housing which could be 3D printed to reflect the commercially viable image I had in my mind. Below is a screenshot of the model, top & bottom views, which I designed using Fusion360:



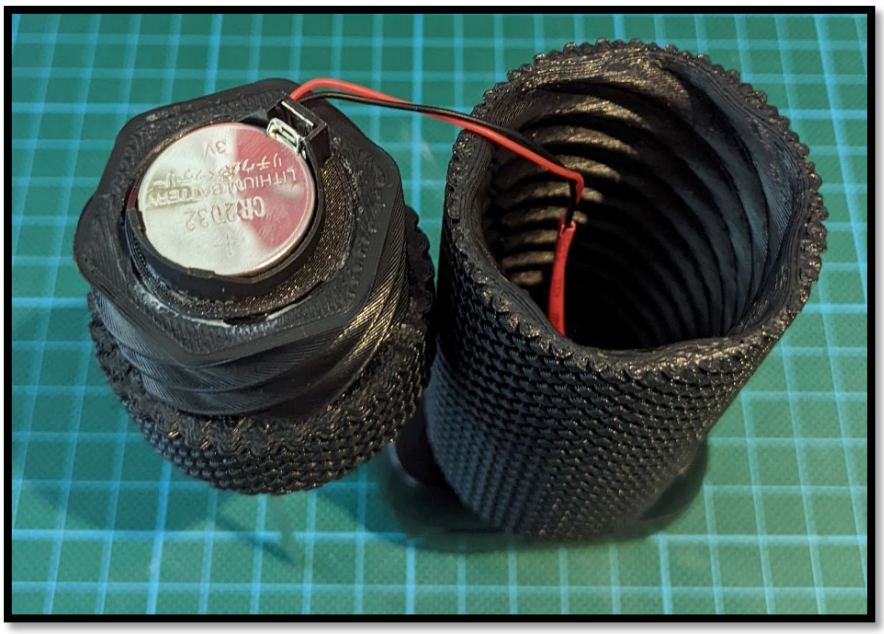
Once I had produced the designs, I got in contact with a fantastic 3D printing service based in Dublin called Suir3D. After back & forth discussions with Sean from Suir3D, we came to the decision to print my device's housing with ASA filament, which is the "*most UV resistant material in 3D printing – ten times more weather resistant and UV-resistant than ABS" (Toor, 2021)* & logically makes sense for the purpose of this device where it would typically be in direct sunlight for hours per day.

The service of 3D printing is an expensive process, typically clocking in at €70 per 24 hours. Since it was so expensive, I had one shot to get the design right. I was able to achieve an aesthetically pleasing device, at least in my eyes, even though it is slightly larger than I had intended. This is one of the aspects which I plan to change in future versions of my device.

With all of these details, Sean got to work printing my design at their 3D printing labs based near The Point Village. The PB engraving stands for Pocket Botanist! Below are some screenshots of the printing process:



Once I had acquired all of the printed pieces, I got to work installing my device into the new housing. As we can see below, this is the finished version of the Pocket Botanist device. All of the aforementioned components are stored in the base of the device, leaving access to the 3v battery for changing & connects to the LED at the top of the device. To replace the battery, the user must simply untwist the housing & replace it. In future, I would incorporate a waterproof access-port for a USB-C cable to charge a rechargeable battery. When I got the housing from the 3D printers, I realised the size could be drastically reduced. In future, I will be redesigning the device to be a fraction of the size. This will allow us to produce more for less cost, while also producing a more aesthetically pleasing device for consumers. Also I plan to provide three different types of device, each with different length metal probes such as Small or Large, to cater to any type of plant.



Finally, I had completed my device & it was ready for testing. Below is an image of my device:

I had already tested the previous prototypes but below is a demonstration of the device's functionality. On the left you'll see the control test, where a glass of water is used to demonstrate the principle of this device in play. On the right you'll see the device being demonstrated using a plant whose soil has already been sufficiently watered. Please note the LED light at the top of the device, this is the indicator for when the soil is too dry or not. If the LED is on then the soil is too dry, if the LED is off then the soil has sufficient moisture levels. Please double click each link to view the videos.

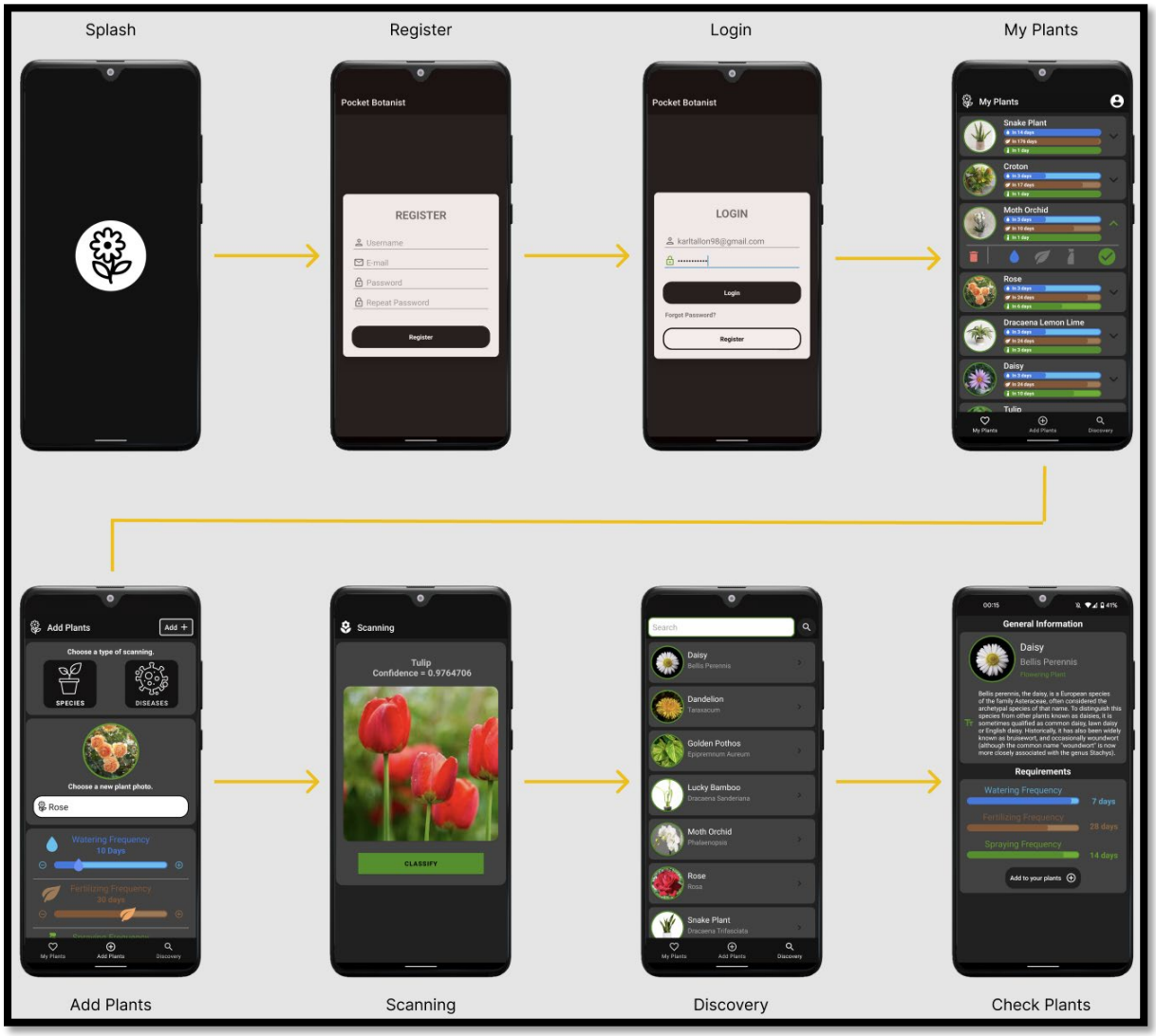
Plant Water Test.mp4 In-Plant Demo.mp4

2.4. Graphical User Interface (GUI)

The below diagram demonstrates each of the key screens found within Pocket Botanist. For a brand-new user, going from the Splash Screen to the Login Screen is a linear path as they must register an account before accessing Pocket Botanist. Once the user has accessed the app, they may choose any one of the three tabs, located at the bottom of the screen, to direct them to the three major activities: My Plants, Add Plants & Discovery. Each activity presents its own unique features which empowers users to become better botanists.

For first time users who has logged in, there is somewhat of an underlying linear path to be taken as they need to research the plant they want to add & its needs in the “Discovery” page, then add that first plant inside of the “Add Plants” page, then return back to the “My Plants” page to view their newly added plant collection.

Once users have already established a profile with Pocket Botanist, their app usage will vary as they return for specific features more than others, depending on their personal needs. This was observed during testing as certain users utilised particular features more than others, reportedly since some users are more interested in the image classification than plant research, or vice versa.



2.5. Testing

Throughout this entire project, I have continuously conducted my own testing so that I could confirm the functionality of each function within both my app & the accompanying device. In this section, I will be discussing both the focus group testing, which was conducted using test users, but also the performance testing which was conducted through Android Studio.

2.5.1. Focus Group – Usability Testing

All of the below tests were performed by both the developer (myself) & a focus group of 5 test users, all with individual accounts. All of the test users have various levels of knowledge relating to both botany & android mobile usage, with one user being a certified botanist & landscape architect, while another two users are Computer Science graduates. Though all of the test users gave me valuable feedback for the project, these three users in particular allowed for a greater insight into both the practicality & functionality of my project. All testers will be testing the same version of Pocket Botanist, which they are all seeing for the first time & will be given no guidance on how to navigate the app. I used Android Studio to install the application on each user's Android device & sat beside them to observe if tests passed, the devices used will be listed beside their names. Once the tests were over, I asked the testers for their feedback on each requirement. The users were asked if they found the tests easy & straightforward or difficult & unintuitive. The requirements for these tests are as follows:

1. **Registering**: If the user can successfully register & their information is saved to Firebase by the system, the test passes.
2. **Login**: If the user can successfully login, the test passes.
3. **Add Plants**: If the user can successfully add a plant to their collection, with all of its requirements & an image, the test passes.
4. **Scanning**: If the user can successfully return the appropriate result from both the Species & Disease scanner, the test will pass.
5. **My Plants**: If the user can successfully view the newly added plant within their collection, then update its requirements, the test passes.
6. **Discovery**: If the user can successfully view the Pocket Botanist plant database, then access one of those entries for further information, the test passes.
7. **Device**: If the user can successfully operate the device with little guidance, the test passes.

Tester 1 (OnePlus Nord 2):

1	Registration was clear.
2	Logging in was standard.
3	Adding plants was simple to do, the page is intuitively laid out.
4	Surprisingly accurate & extremely quick at identifying a dandelion in my garden.
5	Plant collections are clear to view & easy to update.
6	Discovery is easy to navigate & read, nice spacing throughout the app.
7	Interesting device, very straightforward to use

Tester 2 (Samsung Galaxy A33):

1	Making an account was quick.
2	Log in was also quick.
3	Was easy to add my plant.
4	Scanning was super-fast!
5	I found viewing plants to be straightforward, separate collections could be nice.
6	Searching for plants for easy & the information was handy!
7	I found the device useful & easy to use.

Tester 3 (Google Pixel 6):

1	Normal registration process, Google sign-in would be nice.
2	Login page is grand.
3	Adding plants was straightforward, a notes section could be handy.
4	The scanning is a cool feature, interested to see it with larger datasets.
5	Easy to view plants & update them. The overall colour scheme is nice.
6	Nice layout, would like to see a dense database of plants.
7	Very well-made device, the print quality is great & it functions well.

Tester 4 (Xiaomi 12 Pro):

1	Easy to make my account.
2	Easy to log into my account.
3	Intuitive process for adding plants.
4	Had never seen this technology used, very impressive!
5	Nice design for viewing plants.
6	Really enjoyable being able to look into the individual plants & their needs.
7	Once explained to me, I was easily able to use the device.

Tester 5 (Samsung Galaxy S22 Ultra):

1	Straightforward process.
2	Snappy login.
3	Handy for cataloguing new plants, being able to add notes would be good.
4	Very useful for identifying, being able to use offline would be handy for outdoors.
5	Effective layout for collections.
6	The page is easy to navigate, a bigger number of plants would be nice!
7	Very helpful device, would definitely use in my own home!

Overall, the response from the testers was positive with some very helpful feedback.

2.6. Evaluation

Though I conducted focus group testing for external feedback, the general performance testing of the app was conducted by the developer (myself) to ensure correct functionality throughout the app prior to the focus group testing. Putting myself in the shoes of an "end-user" who has never seen the app before, I tried my best to conduct my own usability testing under the aforementioned testing requirements while attempting to adhere to Nielson's *"10 Usability Heuristics for User Interface Design" (Nielsen, 1994).*

2.6.1. Heuristic Evaluation

The evaluation of Pocket Botanist has been completed by the developer (myself). The Pocket Botanist app was simple to navigation, with a nice colour scheme & clear design. The app's performance felt "snappy" while executing functions & switching between pages. The scanners could possibly be both contained on one page in a single function, but at the moment they are separate for distinction & functional purposes since they are two separate TensorFlow models.

2.6.2. Performance Testing – Normal Load

The performance testing was conducted within Android Studio using JUnit. These tests were conducted in Android Studio using an emulated Google Pixel 5, same as my personal device, 3 times each & the average of the results was taken. The results of these performance tests can be seen below:

- Cold Boot -> Login Page: 1.2 seconds.
- Login Page -> Register Page: 0.325 seconds.
- Perform Registration: 0.562 seconds.
- Login Page -> My Plants Page: 1.203 seconds.
- My Plants Page -> Add Plants Page: 0.314 seconds.
- Select image dialog box to show: 0.103 seconds.
- "Add +" button operation: 0.631 seconds.
- Add Plants Page -> Species Scanner Page: 0.587 seconds.
- Add Plants Page -> Diseases Scanner Page: 0.572 seconds.
- "Classify" button operation: 0.094 seconds.
- Add Plants Page -> Discovery Page: 1.187 seconds.
- Search function: 0.479 seconds.
- Discovery Page -> Plant Info Page: 0.249 seconds.

2.6.3. Performance Testing – Heavy Load

Once I had conducted my control tests to get the baseline measurement for what normal performance levels should be, I conducted the same tests again while the device was under synthetic load, this placed the device's RAM under 85% usage. This excludes the stress placed on the device's RAM by the app itself. These tests were conducted on in Android Studio using an emulated Google Pixel 5, same as my personal device, 3 times each & the average of the results was taken. The results of these performance tests can be seen below:

- Cold Boot -> Login Page: 1.5 seconds. (**+0.3 seconds**)
- Login Page -> Register Page: 0.743 seconds. (**+0.418 seconds**)
- Perform Registration: 2.293 seconds. (**+1.731 seconds**)
- Login Page -> My Plants Page: 3.009 seconds. (**+1.806 seconds**)
- My Plants Page -> Add Plants Page: 0.705 seconds. (**+0.391 seconds**)
- Select image dialog box to show: 0.297 seconds. (**+0.194 seconds**)
- "Add +" button operation: 1.594 seconds. (**+.963 seconds**)
- Add Plants Page -> Species Scanner Page: 1.612 seconds. (**+1.025 seconds**)
- Add Plants Page -> Diseases Scanner Page: 1.605 seconds. (**+1.033 seconds**)
- "Classify" button operation: 0.301 seconds. (**+0.207 seconds**)
- Add Plants Page -> Discovery Page: 3.128 seconds. (**+1.941 seconds**)
- Search function: 1.073 seconds. (**+0.594 seconds**)
- Discovery Page -> Plant Info Page: 0.866 seconds. (**+0.617 seconds**)

Once all of the tests were completed, I investigated the Firebase back-end to ensure all the relevant data was successfully uploaded. When I checked the Firebase console, I was able to confirm that all of the tests had successfully uploaded the relevant information to their correct locations, even with the tests which performed much slower under load. The tests conducted under load were obviously going to be affected but it was interesting to see the operations which used internet connections were more severely affected by this synthetic stress placed upon the phone.

3.0 Conclusions

Pocket Botanist was developed with the purpose of empowering users with the necessary tools for allowing them to become better plant caretakers. There is the helpful accompanying hardware, which I designed to complement the Pocket Botanist's purpose in empowering users & allowing for easy soil monitoring. I believe this project has the ability to encourage users to become plant owners. The Pocket Botanist app also allows for easily information & media management through its tandem usage of Firebase & Glide. As for disadvantages, I believe this project could be taken much further, with interest being shown in my project being taken forward by a Data Science masters student to further bolster the TensorFlow & Plant Database aspects of the project, since these aspects have been kept reasonably small just to act as a proof of concept for the app. Beyond the datasets within the app, I believe further work can be done with the app's design, allowing for more fluid animations & movement between pages. In relation to the Pocket Botanist device,

4.0 Further Development or Research

If given additional time & resources, I would like to improve upon certain aspects of my project in future. The additional time would be beneficial for improving upon both my app's functionality & the datasets which go into the operation of my features.

Beyond the app's further development, I would put the additional time & resources towards further developing the Pocket Botanist app. The cost of 3D printing is pricey but since I intend to further work on this device, I have discussed splitting the cost of a 3D printer with some friends & considered created a product from my device. There has been additional preliminary interest in this product from my college supervisor, Paul Stynes, who advised me on submitting an Invention Disclosure form to help patent my idea. Apart from my supervisor's interest, I have had interest expressed in this product's commercial viability by the owner of a Garden Centre store located in Monkstown, Dublin.

At the time of writing this report, I have the Pocket Botanist app available on my GitHub as a downloadable APK file (Code Envato Tuts+, 2021.). In future when I feel the app is more ready, I would like to deploy the app to the Google Play Store & possible develop an iOS version for Apple.

In terms of project progression into the future, Bord Bia provides *"a host of resources for the small Food, Drink & Horticulture industry" (www.bordbia.ie, 2020)* so there is possibilities of sponsorship in this regard but this requires my app to be established & earning funds as they require a turnover exceeding €100,000 to become a Bord Bia Client.

I have thoroughly loved working on this project so far & I look forward to working on it in future.

5.0 References

[1] Android Developers. (2019). *Android Studio features | Android Developers*. [online] Available at: https://developer.android.com/studio/features [Accessed 10 Dec 2021].

[2] Firebase. (2019.). *Firebase FAQ*. [online] Available at: https://firebase.google.com/support/faq/ [Accessed 10 Dec 2021].

[3] TensorFlow. (2020.). *Why TensorFlow*. [online] Available at: https://www.tensorflow.org/about [Accessed 10 Dec 2021].

[4] GitHub. (2019). *Build software better, together*. [online] Available at: https://github.com/about [Accessed 24 April 2022].

[5] Oscar (2013). *How to use BJT Bipolar Junction Transistor - Beginner's Tutorial*. [online] Oscar Liang. Available at: https://oscarliang.com/bjt-bipolar-junction-transistor-beginner-tutorial/ [Accessed 26 April 2022].

[6] J, A.P. and Gopal, G. (2019). Data for: Identification of Plant Leaf Diseases Using a 9-layer Deep Convolutional Neural Network. *data.mendeley.com*, [online] 1. doi:10.17632/tywbtsjrjv.1 [Accessed 28 April 2022].

[7] TensorFlow. (2020.). *tf_flowers | TensorFlow Datasets*. [online] Available at: https://www.tensorflow.org/datasets/catalog/tf_flowers [Accessed 28 April 2022].

[8] Toor, R. (2021.). *Best 3D Printing Filament for Outdoors? UV Resistance Guide - Filamentive*. [online] https://www.filamentive.com/. Available at: https://www.filamentive.com/best-3d-printing-filament-for-outdoors-uv-resistance-guide/ [Accessed 28 April 2022].

[9] Nielsen, J. (1994). *10 Heuristics for User Interface Design*. [online] Nielsen Norman Group. Available at: https://www.nngroup.com/articles/ten-usability-heuristics/ [Accessed 5 May 2022].

[10] Code Envato Tuts+. (2021.). *How to Generate APK and Signed APK Files in Android Studio*. [online] Available at: https://code.tutsplus.com/tutorials/how-to-generate-apk-and-signed-apk-files-in-android-studio--cms-37927 [Accessed 12 May 2022].

[11] www.bordbia.ie. (n.d.). *Small Business*. [online] Available at: https://www.bordbia.ie/industry/small-business/ [Accessed 13 May 2022].

[12] Android Developers. (2022.). *Build local unit tests*. [online] Available at: https://developer.android.com/training/testing/local-tests [Accessed 13 May 2022].

[13] Android Developers. (2022.). *Fundamentals of Testing*. [online] Available at: https://developer.android.com/training/testing/fundamentals [Accessed 13 May 2022].

[14] Android Developers. (2022.). *Overview of measuring app performance*. [online] Available at: https://developer.android.com/studio/profile/measuring-performance [Accessed 13 May 2022].

6.0 Appendices

6.1. Project Proposal

National College of Ireland

Project Proposal

Pocket Botanist

An Android App

01/11/2021

BSHC4

Software Development

2021/2022

Karl Tallon

X18752469

X18752469@student.ncirl.ie

1. Objectives

The objective of this project is to produce plant care android app which records & keep track of the user's plant collection, accompanied by various features to aid plant care. It would be created using Android Studio, Firebase & Glide as its cornerstone technologies. As of the time of this project, the main features I've chosen so far are:

1. Ability to use your camera (through Image Processing) to recognise your plants & those you do not know the name of, additionally having an option to manually input the plant if you weren't able to find it – I've seen this be an issue on other Plant Care apps, users complaining they cannot input their own plants if not found. I think this would be a useful feature to bolster a database, similar to how MyFitnessPal allows users to manually enter foods & their details if they cannot find them already stored in the database.

2. A feature to catalogue your plant collection & check each plant's relevant information for details such as where to water them, change fertiliser, spray them.

3. Ability to use your camera (again through Image Processing & Deep Learning) to recognise plant diseases & infections then provide relevant help through in-app information or links to relevant online sources.

4. The accompanying device will be used to more easily monitor the plant's soil moisture & empower the user to feel more confident in the task of caring for their plant collection.

5. Possible development of an accompanying device, one that could monitor light conditions or soil conditions to help care for the plant more easily.

2. Background

Going into this, I wanted to create a project which would make use of interesting technology & would have legitimate commercial viability. I came to the decision of a Plant Care app since over the past 2 years of lockdowns, many people have taken up plant ownership as a hobby to fill the time & as a result it has become a very popular trend amongst all age groups since.

For the main development of the app, I will be using Android Studio to create features such as the categorisation & requirement tracking function while using Firebase as the back-end database along with Glide for the media management.

I still have more research when considering what will be best for creating the Plant Disease & Classification functions utilising Image Processing & possibly utilising Machine Learning methods. Currently, I am researching utilising TensorFlow for the task of identifying images through Image Classification. An image classification model can be trained to recognise various classes of images. Subsequently, TensorFlow will provide optimised & pretrained models that I can deploy within my mobile app to recognise the classes I need.

Below is an example screenshots of how the method would work in theory:

3. State of the Art

As for competition in this area of apps - I have seen a few doing similar things but apart from one app, I haven't seen an app which uses all four of the above features I mentioned, particular 1 & 3 being in the same app. The addition of an accompanying proprietary device is something which I have not seen in other apps, which I think adds to the legitimacy of this project as being commercially viable, even as a standalone the device. The only app I have seen which has all of these features has gotten a lot of bad reviews as it has a substantial monthly payment requirement as it also has an on-call botanist feature for users to directly contact, though they do not have a proprietary device. Other than this, they have only a 3-day free trial for users to try it out & even that plan limits the user's ability to use the app's full functionality.

4. Technical Approach

My current approach to development of each main feature was outlined in Section 2.0. However, as the project progresses & actual development begins, I'm sure this will change as I go.

For requirements, once I finish researching how to best develop each feature, I think I will have a much better grasp of what they will require & how to best manage them.

For my project planning, I have broken down the entire project into more manageable sections (aka feature #1, #2, etc) then I've broken these down to more refined targets such as "Registration Development". I think there will be a clear timeline which will develop, something along the lines of:

1- Create initial basic app for implementing features & testing.
2- Begin incorporating features, most likely starting with Login/Registration.
3- Successfully implement all desired features.
4- Conduct appropriate testing to ensure functionality.
5- Conduct focus group testing.
6- Once all are implemented, refine the app's aesthetic.
7- Design & produce prototype device.
8- Deploy app as downloadable APK.

I plan to have this app completed at least 2 weeks prior to final submission to allow for documentation & presentation preparing.

5. Technical Details

In my project, the language I have used is Java within Android Studio for development. As a result, I have several Java classes for each function & their respectively XML files for their corresponding layouts. The overall Design & Architecture I decided upon for this project is a minimalist one, choosing more of a dark-mode-like colour scheme for my universal colours. Technically, the first class the user encounters is the Splash Screen, but this only lasts 1000 milliseconds as it shows the logo. The following screen is the Login Page, from here the user may use the visible functions, such as text boxes & buttons, to navigate the UI to their next chosen function whether it is the My Plants, Add Plants or Discovery

tabs & the sub-pages contained within those activities. For the overall theme of the project, I have selected a series of dark-mode-like hex colours to utilise in my colour scheme, this is in order to present a clean design with green accents to reflect the plant nature of the app.

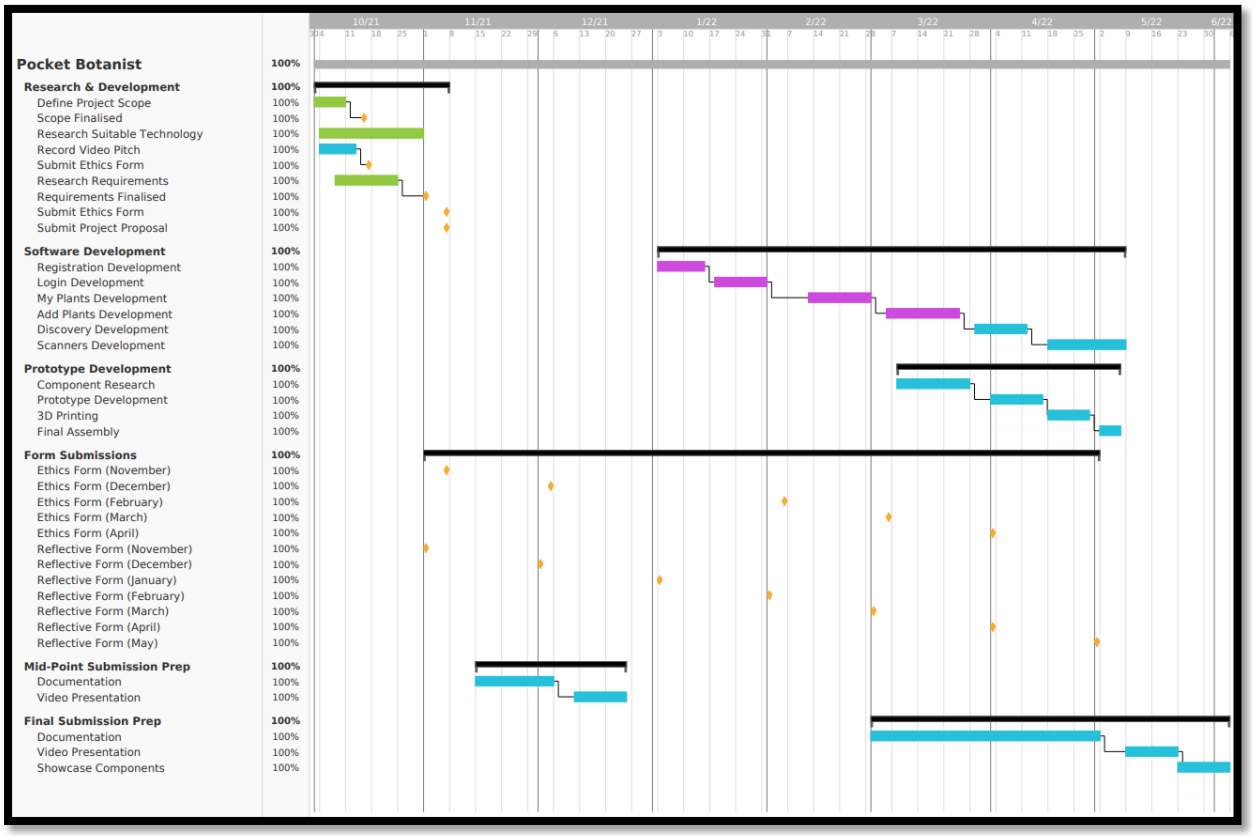
Other technology I am currently researching for implementation is TensorFlow for creating both the Plant Classification & Plant Disease/Infection functions. TensorFlow is an intrinsic aspect of this project.

For the database of the project, I am set on using Firebase with Glide supporting media management. This will be able to store necessary data such as user account details, images, etc.

6. Special Resources Required

As of right now, I do not think I will require any special resources as all the resources I plan to use are open source & free to use. This app was designed using Android Studio & any external code libraries, such as Glide, are implemented there. Using Glide as an example, this is used for its excellent media & information management capabilities. The two datasets which I have researched for my TensorFlow models will be referenced appropriately to give proper credit.

7. Project Plan



8. Testing

As on right now, I have no solid tests planned. However, I have some idea of what the tests could be like. I have been researching the best practices for alpha/beta testing apps using the Google Play Developer console page. Apart from this, there are four main types of testing I am researching for usage currently:

1- Local Unit Testing using JUnits.
2- Instrumented Unit Testing using JUnits.
3- User Interface Tests using Espresso.
4- Focus Groups.
5- Integration Testing.
6- Performance Testing will be conducted to evaluate the overall effectiveness of the app's ability to execute functions.

Throughout this entire project, I have continuously conducted my own testing so that I could confirm the functionality of each function within both my app & the accompanying device. In order to gain extra external perspective, I will seek the help of a focus group comprised of multiple test users to evaluate my app further.

8.1 Performance Testing

The process of Performance Testing allows developers to measure an app's performance under certain conditions, such as RAM & CPU load, to calculate the time it would typically take to execute an action (Android Developers, 2022). After researching into my project more & more, I think this will be the main type of testing which I conduct in my app. For possible uses of Performance Testing in my project, I may use the below method to evaluate my app:

Response Time

- To ensure my app runs smoothly & quickly, I will perform each main action three times within the app to test how long they take to execute, then I will calculate the average time from those results.
- Result times should not exceed 3 seconds, for a phone operating at normal levels with standard internet connection.

8.2 Unit Testing

These Unit Tests will be run locally an emulated device so I may regulate RAM speeds etc *(Android Developers, 2022)*. For possible uses of Unit Testing in my project, I'll present the type of testing I could use JUnit testing for in an example use case:

Use Case 1: Register Account

- Is the user able to establish a database connection?
- Can the user connect to the internet?
    - If the user cannot connect to the internet, the test fails.
- Can data be securely transferred between the app & database?

- The test account is created.
- Prior to sending the data to Firebase, another program will attempt to read this data. If it can successfully read this data, the test fails.
- The data must be securely transferred directly to the Firebase database for the test to be considered a success.
- This data should only be accessible by the Pocket Botanist app & its connected Firebase database.

8.3 Integration Testing

These Integration Tests will allow me to *"verify the way different parts of your app work together" (Android Developers, 2022)*. These types of tests are typically slower than their Unit Test counterparts, but they can be useful when I need to only test a small number of things. I am still not sure if this will be the best fit for my type of project. For possible uses of Integration Testing in my project, I could use it in an example use case like the following:

Use Case 1: Developer adding plants to database

- Integration between the developer (myself) & the Firebase data.
  - The developer inserts the necessary data for a test plant entry.
  - If the app correctly displays the new plant entry, the test is a success.
  - If the app cannot display the new plant entry, the test fails.
  - Further testing will be required to determine the cause of the failure.

6.2 Reflective Journals

6.2.1 November 2021

Student Name: **Karl Tallon – x18752469**

Programme: **BSc (Hons) in Computing – Software Development – BSHC4**

Month: **November 2021**

My Achievements

This month has mainly consisted of research regarding what my project will be, how it will work & how I will make it innovative. Since my project right now involves the EU Digital Covid Cert, the app's main purpose has been a bit up in the air with the changing roadmap of the Government. However, it now seems the EU Digital Covid Cert is here to stay for the foreseeable future.

We also gained some valuable insights from Paul Stynes during our first project meeting. He gave each of us some interesting feedback & recommended good resources for researching the articles relevant to our project ideas.

I also needed to produce my Pitch Video this month which required me to determine the general framework of my project.

My Reflection

This initial research & planning I think will prove invaluable to the execution of my project's development. I think it is important to have a clear & well-made plan prior to beginning, this way you can hopefully develop faster & will less hiccups. The final idea still needs to be fleshed out & I need to decide if I will continue with this idea of an ID verification app or will the requirements to satisfy GDPR/Privacy rules be too much & lead me to pursue a different idea.

My Next Actions

I think at the early stage of the project, the same challenges remain as listed in my project pitch presentation since I'm still in the research phase. That being said, I do want to finalise my idea/plan & proceed with my development as to eliminate procrastination as much as possible.

Supervisor Meetings

This month we have our first meeting with our supervisor Paul, he gave us some great feedback on our project ideas & what our project requirements may be. He critiqued our project ideas & encouraged us to take a more critical look at our work.

6.2.2 December 2021

Student Name: **Karl Tallon – x18752469**

Programme: **BSc (Hons) in Computing – Software Development – BSHC4**

Month: **December 2021**

My Achievements

This month has mainly consisted of research regarding what my project will be, how it will work & how I will make it innovative. My project idea has changed from the EU Covid Cert verification app to now being a Plant Care App which will include multiple features & possibly a cheap physical sensor which can be paired with the app itself.

We also gained some valuable insights from Paul Stynes during our first project meeting. He gave each of us some interesting feedback & recommended good resources for researching the articles relevant to our project ideas.

My Reflection

This initial research & planning I think will prove invaluable to the execution of my project's development. I think it is important to have a clear & well-made plan prior to beginning, this way you can hopefully develop faster & will less hiccups. The final idea still needs to be fleshed out & I am fairly confident I will continue with this idea of Plant Care app.

My Next Actions

I think at the early stage of the project, the same challenges remain as listed in my project pitch presentation since I'm still in the research phase. That being said, I do want to finalise my idea/plan & proceed with my development as to eliminate procrastination as much as possible.

Supervisor Meetings

This month I was able to meet with my supervisor Paul & he gain us some insight into where to go for valuable resources & articles.

6.2.3 January 2022

Student Name: **Karl Tallon – x18752469**

Programme: **BSc (Hons) in Computing – Software Development – BSHC4**

Month: **January 2022**

My Achievements

This month has mainly consisted of research regarding what my project will be, how it will work & how I will make it innovative. My project idea has changed from the EU Covid Cert verification app to now being a Plant Care App which will include multiple features & possibly a cheap physical sensor which can be paired with the app itself.

The high-level structure of the application has been laid out in my technical proposal & may be expanded upon in future.

My Reflection

This initial research & planning I think will prove invaluable to the execution of my project's development. I think it is important to have a clear & well-made plan prior to beginning, this way you can hopefully develop faster & will less hiccups. The final idea has been fleshed out & I am confident I will continue with this idea of Plant Care app.

I have completed the Registration & Login components of the application & have successfully linked them to the Firebase Database.

The biggest challenge currently is completing the TensorFlow components of the application. I had almost completed the proof of concept for the Plant Disease Recognition function but there is a persisting error which I have not solved yet. At the beginning of the 2nd semester, I will be ironing out the process.

My Next Actions

I think at the early stage of the project, the same challenges remain as listed in my project pitch presentation since I'm still in the research phase. That being said, I do want to finalise my idea/plan & proceed with my development as to eliminate procrastination as much as possible.

The TensorFlow issues will require further research & testing to resolve but I am confident this will be possible to accomplish.

Supervisor Meetings

My supervisor Paul & I unfortunately had no meetings this month due to scheduling conflicts between him & I.

6.2.4 February 2022

Student Name: **Karl Tallon – x18752469**

Programme: **BSc (Hons) in Computing – Software Development – BSHC4**

Month: **February 2022**

My Achievements

We also gained some valuable insights from Paul Stynes during our first project meeting. He gave each of us some interesting feedback & recommended good resources for researching the articles relevant to our project ideas.

Going into February, I will continue to work on resolving my TensorFlow issues while I strive to complete the functionality of my project. I have also learnt of a more aesthetically pleasing way to create a login/register/splash screen so I may incorporate that instead of my original design.

My Reflection

This initial research & planning I think will prove invaluable to the execution of my project's development. I think it is important to have a clear & well-made plan prior to beginning, this way you can hopefully develop faster & will less hiccups. The final idea has been fleshed out & I am confident I will continue with this idea of Plant Care app.

The biggest challenge currently is completing the TensorFlow components of the application. I had almost completed the proof of concept for the Plant Disease Recognition function but there is a persisting error which I have not solved yet. At the beginning of the 2nd semester, I will be ironing out the process.

My Next Actions

I think at the early stage of the project, the same challenges remain as listed in my project pitch presentation since I'm still in the research phase. That being said, I do want to finalise my idea/plan & proceed with my development as to eliminate procrastination as much as possible.

The TensorFlow issues will require further research & testing to resolve but I am confident this will be possible to accomplish. I am confident that, with time, I will be able to complete these important features.

Supervisor Meetings

My supervisor Paul & I unfortunately had no meetings this month due to scheduling conflicts between him & I.

6.2.5 March 2022

Student Name: **Karl Tallon – x18752469**

Programme: **BSc (Hons) in Computing – Software Development – BSHC4**

Month: **March 2022**

My Achievements

Progress has been close to zero during this month as our other modules in college are very demanding.

My Reflection

I have not been able to accomplish much in relation to my Software Project as

The biggest challenge still facing me is completing the TensorFlow components of the application.

My Next Actions

I am hoping to complete my other module assignments soon & have the ability to completely dedicate all of my time to the project.

Supervisor Meetings

My supervisor Paul & I unfortunately had no meetings this month due to scheduling conflicts between him & I.

6.2.6 April 2022

Student Name: **Karl Tallon – x18752469**

Programme: **BSc (Hons) in Computing – Software Development – BSHC4**

Month: **April 2022**

My Achievements

This month consisted mainly of software development related activities as I continued working on Pocket Botanist's functionality. Also, once I realised I had enough time remaining, I decided to produce some hardware which will accompany the app. This device will allow users to monitor soil conditions more easily.

My Reflection

I have completed the majority of functionality in my app, with the rest of the work being left to finishing touches & the device prototyping.

My Next Actions

I will continue to work on the final touches for the app & plan to begin prototyping my device.

Supervisor Meetings

My supervisor Paul & I unfortunately had no meetings this month due to scheduling conflicts between him & I.

6.2.7 May 2022

Student Name: **Karl Tallon – x18752469**

Programme: **BSc (Hons) in Computing – Software Development – BSHC4**

Month: **May 2021**

My Achievements

In this final month, I was able to watch the culmination of my work come together as my app & device both came to completion. I am proud to say, these two aspects which make up my project are completed & ready for submission. It has been a difficult month, working hard to produce something I am proud of. Now it is finished & all ready for submission, I am pleased. I have also met with my supervisor, Paul Stynes, to receive some feedback & I was very happy with his opinion of my project.

My Reflection

In hindsight, I would've started work even earlier. If I did, I believe I could complete much more like bolstering the database of plant entries & creating more complex TensorFlow models for image classification.

My Next Actions

Once I submit my project, I plan to continue working on the project as I proceed in life. I truly enjoyed working on this project & I am excited to work on it in future.

Supervisor Meetings

This month I met with Paul to discuss my project & get his insight on what he thought about my almost finished project. Thankfully he was happy with my project. Paul also gave me very insight feedback regarding my Showcase Poster.