The Hashemite University, Zarqa, Jordan
Faculty of Prince Al-Hussein Bin Abdallah II for Information Technology
Computer Science and Applications Department

"Mashtaly"

A project submitted
in partial fulfillment of the requirements for the
B.Sc. Degree in Computer Science and Applications

By:

Saeed Saed Saeed Ibrahim	(2031508)
Zaid Saeed Mohammad Alarbid	(2035013)
Omar Adeeb Mohammad Ghanayem	(2044084)
Mohammad Ala' Mohammad Bal'awi	(2034107)

Supervised by:

Prof. Mohammad Bsoul

Committee Member Names:

Dr. Sari Awwad

Dr. Nabhan Hamadneh

December 2023

CERTIFICATE

It is hereby certified that the project titled ***Mashtaly***, submitted by undersigned, in partial fulfillment of the award of the degree of "Bachelor in Computer Science and Applications" embodies original work done by them under my supervision.

All the analysis, design and system development have been accomplished by the undersigned. Moreover, this project has not been submitted to any other college or university.

Saeed Saed Saeed Ibrahim	*Saeed Ibrahim*
Mohammad Ala' Mohammad Bal'awi	*Mohammad Bal'awi*
Zaid Saeed Mohammad Alarbid	*Zaid Alarbid*
Omar Adeeb Mohammad Ghanayem	*Omar Ghanayem*

ABSTRACT

Mashtaly is an innovative application that utilizes Internet of Things (IoT) technology to address the challenges of plant care and water conservation. This project aims to provide a solution to the growing global water crisis by helping users conserve water and promote sustainability. The **Mashtaly** application allows users to remotely monitor the soil moisture of their plants and receive real-time data and insights on their water needs.

By utilizing IoT sensors for monitoring humidity levels and a camera API to identify plants and provide information on their care requirements, **Mashtaly** offers a comprehensive and reliable solution for plant care.

This project has numerous benefits, including water conservation, improved plant growth, sustainability, cost savings, and convenience.

However, the application does have certain limitations, such as its reliance on an internet connection and its primary focus on indoor plant care. The expected outcomes of this project are numerous and varied, including a reduction in water shortages, plant preservation, increased shelf life and health, and user satisfaction.

The **Mashtaly** application's functional and non-functional user requirements are designed to ensure usability, security, accuracy, reliability, and availability, providing users with a high-quality experience and contributing to the overall success of the project.

TABLE OF CONTENTS

ABBREVIATIONS

- **IoT:** Internet of Things
- **API:** Application Programming Interface
- **UML:** Unified Modeling Language
- **IDE:** Integrated Development Environment
- **UI:** User Interface
- **UX:** User Experience
- **MVVM:** Model View View Model
- **GUI:** Graphical User Interface

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Welcome to our innovative application that harnesses the power of the Internet of Things (IoT) to help conserve water and nurture the plants at the same time. **Mashtaly** is designed to create a more sustainable future by providing a unique solution to the challenges of water conservation and plant care.

1.1 Overview

Mashtaly allows you to remotely monitor the soil moisture of plants, ensuring that they receive just the right amount of water needed to thrive. **Mashtaly** connects with various sensors and devices, such as soil moisture sensors, and weather forecasting, to provide real-time data and insights on your plants water needs.

With **Mashtaly**, you can receive notifications when the plants need attention, whether you're at home or away.

Mashtaly gives the peace of mind, knowing that your plants are being cared for and water is being conserved.

Mashtaly is perfect for homeowners and businesses that want to reduce water usage and promote sustainability. By using **Mashtaly**, you can contribute to the global effort to conserve water and protect the planet's resources.

1.2 Project Motivation

In today's world, where water conservation and sustainability are critical issues, developing a plant care application has become increasingly important. The **Mashtaly** application can provide numerous benefits, including:

- **Water conservation:** As the world faces a water crisis, it is essential to preserve this precious resource. The plant care application can help users save water by giving certain amounts based on the suggestions of the **Mashtaly** application or by the user to irrigate the plant.

- **Improved Plant Growth:** Plants need the right amount of water to grow and thrive. By using the **Mashtaly**, users can ensure that their plants receive the optimal amount of water, leading to improved growth and overall health.
- **Sustainability:** Conserving water and taking care of plants is an essential part of sustainability. By using the **Mashtaly**, users can contribute to a more sustainable future.
- **Cost Savings:** As water becomes more expensive, users can save money on their water bills by conserving it. Additionally, healthy plants can be more productive and aesthetically pleasing, adding value to homes.
- **Convenience:** With an IoT based system, users can monitor their plants remotely, making it more convenient for those who are busy or away from home. This feature is particularly useful for individuals who travel frequently or have a busy schedule.

In addition to the soil moisture sensors and weather forecasting, our plant care application also utilizes innovative technology to monitor plant. By using an application programming interface (API) with a camera, **Mashtaly** application can detect and identify various types of plants and provide information on their care requirements. This feature allows users to easily find out what type of plant they have and how they can give the best care for it, without the need for extensive research or knowledge.

The implementation of IoT sensors for monitoring humidity levels combined with the use of a camera API can lead to a more precise and reliable means of plant care. This approach can help to ensure that plants receive optimal conditions for growth while minimizing the risk of stress and disease.

Overall, our plant care application's innovative approach has the potential to revolutionize the way plant care is managed and bring about significant benefits for plant enthusiasts and the environment. With the use of IoT technology and a camera API, **Mashtaly** can provide a comprehensive and reliable solution for plant care and contribute to the global effort to conserve water and protect the planet's

1.3 Problem Statement

Mashtaly application is designed to address various problems related to plant care, including issues resulting from over or under-watering. These problems are prevalent worldwide and can have severe consequences for plant health. The application's features can help prevent these issues and ensure that plants receive the appropriate amount of water needed for their growth and survival.

Additionally, **Mashtaly** can benefit individuals who may not have the time to care for their plants regularly. With its user-friendly interface and automated features, the **Mashtaly** can assist in monitoring and caring for plants, reducing the burden on busy users.

In summary, **Mashtaly** is a powerful solution for many plant care-related issues, such as over or under-watering and it provides convenient features for users who may not have the time to tend to their plants regularly.

1.4 Project Aim

Revolutionizing Plant Breeding with an Innovative Application for Water Management

1.5 Project Objectives

- Develop a cutting-edge application for plant breeders to enhance plant care.
- Implement soil moisture testing to provide timely watering notifications.
- Integrate a camera API for plant identification and detailed care information.
- Tackle water scarcity challenges by promoting efficient and informed watering practices.

1.6 Project Limitations

The **Mashtaly** application has certain limitations that users should be aware of. Firstly, it is primarily designed for indoor plant care, and may not provide optimal recommendations for outdoor plants or those in specific environments. Secondly, the application relies on an internet connection to function, which may limit its accessibility in areas with poor connectivity. In addition, the accuracy of the data and

recommendations provided by the app may be impacted by factors such as lighting conditions and image quality. Users should keep these limitations in mind while using the application and supplement its recommendations with their own knowledge and expertise where necessary. Despite these limitations, **Mashtaly** remains a valuable tool for plant care enthusiasts, offering a range of features and resources to promote healthy plant growth and maintenance.

1.7 Project Expected Output

The successful completion of this project is expected to yield several positive outcomes, including a reduction in water shortages and the preservation of plant life. By providing users with the necessary information to water their plants optimally, the application can help reduce water waste and promote sustainable practices.

Moreover, the application's features can lead to an increase in plant shelf life and health, adding value to homes and creating aesthetically pleasing environments. The app's user-friendly interface and automated features are also expected to result in a high level of user satisfaction and happiness.

In summary, the expected outcomes of this project are numerous and varied, including water conservation, plant preservation, increased shelf life and health, and user satisfaction. The **Mashtaly** application has the potential to revolutionize plant care practices and bring about significant benefits for both users and the environment.

1.8 Project Schedule

Chapter	Start date	Finish date
Introduction	17/3/2023	22/3/2023
Literature Review	23/3/2023	26/3/2023
Requirement Engineering and Analysis	1/4/2023	15/4/2023
Architecture and Design	28/4/2023	9/5/2023
Implementation Plan	12/12/2023	17/12/2023
Testing Plan	17/12/2023	25/12/2023
Conclusion and Results	25/12/2023	30/12/2023

Table 1.1: Project Schedule

In Table 1.1, the text delineates the temporal facets of the project, presenting a comprehensive schedule that clarifies the commencement and completion dates of each task.

1.9 Report Organization

The subsequent sections of the report are structured as follows. In **Chapter 2**, a comprehensive Literature Review is introduced. Following that, **Chapter 3** outlines the Software Requirements, while **Chapter 4** delves into the presentation of Software Design. The intricacies of design and architecture components are expounded upon in **Chapter 5**. Moving forward, **Chapter 6** entails the testing of the **Mashtaly** application as outlined in the Testing Plan. **Chapter 7** then elucidates on future work and serves as the concluding section of the report.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

In this chapter, we will discuss various applications and projects that share similarities with our **Mashtaly** application. We will explore the unique features that differentiate our project from its counterparts. Furthermore, we will examine the challenges faced by other projects and outline our proposed solutions to these issues.

2.2 Existing Systems

- **Planta - Care for your plants**[1] this is a mobile application available on the Google Play store that aims to help users take better care of their plants. **Planta - Care for your plants** is designed to cater to both novice and experienced plant owners and offers a range of features to help users monitor and care for their plants. With its comprehensive plant care database, watering schedule, and plant identification tool, Planta provides users with the information and tools they need to ensure their plants thrive.

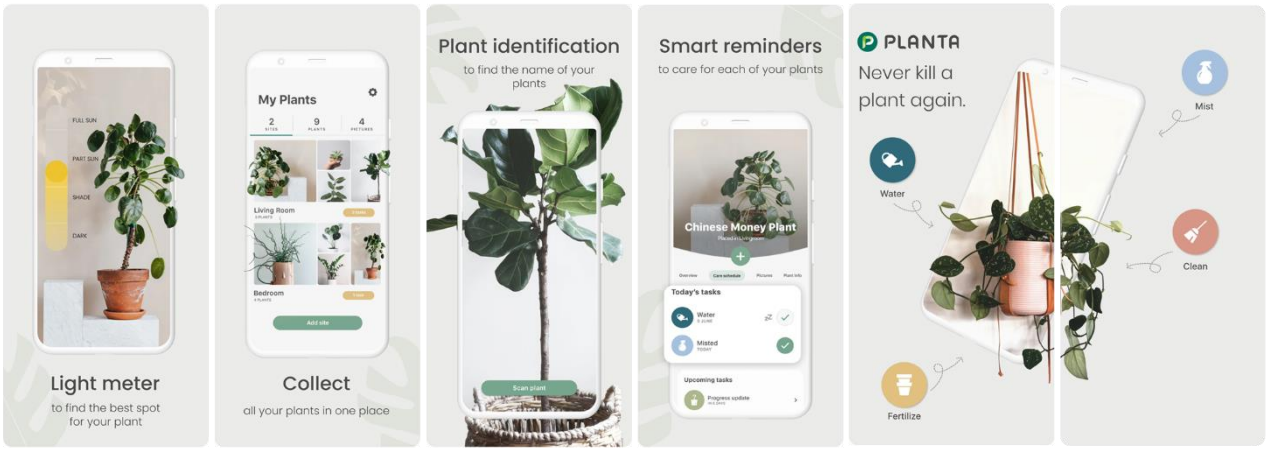


Figure 2.1 :Overview of Planta - Care for your plants

In Table 2.1 above, the following are shown the holistic user interface of the **Planta - Care for your plants** application, highlighting its functionalities that encompass various actions.

- **Plant Parents**[2] this is a mobile application designed to help plant enthusiasts take care of their plants. **Plant Parents** features various tools and resources, such as real-time notifications and a plant library, to monitor and care for plants, making it an essential tool for users who want to ensure their plants' healthy growth and longevity.

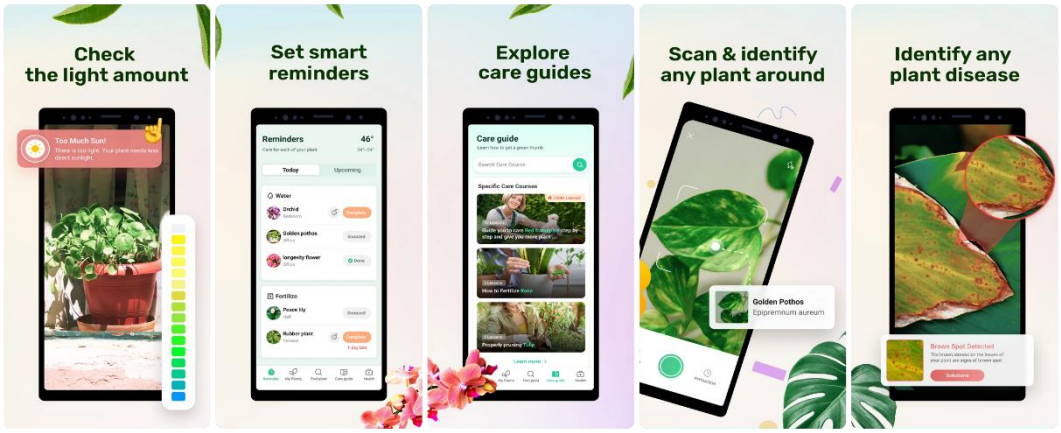


Figure 2.2 :Overview of Plant Parents

In Figure 2.2, check out the user interface of the **Plant Parents** app, spotlighting its functionalities that cover a bunch of actions.

- **Blossom**[3] this is a mobile application available on Google Play store that aims to help gardeners and plant enthusiasts take care of their plants. **Blossom** offers a range of features to help users monitor and care for their plants, such as real-time data on plants' water needs through sensors and a watering schedule. Blossom also provides useful resources and tips on plant care and identification. Additionally, **Blossom** includes tools for managing a garden, such as a garden planner and harvest tracker. Overall, Blossom is a useful and comprehensive application for both novice and experienced gardeners.

Figure 2.3 :Overview of Blossom

In Figure 2.3 depicted, behold the panoramic user interface of the Blossom application, showcasing its myriad functionalities that seamlessly envelop a multitude of actions.

2.3 Overall Problems of Existing Systems

Features / Applications	Planta	Plant Parents	Blossom	Mashtaly
Limited plant database	Yes	Yes	Yes	No
Completely free	No	No	No	Yes
Accuracy	No	No	Yes	Yes
User-friendly User interface	No	No	No	Yes

Table 2.1: Overall Problems of Existing Systems

In Table 2.1 above, the following are shown a brief comparison of features for different applications related to plants, namely "Planta" "Plant Parents" "Blossom" and "**Mashtaly**" Each application is evaluated based on four features.

- **Limited plant database:** A limited plant database is a collection of information and data about a specific set of plants. This database typically includes details such as the plant's scientific name, common name, description, growth habits, cultivation requirements, and other relevant information. The purpose of a limited plant database is to provide a resource for people who want to learn more about a particular group of plants, such as those commonly found in a specific region or those with similar characteristics. This type of database can be useful for gardeners, horticulturists, researchers, and anyone interested in learning more about plants.

- **Completely free:** Refers to an application that is available for download and use without any cost. This means that users can access all of the app's features and content without having to pay any fees or make in-app purchases.

- **Accuracy:** This is an essential aspect of plant care applications as it plays a crucial role in enabling users to effectively care for their plants, regardless of their level of expertise. With precise and reliable information and guidance, users can ensure that their plants thrive and remain healthy over an extended period. By providing accurate data on factors such as watering schedules, plant care applications empower users to make informed decisions and take appropriate action to promote optimal plant growth and health.

- **User interface:** A well-designed UI is crucial to the success of a mobile application. It can greatly impact user engagement, satisfaction, and retention, making it an essential aspect of the application development process.

2.4 Overall Solution Approach

- **Limited plant database:** The **Mashtaly** application boasts an extensive database, owing to its integration with multiple sensors, weather forecasting systems, and various APIs.

- **Completely free:** The **Mashtaly** application is a non-monetized application, providing all users with equal access to its complete range of features, without any in-app purchases.

- **Accuracy:** The data provided in the **Mashtaly** application is meticulously curated to ensure high accuracy levels, sourced from dependable websites and databases such as Pl@ntNet[4] to help users identify plants by utilizing image recognition technology.

- **User-friendly User interface:** The user interface of the **Mashtaly** application is designed to be user-friendly and intuitive, providing an effortless and comprehensible experience to the user.

CHAPTER 3: REQUIREMENT ANALYSIS

This chapter aims to provide a comprehensive list of stakeholders, including individuals and business owners, who may be impacted by the decision, activity, or outcome of this project. Each stakeholder will be categorized based on their involvement and level of influence, more over the chapter will also present a use case diagram that depicts the relationship between the user and various use cases, highlighting both functional and non-functional requirements.

3.1 Stakeholders

- **Primary**
  - **Plant Breeder:** The plant breeders are individuals who will be using the application to take care of their plants. As the primary stakeholders, **Mashtaly** aims to provide a comprehensive platform where plant breeders can add plants and post articles and even sale or display their regular and rare plants. This allows for a more interactive and engaging user experience.

  - **Admin:** The admin is a person who is responsible for managing and organizing various aspects of the application. This includes tasks such as managing user accounts, approving user posts, and adding new plants to the application's database. The admin plays a critical role in ensuring that the application runs smoothly and efficiently.

- **Secondary**
  - **API's:** Application Programming Interfaces, act as a software intermediary that enables communication between different applications. They are commonly used in web and mobile app development to share and exchange data. **Mashtaly** employs two APIs: the weatherapi[5], which identifies the user's location and retrieves weather data, and the Pl@ntNet, which uses a camera to capture images of plants and identify them. These APIs are vital components of the application, providing accurate and relevant information to users and enhancing their overall experience. By utilizing these APIs, **Mashtaly** can offer a comprehensive and effective plant care application that meets the needs of its users.

- **Sensor:** A soil moisture sensor is a device used to measure the amount of water present in the soil. In **Mashtaly**, a soil moisture sensor is utilized to collect this data and send it to the app via Wi-Fi connection. This information is then used to notify users when their plants require watering and to help conserve water by suggesting the appropriate amount needed, as shown in Figure 3.1 ESP8266 is a low-cost, Wi-Fi-enabled microcontroller chip designed for IoT applications. It was developed by the Chinese company Espressif Systems and was first released in 2014. The chip is widely used in a variety of projects, from simple home automation systems to more complex applications. The most important of these ESP8266 is the presence of a soil moisture sensor, which is an electronic device that measures the moisture content in soil. It is commonly used in agriculture, horticulture, and gardening to ensure plants receive the appropriate amount of water to grow and thrive.

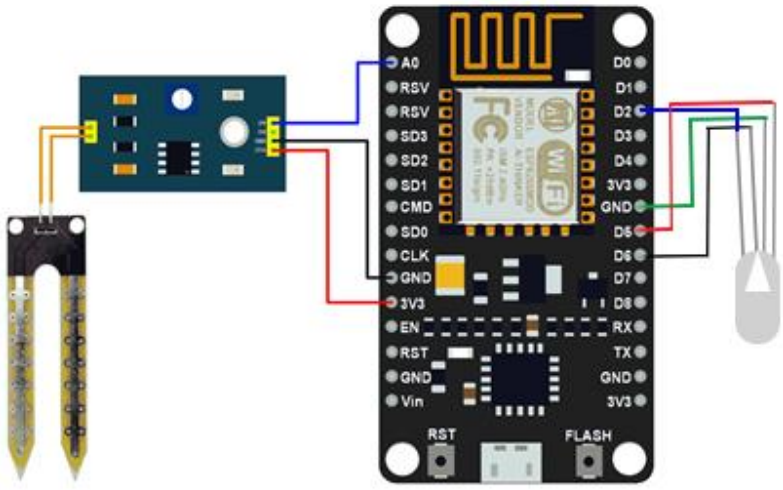


Figure 3.1 :ESP8266, Soil Moisture Sensor and RGB LED

- **Database:** A database is a structured collection of data that is organized in a way that enables efficient storage, retrieval, and management of information. **Mashtaly's** database plays a crucial role in storing and managing data related to users, APIs, and plant information. It serves as the backbone of the application and helps ensure that **Mashtaly** can provide accurate and relevant information to its users. The proper management of this database is essential to the success of the **Mashtaly**.

3.2 UML Use Case Diagram

This section contains the use case diagram and all that it contains of functional user requirements and non-functional user requirements, and as shown in Figure 3.2.

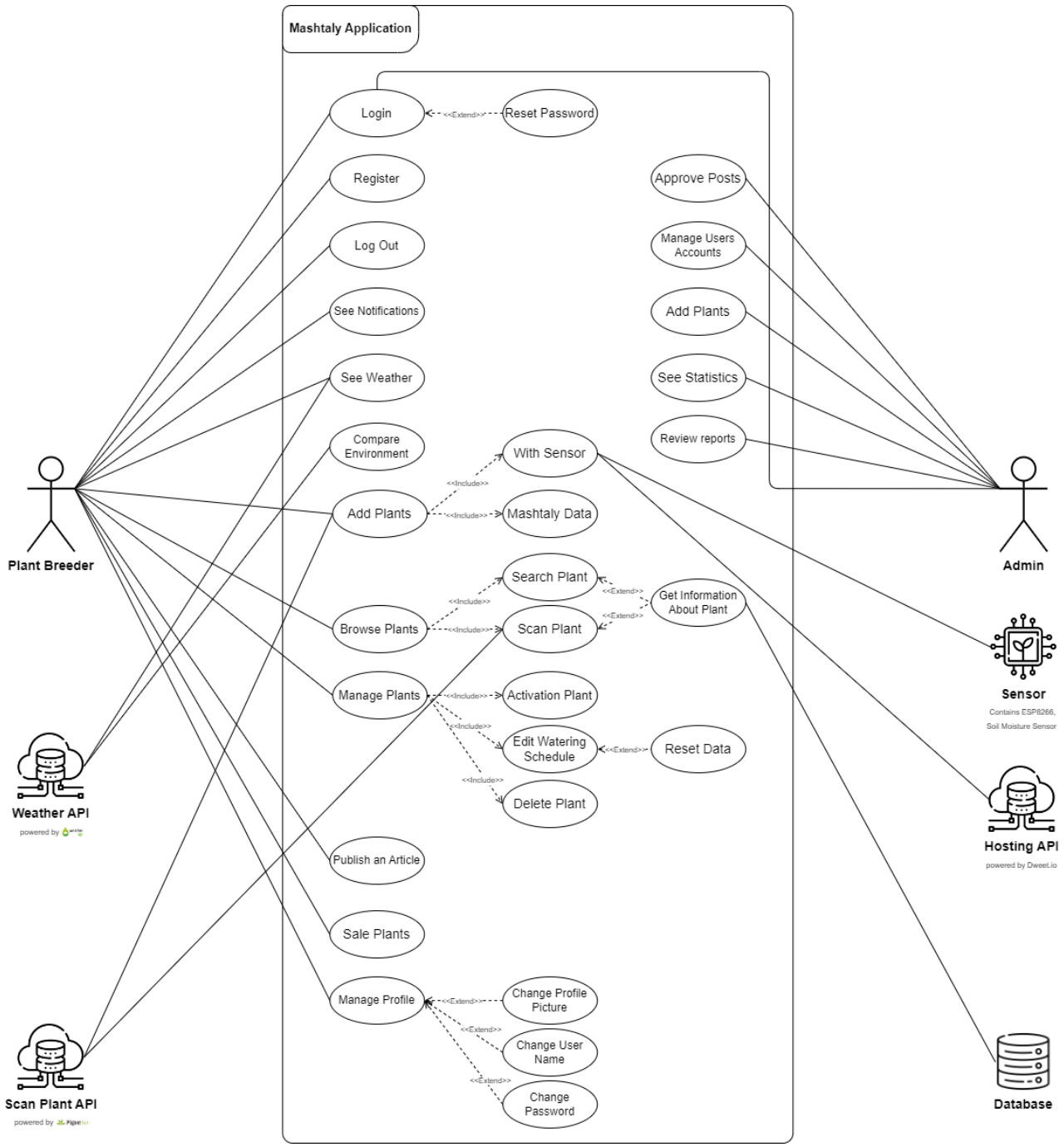


Figure 3.2: Use Case Diagram

3.3 Functional User Requirements

- **Login:** The login screen is the gateway for registered plant breeders to access the full range of features available within the **Mashtaly** application. Upon entering their unique email and password, the system will validate their credentials to ensure they are authentic. If the login details are verified successfully, the user will be granted access to their account and all associated functionalities.

- **Register:** The registration process involves a new plant breeder entering their email and password and clicking on the "Register" button. The system will then validate the entered data to ensure there are no pre-existing user accounts with the same email and password combination. Upon successful validation, the user's registration will be processed, and an email verification will be sent to the registered email address. This email verification is essential to confirm the user's identity and authenticate their account registration.

- **See Notifications:** Plant breeders have the ability to view notifications from the database regarding necessary watering for their plants. These notifications serve as timely alerts to ensure optimal plant growth.

- **See Weather:** Plant breeders have access to real-time weather conditions provided by the weatherapi. This feature enables them to monitor and track weather patterns that may impact their plants and adjust maintenance schedules accordingly.

- **Compare Environment:** Plant breeders can compare the ideal weather conditions for their plants with the current weather conditions. This comparison allows them to evaluate whether the current weather is suitable or unsuitable for optimal plant growth. This information is vital for making informed decisions about plant maintenance and care.

- **Add Plants:** Plant breeders can add plants and record data readings through either the **Mashtaly** database or the sensor system. This feature provides

flexibility in data collection, allowing for a more comprehensive and accurate analysis of plant growth and health.

- **Browse Plants:** Plant breeders can to access information about plants by searching for them by name or by scanning the plant using the camera on the **Mashtaly** application. When a plant is scanned, the **Mashtaly** application sends the data to the API for analysis, which then provides the name of the plant to the user. This allows plant breeders to identify specific plant varieties quickly and efficiently, saving time and effort in the identification process.

- **Manage Plants:** The plant breeder can customize the plant's data according to their requirements and reset it if necessary. Additionally, they can activate or deactivate the plant as needed. If the plant breeder no longer needs the plant, they can also delete it. This feature gives plant breeders flexibility and control over their plants' data, allowing them to make adjustments and changes as needed. It ensures that the data on the platform is accurate and up to date, providing plant breeders with reliable information to make informed decisions.

- **Publish an Article:** Plant breeders can to create and publish articles related to plants. This feature provides plant breeders with a platform to share their knowledge and insights on plants, enabling them to contribute to the wider community of plant enthusiasts. Plant breeders can create articles on various topics related to plants, including cultivation, breeding, and maintenance. They can add images, videos, and other media to their articles to enhance the user experience and make the content more engaging.

- **Sale Plants:** Plant breeders can to sale their rare or ordinary plants through the platform. This feature provides plant breeders with a marketplace to showcase and sale their plants, enabling them to reach a wider audience of potential buyers. Buyers can browse the plants and contact the plant breeder to purchase the plants they are interested in. This feature promotes commerce and facilitates the exchange of plants among the plant breeding community.

- **Manage Profile:** Plant breeders can update their profile picture and modify their password as needed. Additionally, plant breeders can change user name.

- **Approve Posts:** The administrator can to review the content of a post and determine whether to approve or reject it. The administrator has the ability to view the post's contents to ensure that it meets the established criteria and standards.

- **Manage Users Accounts:** The administrator can modify user account settings and resolve any issues that may arise with the user's account. For example, if a user is found to be violating platform guidelines, the administrator may disable their account to prevent further violations. Additionally, if a user forgets their password or experiences issues accessing their account, the administrator can change the user's password and help resolve the issue. This feature ensures that users have access to reliable support when encountering account-related problems and helps maintain the integrity of the platform.

- **Add Plants:** The administrator can efficiently manage and organize the database by adding new plants and updating existing ones, ensuring that the platform's plant-related data is always accurate and up to date, so can plant breeders access accurate information on a broader range of plants.

- **See Statistics:** The administrator can to access data on the platform's sensor count, user count, and plant count. This data is crucial for monitoring and managing the platform's performance and ensuring that it can effectively meet user needs.

- **Review reports:** The administrator can review the modified post and take appropriate action, which may involve either removing the post or retaining it based on the evaluation.

3.4 Non-Functional User Requirements

- **Usability: Mashtaly** has been designed with a user-friendly interface, allowing users to navigate and use the application without the need for extensive guidelines or assistance. The aim is to ensure that the app is easy to understand and use, thereby enhancing the user experience and making plant care accessible to all users, regardless of their level of expertise.

- **Security: Mashtaly** prioritizes the security of its user's data by implementing measures to protect against external threats and internal attacks. The **Mashtaly** data is stored securely and is protected from unauthorized access.

- **Availability: Mashtaly** will ensure high availability, with the application accessible to online users at all times.

- **Accuracy**: **Mashtaly** ensures accuracy by utilizing a soil moisture sensor and a reliable API to provide precise information for plant care.

- **Reliability: Mashtaly** aims to provide a reliable and stable service with minimal downtime or disruptions. The application is designed to minimize the risk of data loss, errors, or other issues that could impact the user experience, ensuring that users can access and use the application whenever they need it.

CHAPTER 4: ARCHITECTURE AND DESIGN

In this chapter, we intend to offer a detailed and comprehensive overview of the **Mashtaly** application, which includes its fundamental structure, functionality, and design. By thoroughly examining these critical aspects of the platform, users can gain a better understanding of how it operates and the range of capabilities it offers.

4.1 Software (System) Architecture

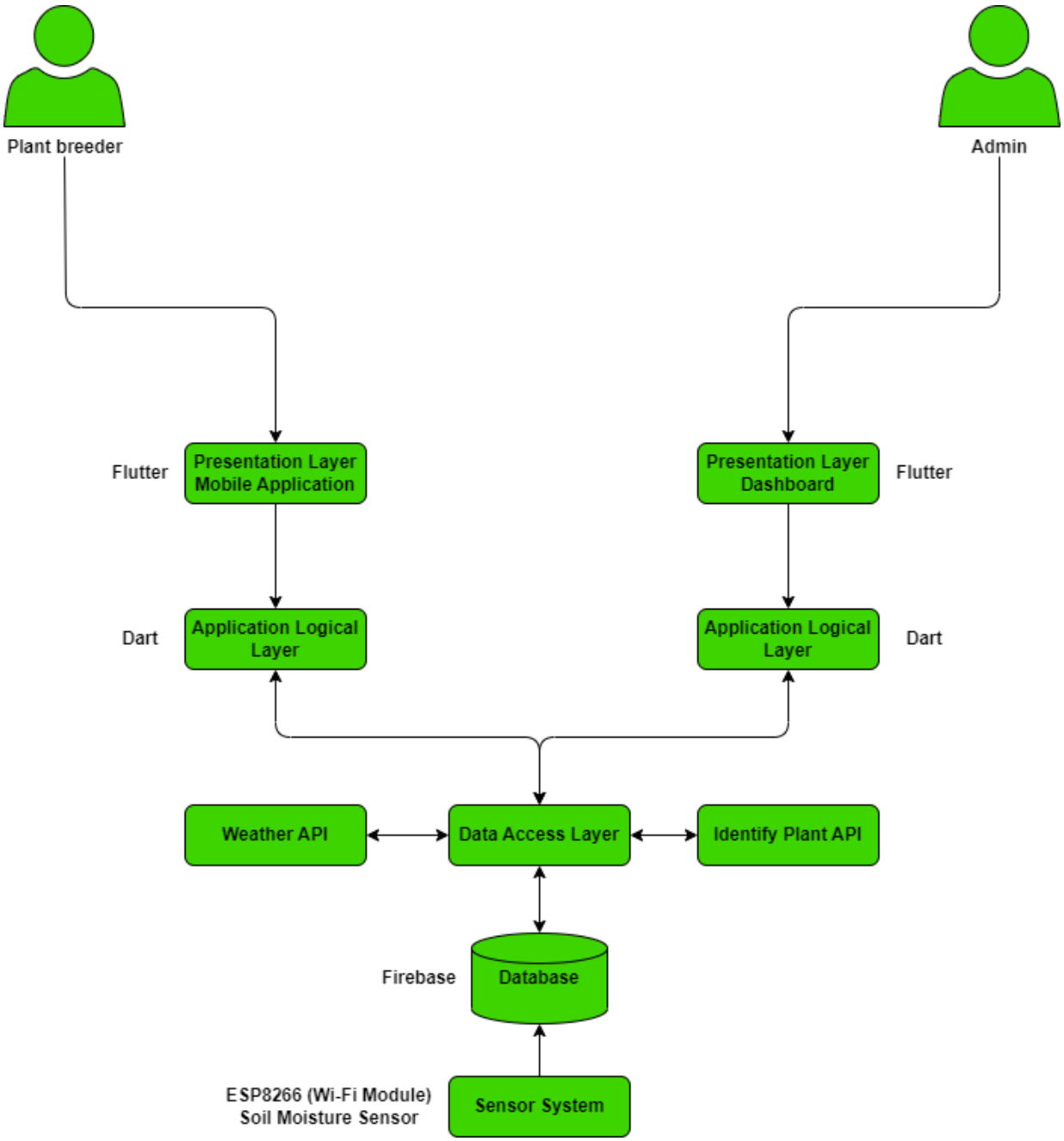


Figure 4.1: Software (System) Architecture Diagram

In Figure 4.1 above, the following are shown:

- **Presentation Layer**
  - **Mobile Application**: This is specifically designed for plant breeders, enabling them to log in, add, nurture plants, publish articles, and sale plants, among other functions outlined in Chapter 3.
  - **Dashboard:** This is tailored to the needs of administrators, providing them with access to statistics related to **Mashtaly's** performance and the ability to approve posts published by plant breeders. Other functions available to admins are also detailed in Chapter 3.

- **Application Logical Layer:** This layer is responsible for handling user requests initiated via user interfaces built using Flutter on the breeder side, and via a Flutter-built tablet on the administrator side.

- **Data Access Layer:** To ensure seamless data exchange between these interfaces, Firebase is utilized to receive and send data in real-time.

4.2 Software Design

4.2.1 UML sequence/communication diagram

Sequence diagrams are all about capturing the order of interactions between objects and classes in a system and can describe which interactions will be triggered when a particular use case is executed and in what order those interactions will occur, and as shown in Figures 4.2– 4.8.

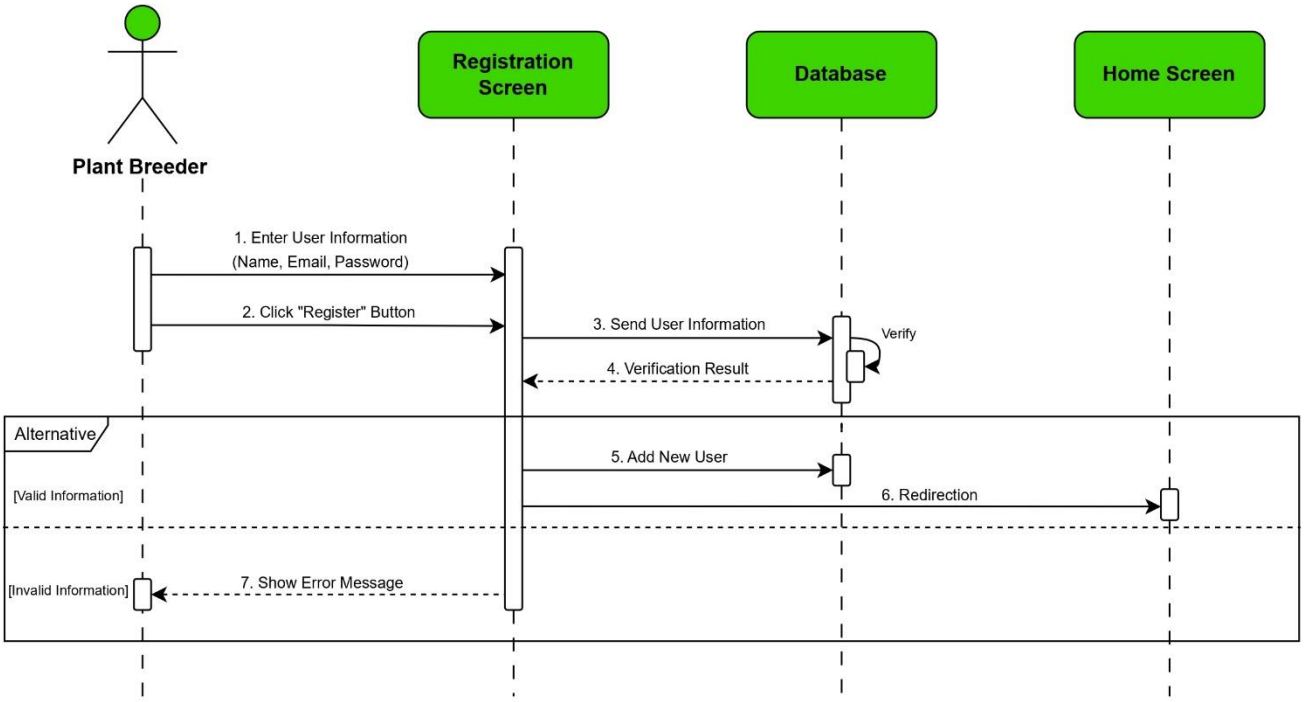


Figure 4.2 :Sequence Diagram – Registration Process

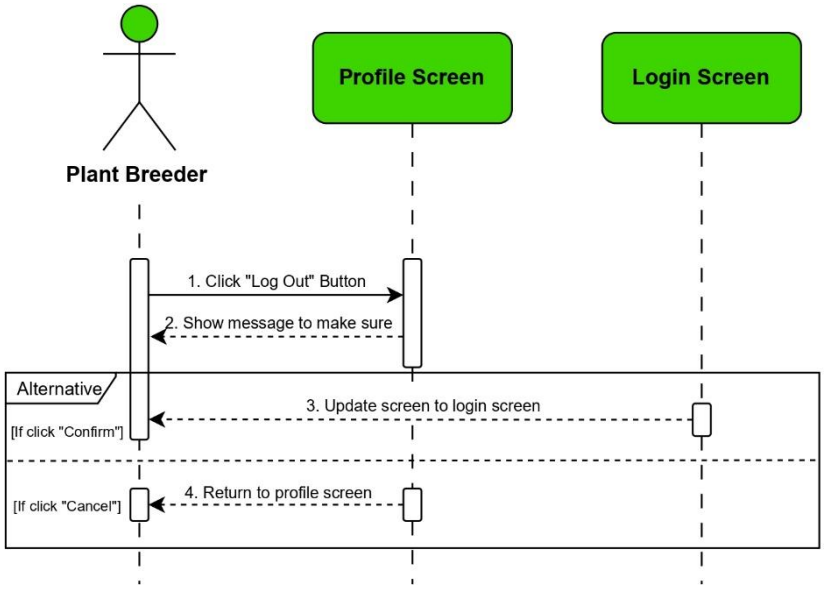


Figure 4.3 :Sequence Diagram – Log Out Process

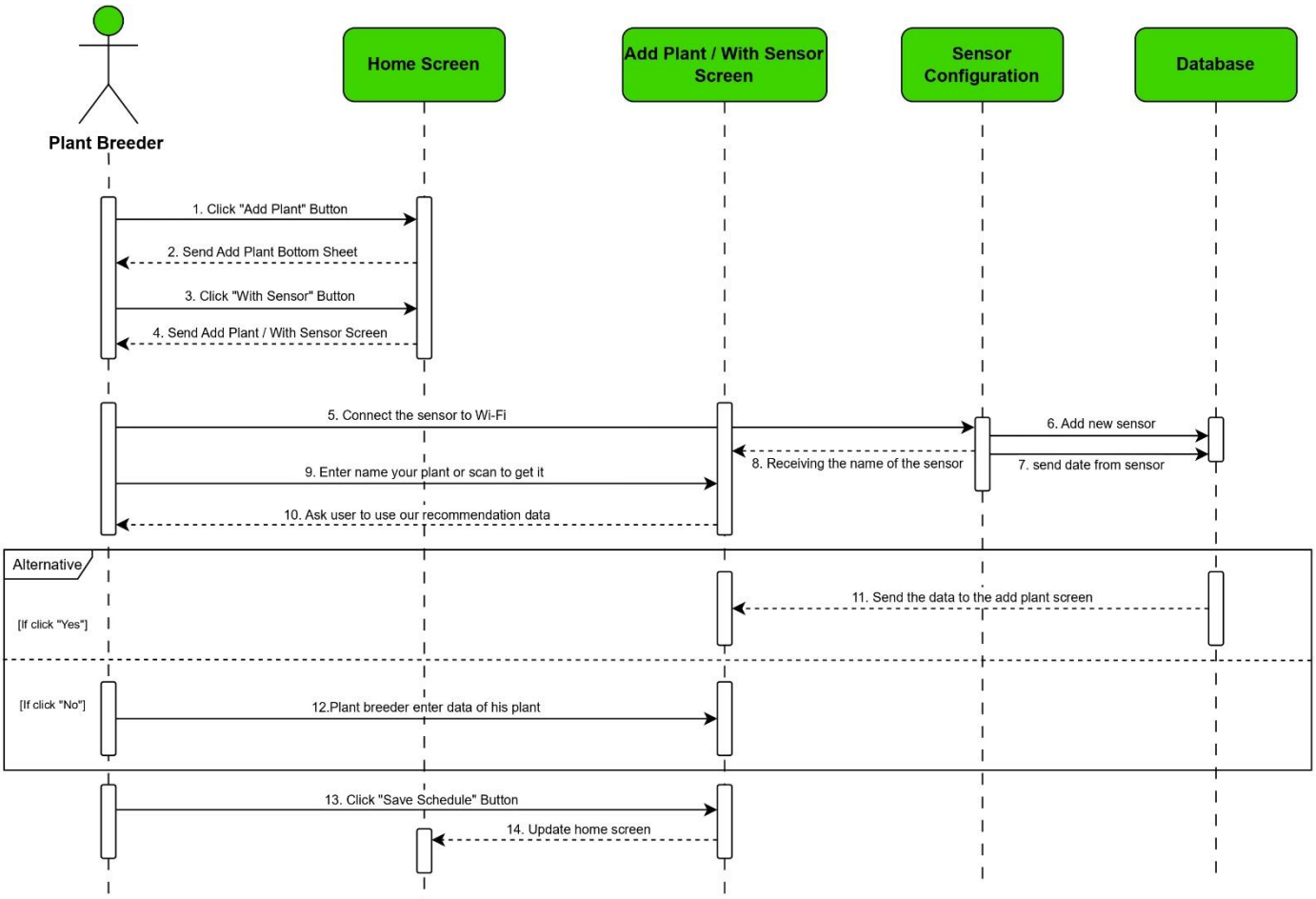


Figure 4.4 :Sequence Diagram – Add Plant / With Sensor Process

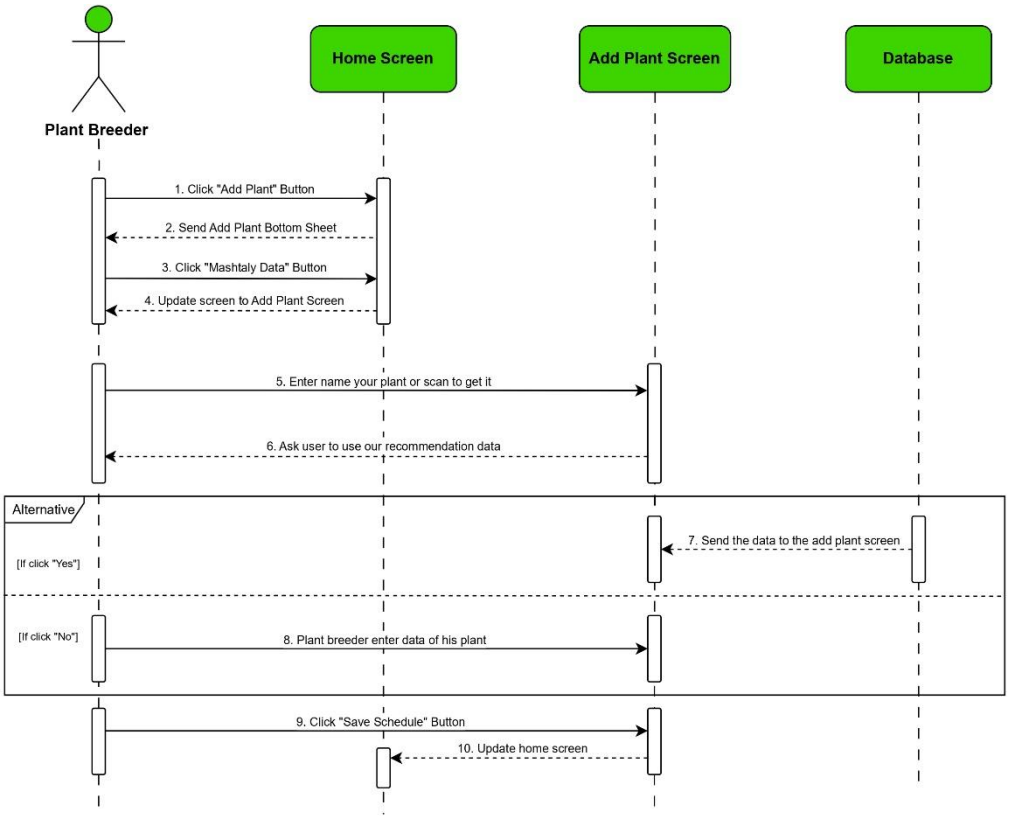


Figure 4.5: Sequence Diagram – Add Plant / **Mashtaly** Data Process

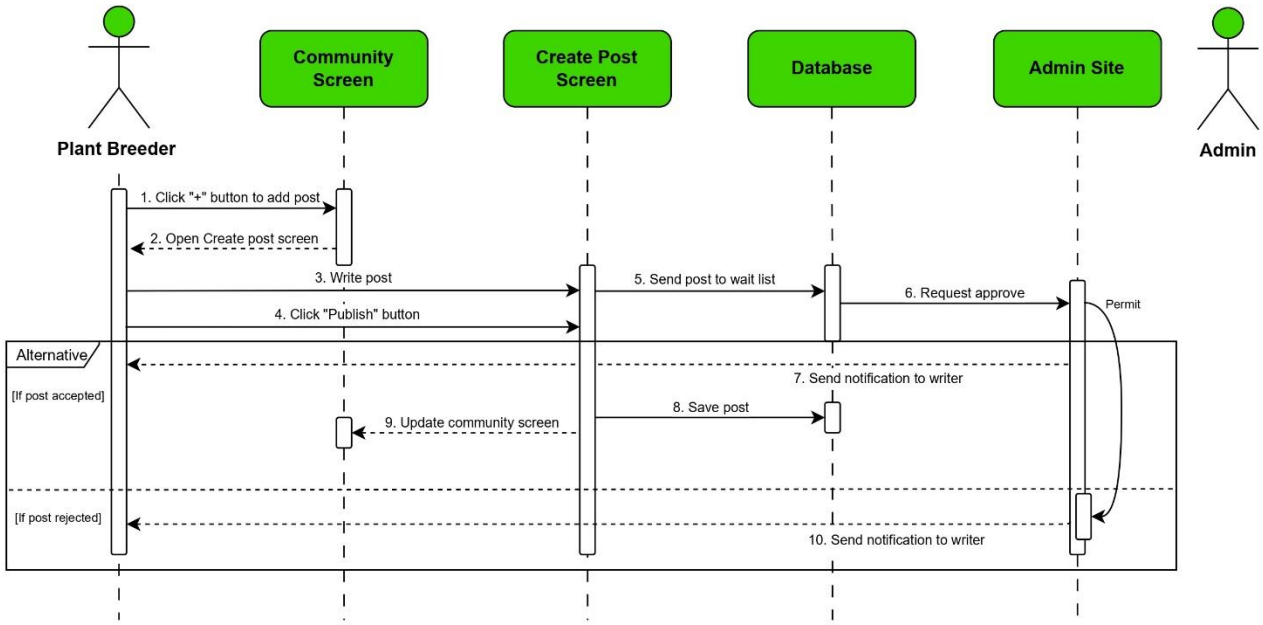


Figure 4.6: Sequence Diagram – Create Post / Article Process

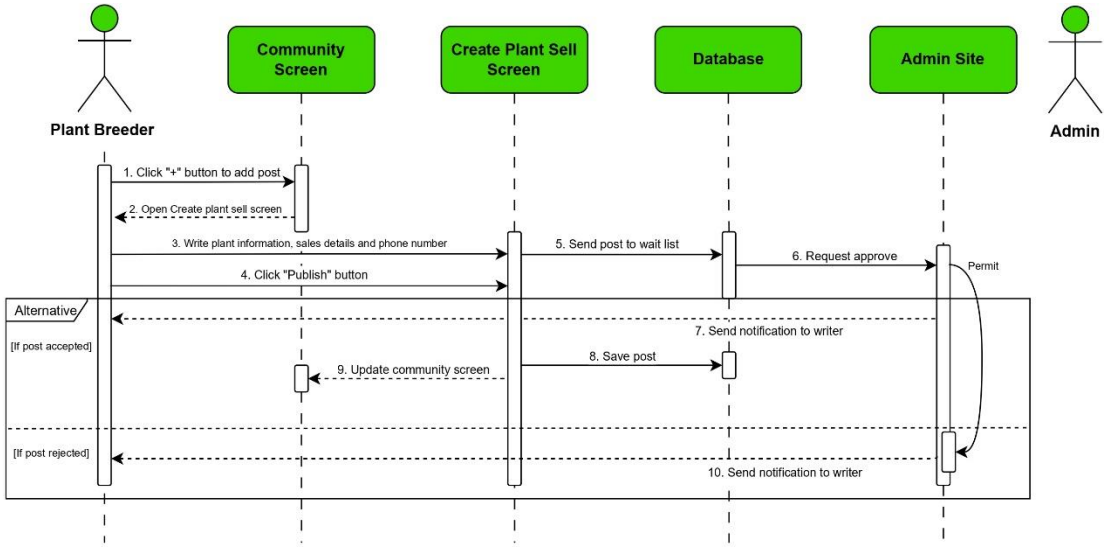


Figure 4.7: Sequence Diagram – Create Post / Sale Plant Process

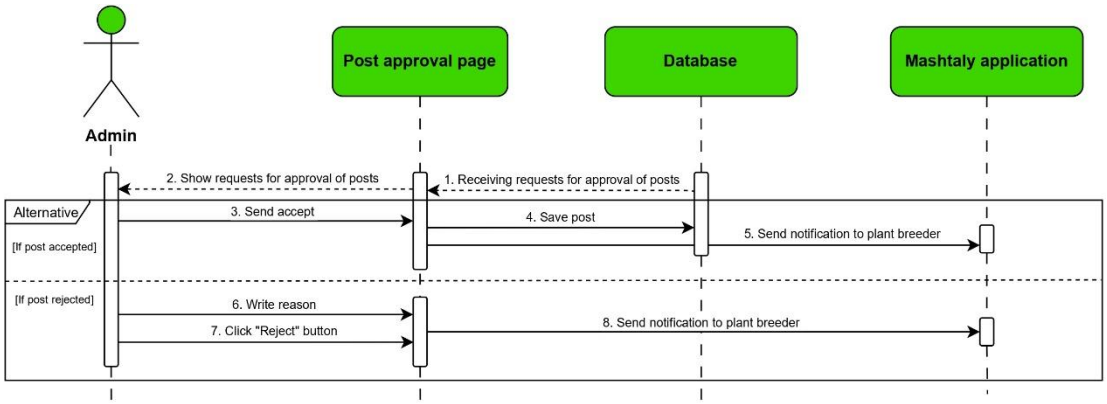


Figure 4.8: Sequence Diagram – Post approval Process

4.2.2 UML Class Diagram

The class diagrams offer a static view of the system, showcasing the classes, interfaces, associations, and collaborations within it. They provide a clear representation of the system's structure, aiding in communication, design, and development processes. Figure 4.9 shows a general class diagram in our system.

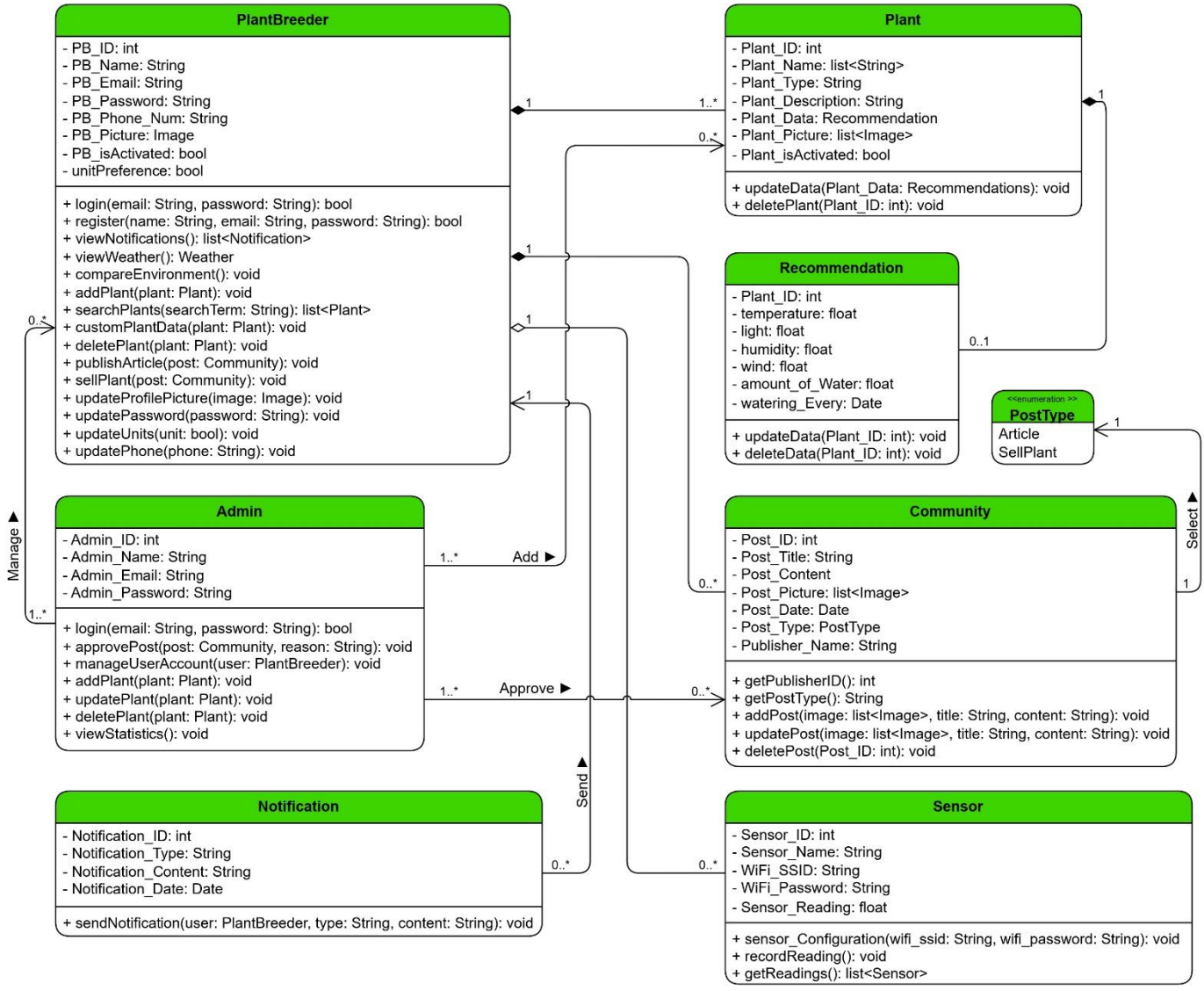


Figure 4.9: Class Diagram

4.2.3 ER Diagram

Entity-relationship is a model that describes how entities (such as plant breeder, plant, and community) relate to each other within a system, the figure 4.10 show the ER diagram for **Mashtaly** system.

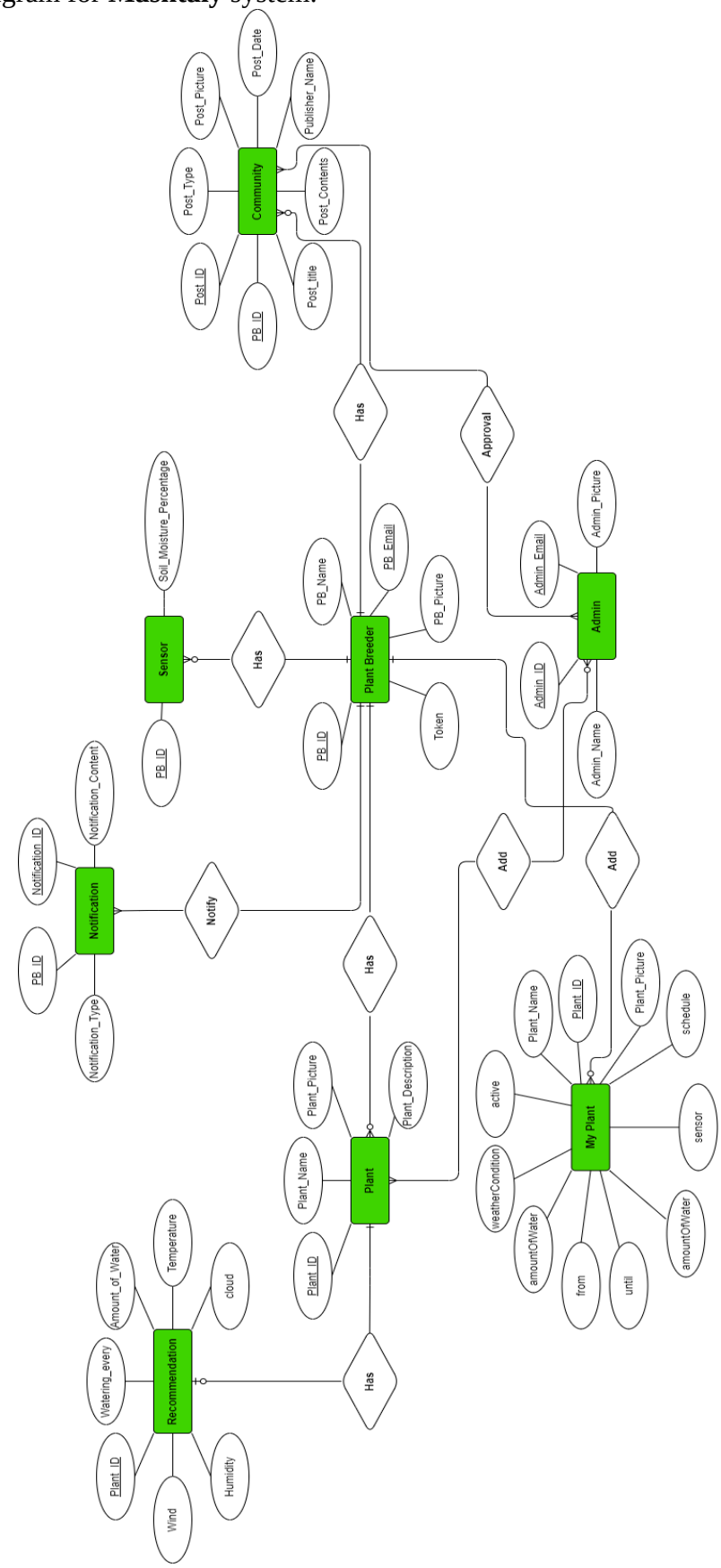


Figure 4.10 :ER Diagram

4.3 Graphical User Interface (GUI)

4.3.1 Mashtaly GUI

The mobile application interface for plant breeders is specifically tailored to their needs, with features such as the ability to log in, add, follow, and nurture plants, as well as publish articles, and sale plants. The interface also includes a range of features to help breeders monitor their plants, including a soil moisture sensor that sends notifications about plant watering needs and a camera API for identifying plants.

As shown in Figure 4.11, the visual identity of the **Mashtaly** application includes the logo that expresses the leaves of the plant in the form of the letter (M), the application icon, and the color scheme that reflects the plant colors and the type of font used.

Figure 4.11 :The Visual Identity of **Mashtaly** Application

Figure 4.12 shows the Introduction to the **Mashtaly** application, giving you a general idea of the application.

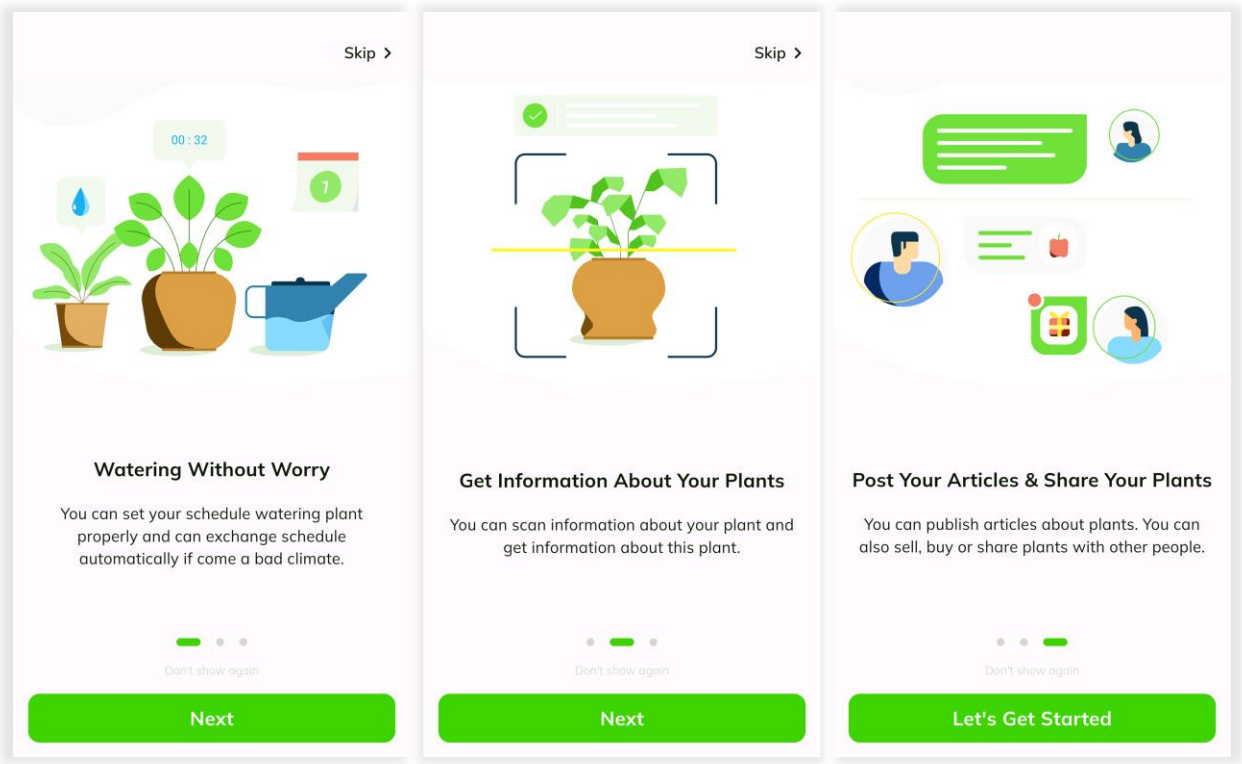


Figure 4.12: Introduction Screens

Figure 4.13 shows the login screen of the **Mashtaly** application, allowing plant breeders to securely access their accounts. To log in, breeders are required to enter their email and password. If there are no errors, they will be directed to the home page of the application. However, if there is an error with the email or password, a message will be displayed on the screen informing the breeder of the issue.

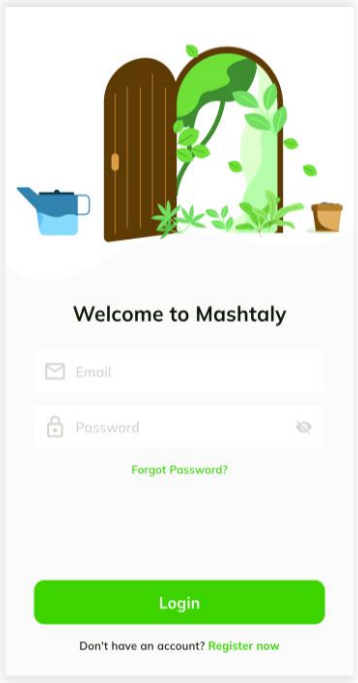


Figure 4.13: Login Screens

Figure 4.14 shows the Registration Screen. If the plant breeder does not have an account on the application, they can register through this screen by entering their name, email, and password.

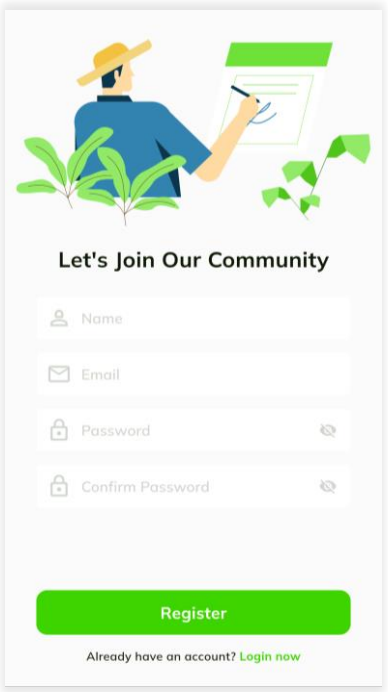


Figure 4.14: Registration Screen

Figure 4.15 shows the Reset Password Screens. If the plant breeder forgets their password, they can reset it by entering the email associated with their account and receiving an OTP via email. They can then receive email for reset password.

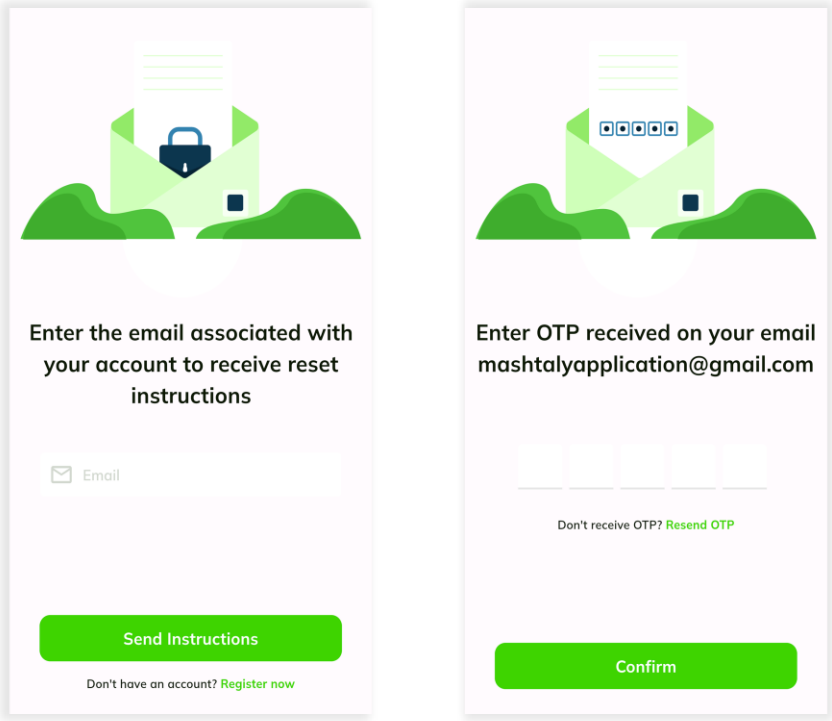


Figure 4.15: Reset Password Screens

Figure 4.16 shows the home screen where plant breeders can search for plants and view current weather conditions, watering schedule, and accurate information about light, wind, and humidity in their location. Plant breeders can also add their houseplants by clicking on "Add Plant" from the home screen.

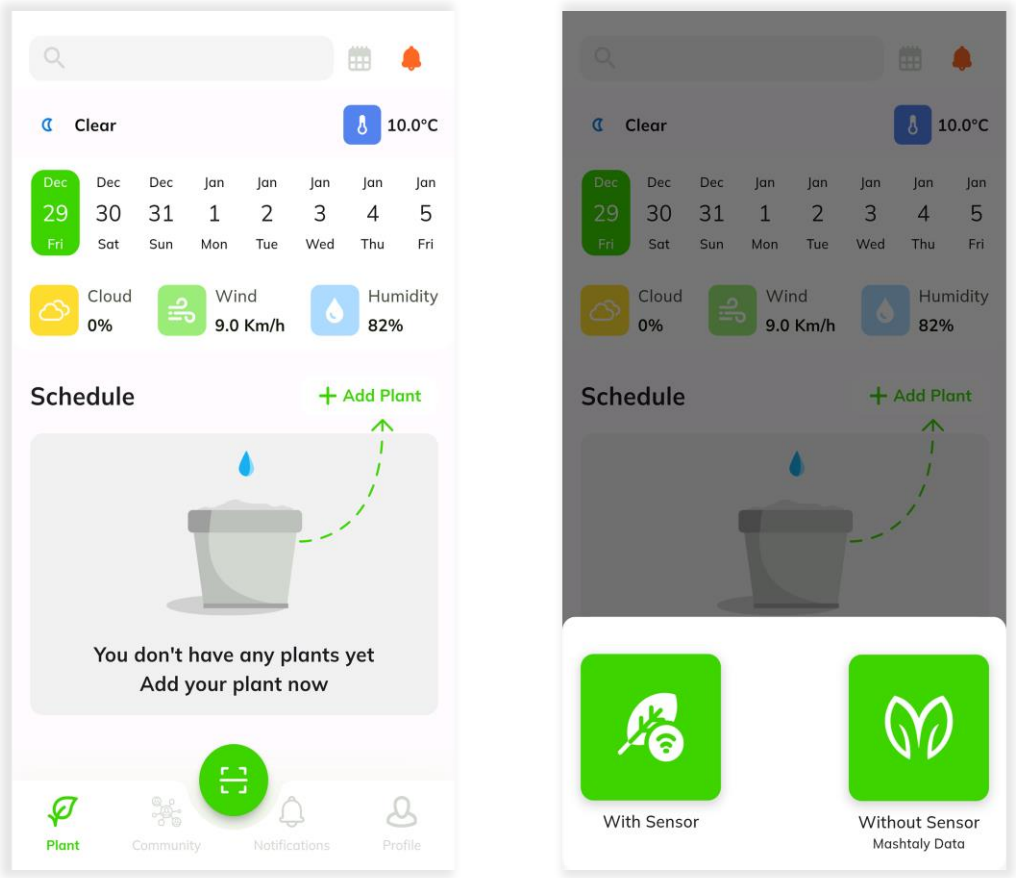


Figure 4.16 :Home Screen

As shown in Figure 4.16, two options will appear: the first is "With Sensor," which enables plant breeders to obtain accurate information about the moisture level of the plant soil; the other option is "**Mashtaly** Data" for those who do not want to take advantage of the sensor characteristics. They can add the plant through this option to enjoy a large number of plants in our database. Figure 4.17 shows the Add Plant/With Sensor screens.

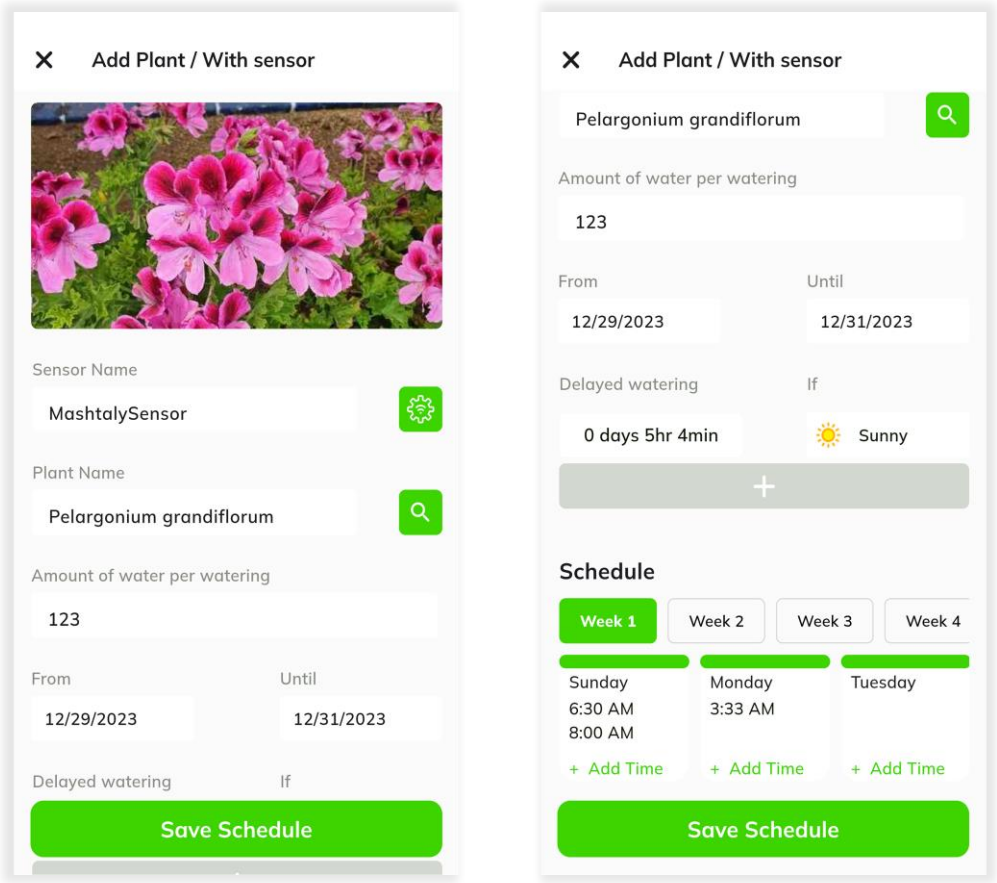


Figure 4.17: Add Plant / With Sensor Screens

Figure 4.18 shows the steps to configure the sensor to connect the ESP8266 (Wi-Fi Module) to the Wi-Fi using a library in Arduino software called WiFiManager[6]. This allows the sensor to send the percentage of soil moisture taken from the sensor to the database and display it in the application.

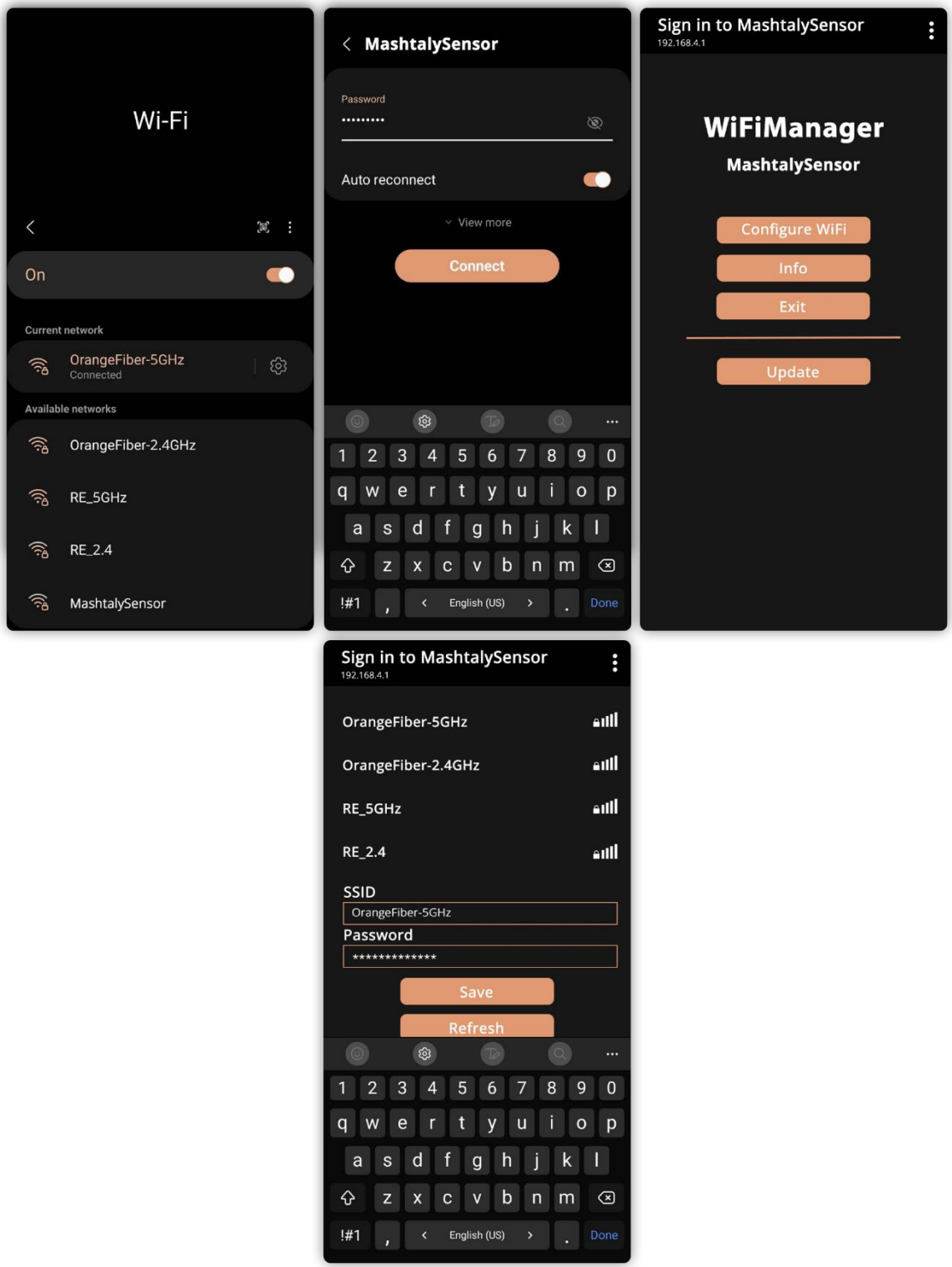


Figure 4.18: Sensor Configuration Screens

Figure 4.19 shows the process of adding a plant through "**Mashtaly** data," which involves utilizing the continuously updated data stored in the database. Additionally, plant breeders can use plant expert recommendations, a feature also depicted in Figure 4.17, or manually enter plant data.

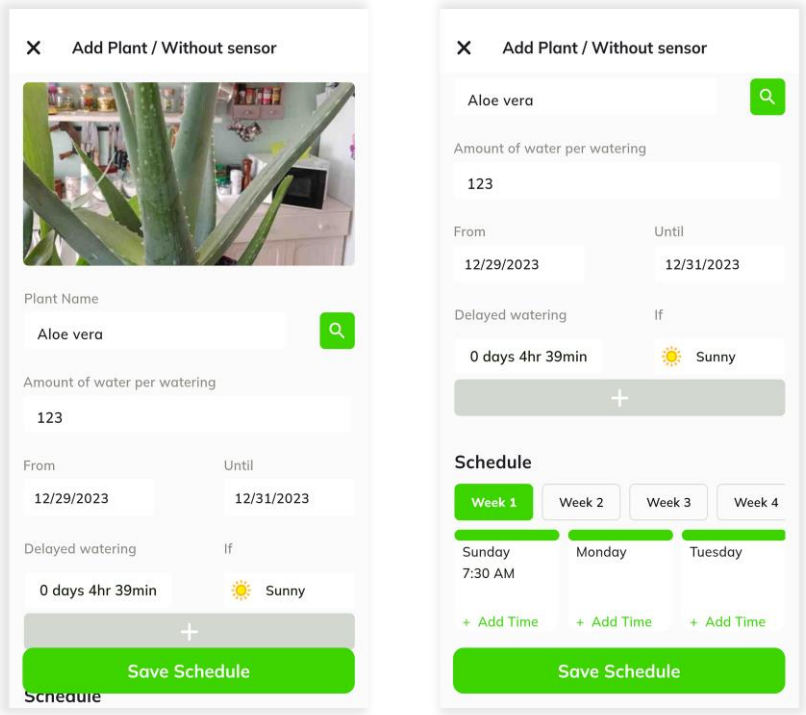


Figure 4.19: Add Plant Screens

Figure 4.20 and 4.21 show the plant scanning feature through the camera to obtain information about a plant. This feature is also available to extract the name of the plant in case the user does not know it, as shown in Figure 4.24 and 4.22.

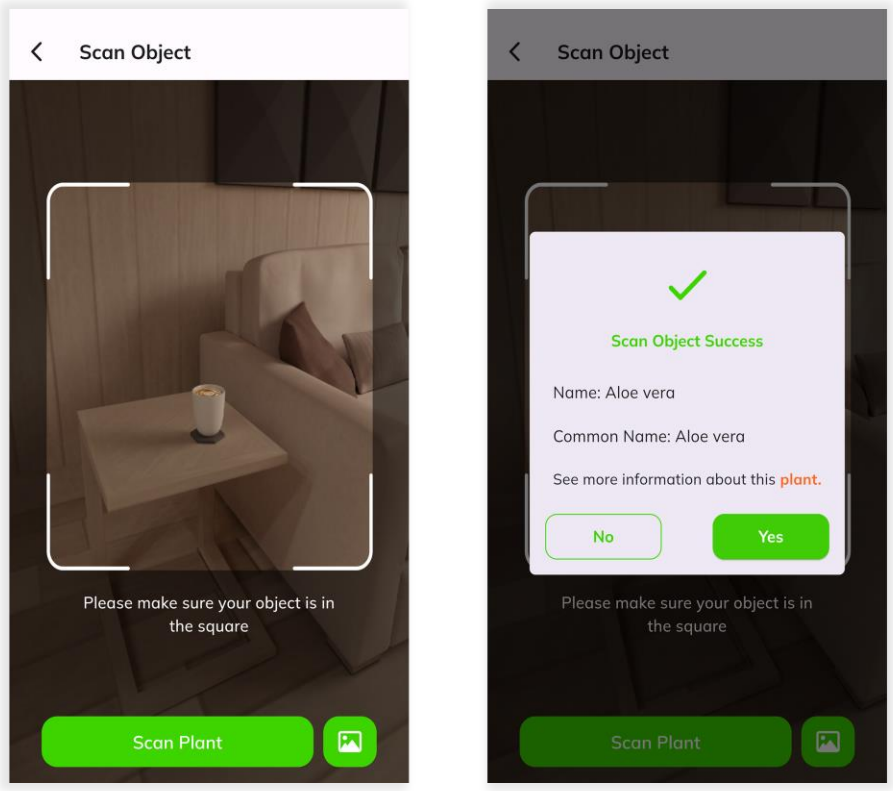


Figure 4.20: Scan Plant Screen

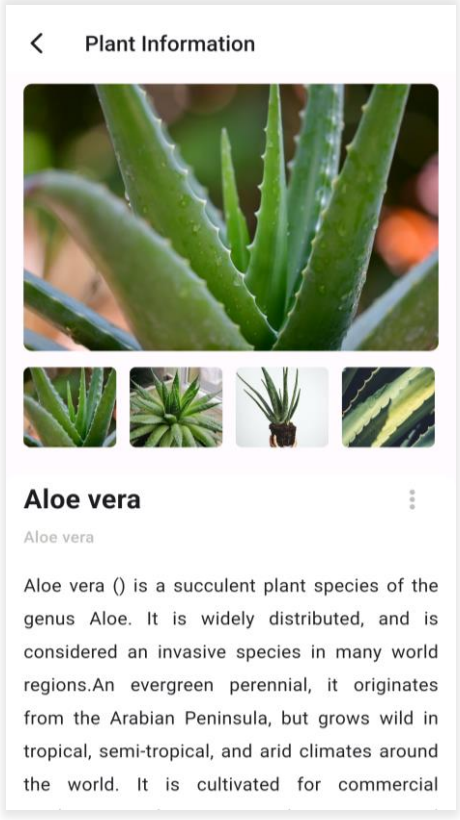


Figure 4.21: Plants Information Screen

As shown in Figure 4.21, there is a feature on the "Compare Environment" screen that enables plant breeders to compare the plant's ideal temperature with the current temperature, as well as the light and humidity levels. Based on this information, an assessment of the environment is provided.

Figure 4.22 shows that plant breeders have the option to deactivate a plant or edit its information. To address any confusion or errors, there is also an option to reset or delete the plant information.

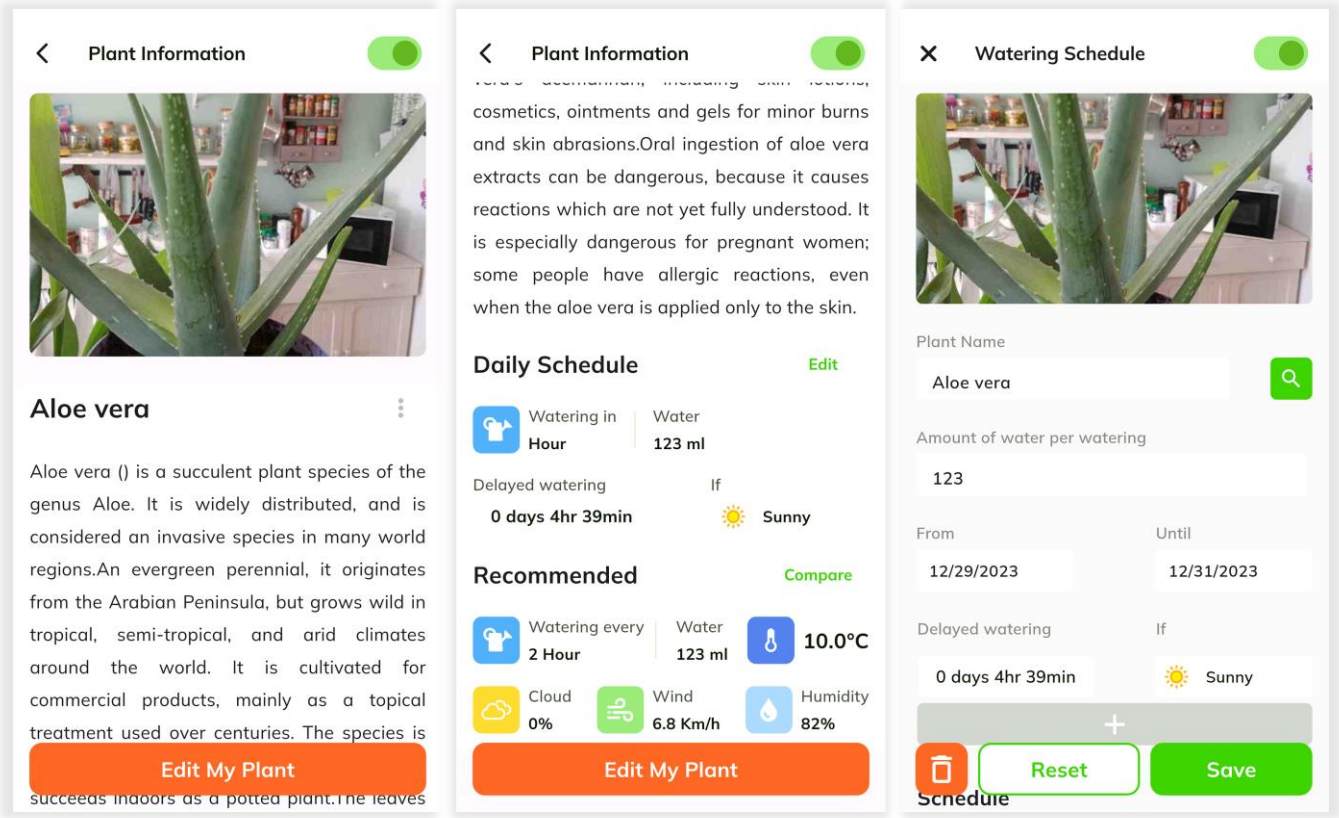


Figure 4.22: Plants Information and Watering Schedule Editing Screen

Figure 4.23 displays the latest plant-related articles written by plant breeders, along with the option for plant breeders to sale their plants through this page.

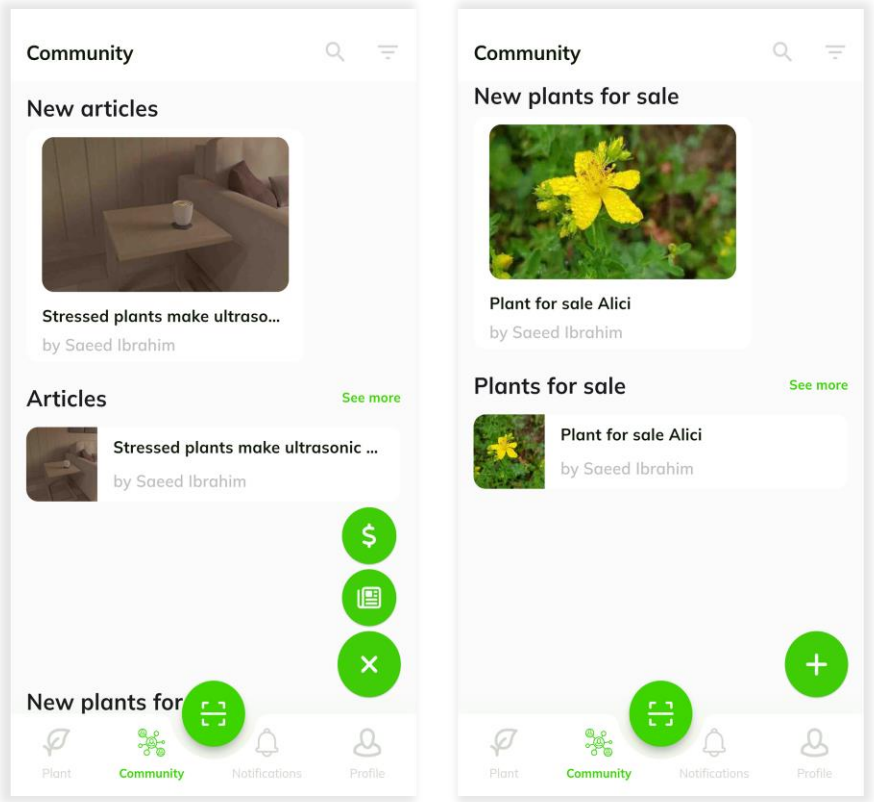


Figure 4.23: Community Screen

Figure 4.24 displays the screen where a plant breeder can publish an article or sale their plants. They can upload an image related to the topic and write the title and content of the article or details of the sale.

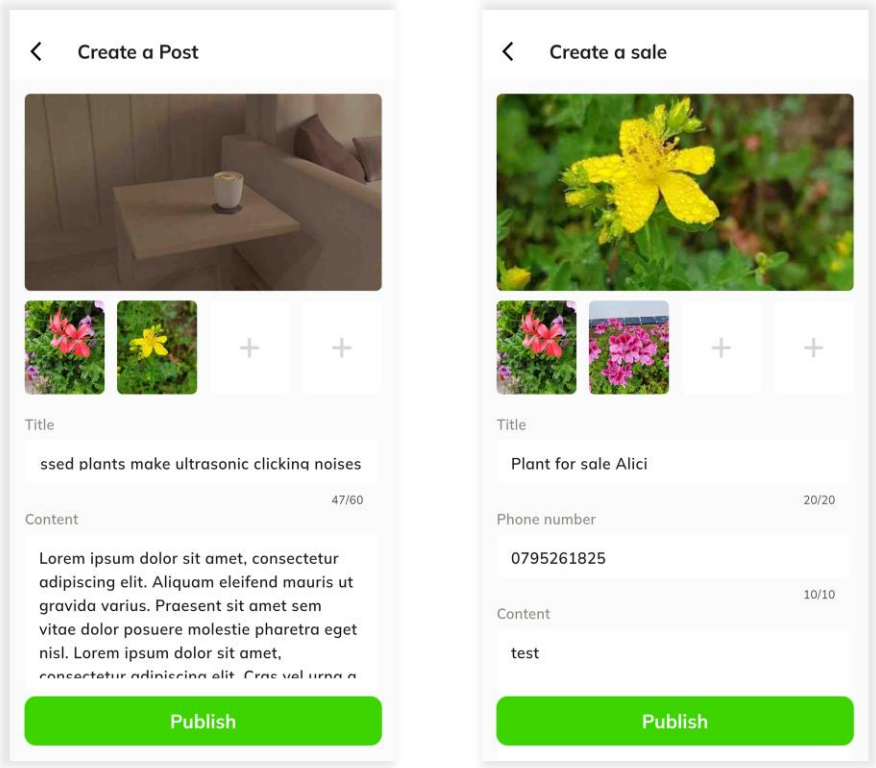


Figure 4.24 :Community / Create Post Screen

Figure 4.25 displays all notifications received by plant breeders, including reminders for watering plants, warnings, and notifications regarding the approval or rejection of published posts.

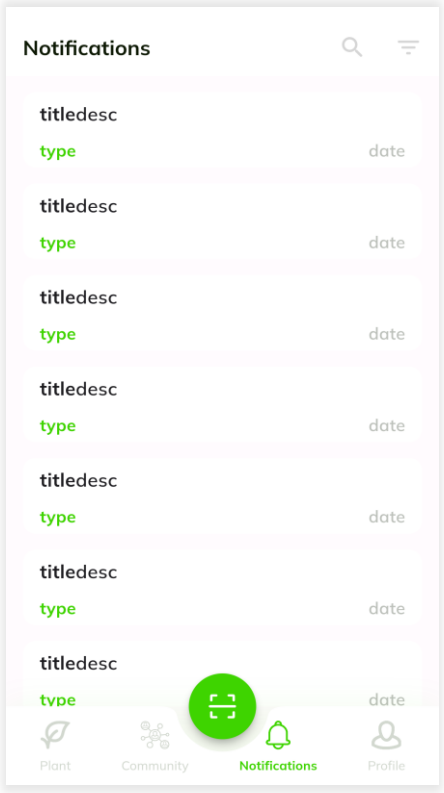


Figure 4.25: Notifications Screen

Figure 4.26 shows each plant breeder's profile screen, which contains all the posts they have published. It also includes account settings and other options, as shown below.

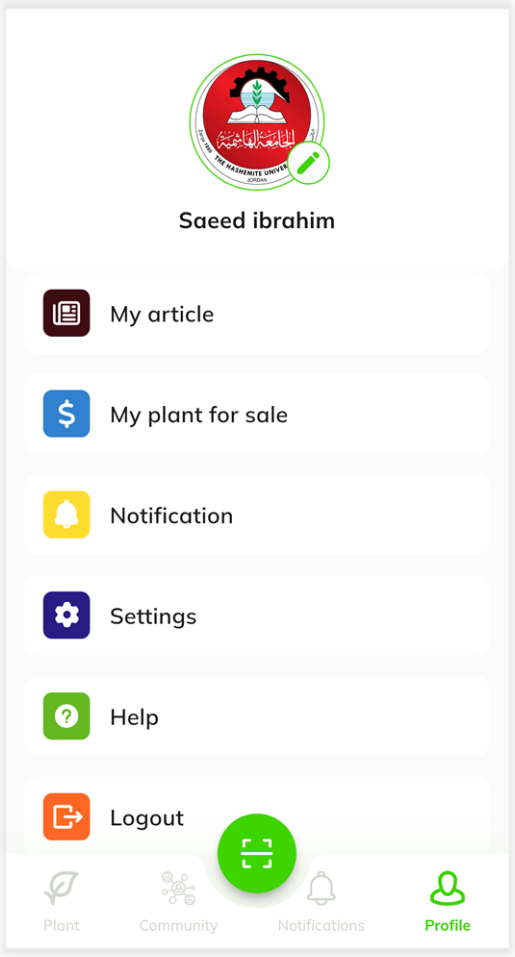


Figure 4.26: Profile Screen

4.3.2 Mashtaly Dashboard GUI

Figure 4.27 shows the statistics screen, which contains the statistics of publications either plants for sale or reports published by plant breeders. Also includes, as described below.

Figure 4.27: Statistics Screen

Figure 4.28 shows the Articles screen, which Articles reports published by plant breeders but awaiting approval by the administrator, as shown below.

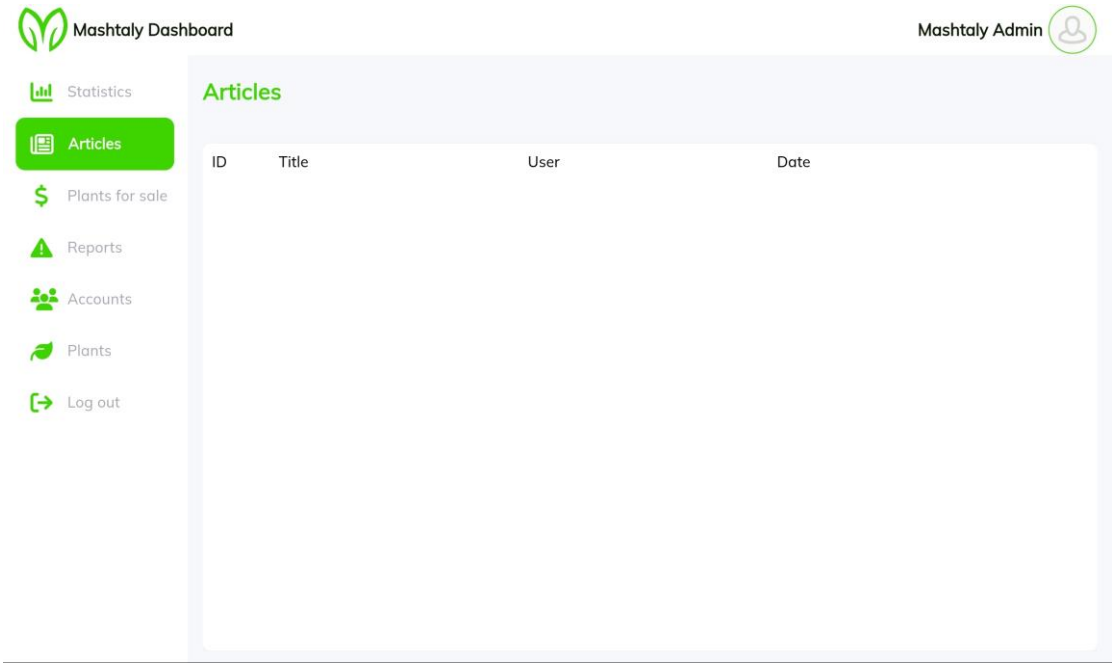


Figure 4.28: Articles Screen

Figure 4.29 shows the Plants for sale screen, which contains Plants for sale published by plant breeders but awaiting approval by the administrator, as shown below.

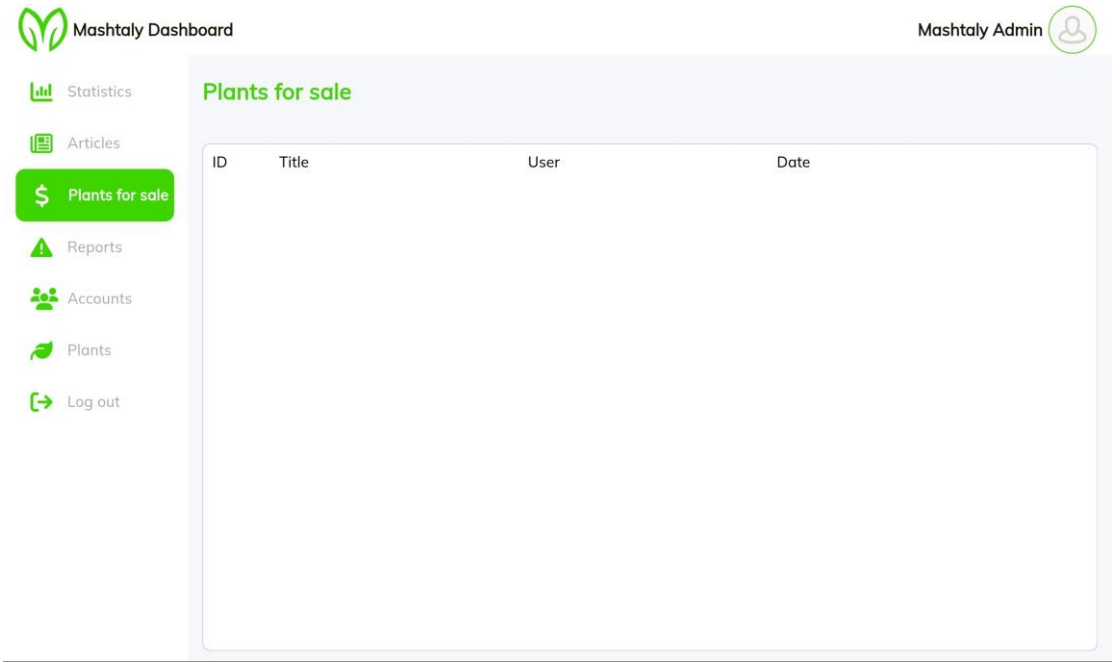


Figure 4.29: Plants for sale Screen

Figure 4.30 shows the Reports screen, which contains Reports on posts published by plant breeders but awaiting an administrator's review, as shown below.

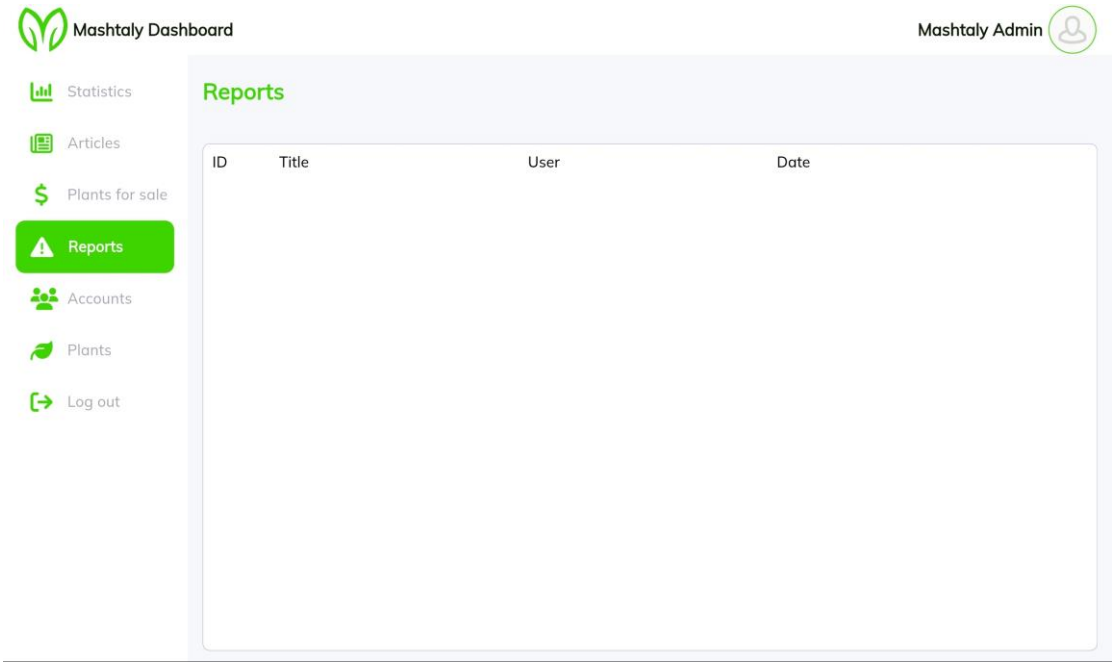


Figure 4.30: Reports Screen

Figure 4.31 shows the accounts screen, which contains the accounts of plant breeders through which you can send a password reset, activate and deactivate the account, as shown below.

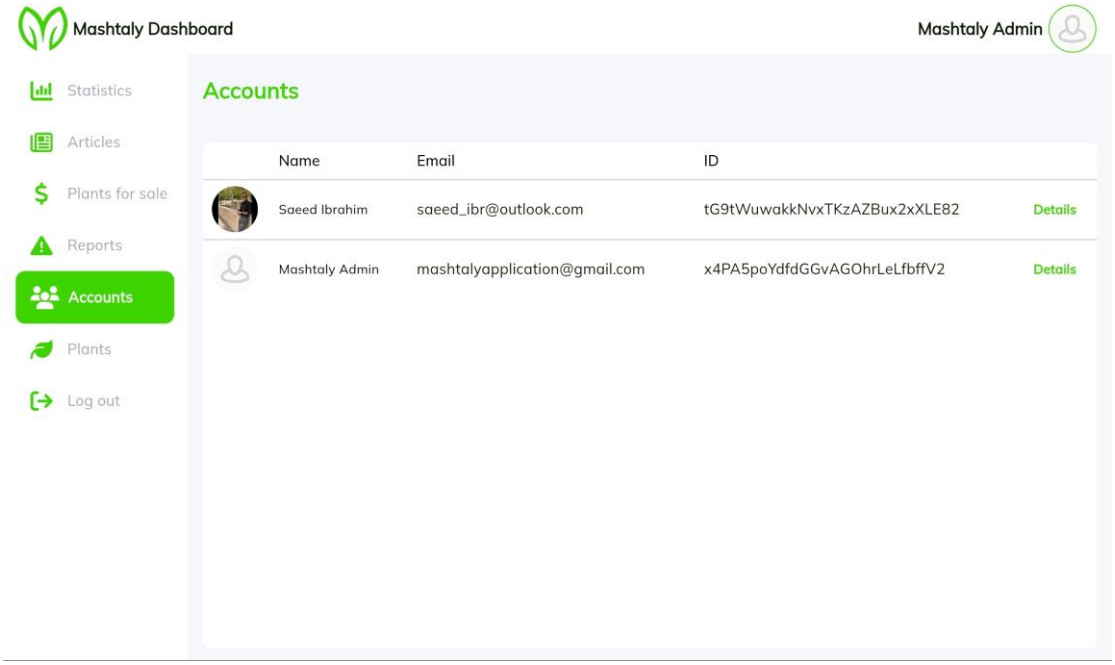


Figure 4.31: Accounts Screen

Figure 4.32 shows the plants screen, which contains all the plants in the **Mashtaly** app and which were added by the administrator through the "Add Plant" button, as shown below.

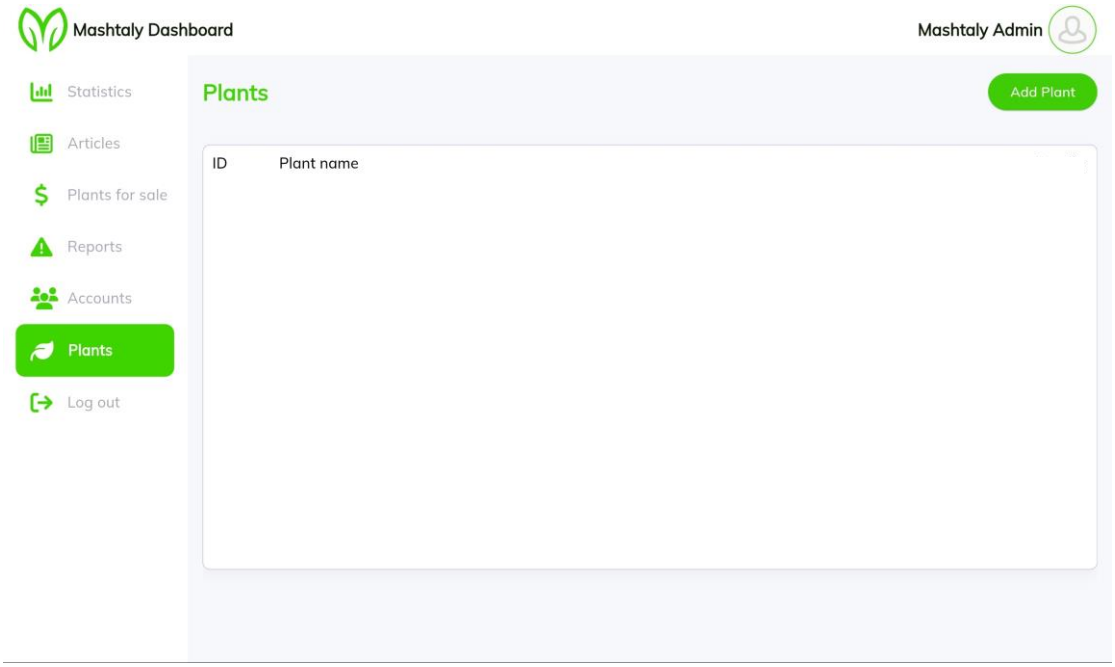


Figure 4.32: Plants Screen

CHAPTER 5: IMPLEMENTATION PLAN

5.1 Description of Implementation

First, we came up with the idea for **Mashtaly**. Then, we conducted an analysis and researched similar applications to learn from their feedback and avoid potential issues, such as complexity. Our goal was to make **Mashtaly** user-friendly.

To bring our idea to life, we created a prototype using Adobe Xd[7]. Next, we needed a programming language that could effectively implement the prototype. We chose Flutter[8], which offers a user-friendly UI. For the logical part of the application, we utilized Dart[9]. Visual Studio Code[10] served as our Integrated Development Environment (IDE).

To ensure user authentication and protection against attacks, we integrated Firebase[11] into the application. Firebase provides robust features for user authentication and security.

Furthermore, we utilized Firebase Cloud to store various types of data. This allowed us to securely store and manage user information, as well as other relevant data required for the functioning of **Mashtaly**.

By leveraging the capabilities of Flutter, Dart, Visual Studio Code, and Firebase, we were able to create a user-friendly application that provides seamless user authentication, data storage, and overall functionality.

5.2 Programming Language and Technology

- **Flutter:** This is an open-source UI software development kit created by Google. It allows developers to build cross-platform applications for Android, iOS, Linux, macOS, Windows, Google Fuchsia, and the web from a single codebase. In the case of the **Mashtaly** application, Flutter was employed to design user-friendly interfaces with just one codebase. This approach enabled the development team to create applications for multiple platforms, saving time and effort. Flutter also facilitated rapid development by leveraging the existing packages available on the pub.dev[12] platform, which provides pre-built functionality and components that could be easily integrated into the application. Overall, the use of Flutter in the development of the **Mashtaly** application streamlined the development process and ensured a consistent user experience across different platforms.

- **Dart:** The Dart language played a significant role in the development of the **Mashtaly** application. Dart is a programming language created by Google. It supported the development team by providing a language that is object-based and shares similarities with the Java programming language. This familiarity with Java made it easier for developers to transition and work with Dart.
  Additionally, the **Mashtaly** application implemented the Model View View Model (MVVM)[13] design pattern, which leveraged the capabilities of Dart. MVVM architecture separates the user interface from the underlying elements, such as database operations and API data retrieval. This division improved the performance of the application by ensuring a clear separation of concerns and promoting efficient code organization.
  By utilizing Dart and implementing the MVVM design pattern, the **Mashtaly** application achieved improved code maintainability, modularity, and scalability. The Dart language, combined with the MVVM architecture, helped optimize the performance of the application by providing a structured approach to managing data and interactions between the user interface and the application's backend components.

- **Soil Moisture Sensor[14]**: This is a device used to measure the moisture content in the soil. It plays a crucial role in plant care and irrigation systems by providing valuable information about soil moisture levels. The sensor is typically inserted into the ground near the plant's root zone, where it can detect the moisture level in the soil. This data is then used to determine when and how much water should be supplied to the plants.

- **Arduino IDE[15]:** This is a software platform used for programming and developing applications for Arduino microcontrollers. It provides a user-friendly interface and a set of tools that simplify the process of writing, compiling, and uploading code to Arduino boards.

- **Firebase:** Firebase is a powerful and versatile platform provided by Google for developing mobile and web applications. It offers a comprehensive set of tools

and services that streamline backend infrastructure, authentication, real-time database storage, cloud storage, and more. In the case of the **Mashtaly** application, Firebase played a crucial role in supporting its development. Firebase Authentication was utilized to ensure secure user authentication and protection against attacks. It provided a reliable and convenient way for users to register, log in, and authenticate their identities within the application. Firebase Cloud Firestore was used to store various types of data, including user information such as names and email addresses, as well as plant data and recommendations obtained for each plant. Cloud Firestore offers a flexible and scalable NoSQL database solution that allows the application to efficiently store and retrieve data in a structured manner. Additionally, Firebase Real-time Database was employed to synchronize soil moisture sensor data with the application's database in real-time. This feature ensured that the application could provide up-to-date and accurate information on soil moisture levels for the users' plants. Overall, the utilization of Firebase in the **Mashtaly** application provided a robust backend infrastructure, secure user authentication, efficient data storage, and real-time data synchronization. These features enhanced the overall performance and functionality of the application, providing a reliable and seamless user experience.

5.3 Part of Implementation

Following the development plan, here is an excerpt of the code employed in the implementation of **Mashtaly**. For the remaining portion, please refer to Appendix A:

main.dart

```dart
import 'package:connectivity_plus/connectivity_plus.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_messaging/firebase_messaging.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:mashtaly_app/Business_Layer/cubits/show_sensor_data/cubit/show_sensor_data_cubit.dart';
import 'package:mashtaly_app/Services/notification_service.dart';
import 'package:shared_preferences/shared_preferences.dart';

import 'Animations/no_connection_screen.dart';
import 'Animations/splash_screen.dart';
import 'Auth/auth.dart';
import 'Business_Layer/cubits/add_plant/add_plant_Cubit.dart';
import 'Business_Layer/cubits/show_weather/weatherCubit.dart';
import 'Constants/colors.dart';
import 'Presentation_Layer/Screen/OnboradingScreen/onboarding_screen.dart';

void main() async {
  SystemChrome.setSystemUIOverlayStyle(
      const SystemUiOverlayStyle(statusBarColor: Colors.black));
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  await notificationInitialize();
  FirebaseMessaging.onBackgroundMessage(backgroundHandler);
  runApp(const SplashScreenApp());
}

class SplashScreenApp extends StatelessWidget {
  const SplashScreenApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: SplashScreen(
        onSplashFinished: () async {
          var connectivityResult = await Connectivity().checkConnectivity();
          bool hasInternet = connectivityResult != ConnectivityResult.none;

          SharedPreferences prefs = await SharedPreferences.getInstance();
          int? isViewed = prefs.getInt('onBoard');

          runApp(MyApp(
            isViewed: isViewed,
            hasInternet: hasInternet,
          ));
        },
      ),
    );
  }
}

class MyApp extends StatelessWidget {
  final int? isViewed;
  final bool hasInternet;

  const MyApp({Key? key, this.isViewed, required this.hasInternet})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: hasInternet
          ? MashtalyApp(
              isViewed: isViewed,
            )
          : const NoConnectionScreen(),
    );
  }
}

class MashtalyApp extends StatelessWidget {
  final int? isViewed;

  const MashtalyApp({Key? key, required this.isViewed}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MultiBlocProvider(
      providers: [
        BlocProvider<WeatherCubit>(
          create: (context) => WeatherCubit(),
        ),
        BlocProvider<AddPlantCubit>(
          create: (context) => AddPlantCubit(),
        ),
        BlocProvider<ShowSensorDataCubit>(
          create: (context) => ShowSensorDataCubit(),
        ),
      ],
      child: Builder(builder: (context) {
        return MaterialApp(
          debugShowCheckedModeBanner: false,
          theme: ThemeData(
            fontFamily: 'Mulish',
            // datePickerTheme: const DatePickerThemeData(
            //   backgroundColor: Colors.white,
            // ),
            textSelectionTheme: TextSelectionThemeData(
              selectionColor: tPrimaryActionColor.withOpacity(.5),
              cursorColor: tPrimaryActionColor.withOpacity(.6),
              selectionHandleColor: tPrimaryActionColor.withOpacity(1),
            ),
          ),
          home: isViewed != 0 ? const OnBoardingScreen() : const Auth(),
        );
      }),
    );
  }
}
```

Figure 5.1: Main

onboarding_Model .dart

```dart
import '../../Constants/assets.dart';
import '../../Constants/text_strings.dart';

class OnBoardingModel {
  String img;
  String text;
  String desc;

  OnBoardingModel({required this.img, required this.text, required this.desc});
}

List<OnBoardingModel> screens = <OnBoardingModel>[
  OnBoardingModel(
      img: Assets.assetsImagesOnboardingImagesOnboarding1,
      text: tOnBoardingTitle1,
      desc: tOnBoardingSubTitle1),
  OnBoardingModel(
      img: Assets.assetsImagesOnboardingImagesOnboarding2,
      text: tOnBoardingTitle2,
      desc: tOnBoardingSubTitle2),
  OnBoardingModel(
      img: Assets.assetsImagesOnboardingImagesOnboarding3,
      text: tOnBoardingTitle3,
      desc: tOnBoardingSubTitle3),
];
```

Figure 5.2: Introduction Screen Model

onboarding_screen.dart

```dart
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:shared_preferences/shared_preferences.dart';

import '../../../Auth/auth.dart';
import '../../../Constants/colors.dart';
import '../../Widget/onboarding_model.dart';

class OnBoardingScreen extends StatefulWidget {
  const OnBoardingScreen({super.key});

  @override
  State<OnBoardingScreen> createState() => _OnBoardingScreenState();
}

class _OnBoardingScreenState extends State<OnBoardingScreen> {
  int currentIndex = 0;
  late PageController _pageController;

  @override
  void initState() {
    _pageController = PageController(initialPage: 0);
    super.initState();
  }

  @override
  void dispose() {
    _pageController.dispose();
    super.dispose();
  }

  _storeOnBoaringInfo() async {
    int isViewed = 0;
    SharedPreferences prefs = await SharedPreferences.getInstance();
    await prefs.setInt('onBoard', isViewed);
  }

  Route _createRoute() {
    return PageRouteBuilder(
      pageBuilder: (context, animation, secondaryAnimation) => const Auth(),
      transitionsBuilder: (context, animation, secondaryAnimation, child) {
        const begin = Offset(0.0, 1.0);
        const end = Offset.zero;
        const curve = Curves.ease;

        var tween =
            Tween(begin: begin, end: end).chain(CurveTween(curve: curve));

        return SlideTransition(
          position: animation.drive(tween),
          child: child,
        );
      },
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      extendBodyBehindAppBar: true,
      appBar: AppBar(
        elevation: 0,
        actions: [
          TextButton(
            onPressed: () {
              Navigator.pushReplacement(
                context,
                MaterialPageRoute(
                  builder: (context) => const Auth(),
                ),
              );
            },
            child: Row(
              children: [
                currentIndex != 2
                    ? const Text(
                        "Skip",
                        style: TextStyle(
                          color: tPrimaryTextColor,
                          fontWeight: FontWeight.w600,
                          fontSize: 16,
                        ),
                      )
                    : const Text(""),
                currentIndex != 2
                    ? const Icon(
                        Icons.keyboard_arrow_right_rounded,
                        color: tPrimaryTextColor,
                        size: 24,
                      )
                    : const Text(""),
              ],
            ),
          ),
        ],
      ),
      body: PageView.builder(
        itemCount: screens.length,
        controller: _pageController,
        physics: const PageScrollPhysics(),
        onPageChanged: (int index) {
          setState(() {
            currentIndex = index;
          });
        },
        itemBuilder: (context, index) {
          return Column(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            crossAxisAlignment: CrossAxisAlignment.center,
            mainAxisSize: MainAxisSize.max,
            children: [
              Image.asset(
                screens[index].img,
                fit: BoxFit.fill,
                alignment: Alignment.topCenter,
              ),
```

onboarding_screen.dart

```dart
Column(
              children: [
                Padding(
                  padding: const EdgeInsets.only(
                    left: 16,
                    right: 16,
                    bottom: 20,
                  ),
                  child: Text(
                    screens[index].text,
                    textAlign: TextAlign.center,
                    style: const TextStyle(
                      color: tPrimaryTextColor,
                      fontWeight: FontWeight.w700,
                      fontSize: 28,
                    ),
                  ),
                ),
                Padding(
                  padding: const EdgeInsets.only(left: 16, right: 16),
                  child: Text(
                    screens[index].desc,
                    textAlign: TextAlign.center,
                    style: const TextStyle(
                      color: tPrimaryTextColor,
                      fontWeight: FontWeight.w400,
                      fontSize: 16,
                    ),
                  ),
                ),
              ],
            ),
            Column(
              children: [
                SizedBox(
                  height: 10,
                  child: ListView.builder(
                    itemCount: screens.length,
                    shrinkWrap: true,
                    scrollDirection: Axis.horizontal,
                    itemBuilder: (_, index) {
                      return Row(
                        mainAxisAlignment: MainAxisAlignment.center,
                        children: [
                          Container(
                            margin:
                                const EdgeInsets.symmetric(horizontal: 8.0),
                            height: 8,
                            width: currentIndex == index ? 25 : 8,
                            decoration: BoxDecoration(
                              color: currentIndex == index
                                  ? tPrimaryActionColor
                                  : tSecondActionColor,
                              borderRadius: BorderRadius.circular(4),
                            ),
                          ),
                        ],
                      );
                    },
                  ),
                ),
                Padding(
                  padding: const EdgeInsets.only(left: 16, right: 16),
                  child: TextButton(
                    onPressed: () async {
                      await _storeOnBoaringInfo();
                      Navigator.pushReplacement(
                        context,
                        MaterialPageRoute(
                          builder: (context) => const Auth(),
                        ),
                      );
                    },
                    child: const Text(
                      "Don't show again",
                      textAlign: TextAlign.center,
                      style: TextStyle(
                        color: tSecondActionColor,
                        fontWeight: FontWeight.w300,
                        fontSize: 12,
                      ),
                    ),
                  ),
                ),
                InkWell(
                  onTap: () {
                    if (index == screens.length - 1) {
                      Navigator.pushReplacement(
                          context,
                          MaterialPageRoute(
                              builder: (context) => const Auth()));
                    }
                    _pageController.nextPage(
                      duration: const Duration(microseconds: 300),
                      curve: Curves.bounceIn,
                    );
                  },
                  child: Stack(
                    alignment: Alignment.center,
                    children: [
                      Padding(
                        padding: const EdgeInsets.only(
                          bottom: 15,
                        ),
                        child: Container(
                          height: 50,
                          width: 300,
                          decoration: BoxDecoration(
                              color: tPrimaryActionColor,
                              borderRadius: BorderRadius.circular(12)),
                          child: Row(
                            mainAxisAlignment: MainAxisAlignment.center,
                            mainAxisSize: MainAxisSize.max,
                            children: [
                              currentIndex != 2
                                  ? const Text(
                                      "Next",
                                      style: TextStyle(
                                        color: tThirdTextColor,
                                        fontWeight: FontWeight.bold,
                                        fontSize: 20,
                                      ),
                                    )
                                  : const Text(
                                      "Let's Get Started",
                                      style: TextStyle(
                                        color: tThirdTextColor,
                                        fontWeight: FontWeight.bold,
                                        fontSize: 20,
                                      ),
                                    ),
                            ],
                          ),
                        ),
                      )
                    ],
                  ),
                ),
              ],
            ),
          ],
        );
      },
    );
  }
}
```

Figure 5.3: Introduction Screen

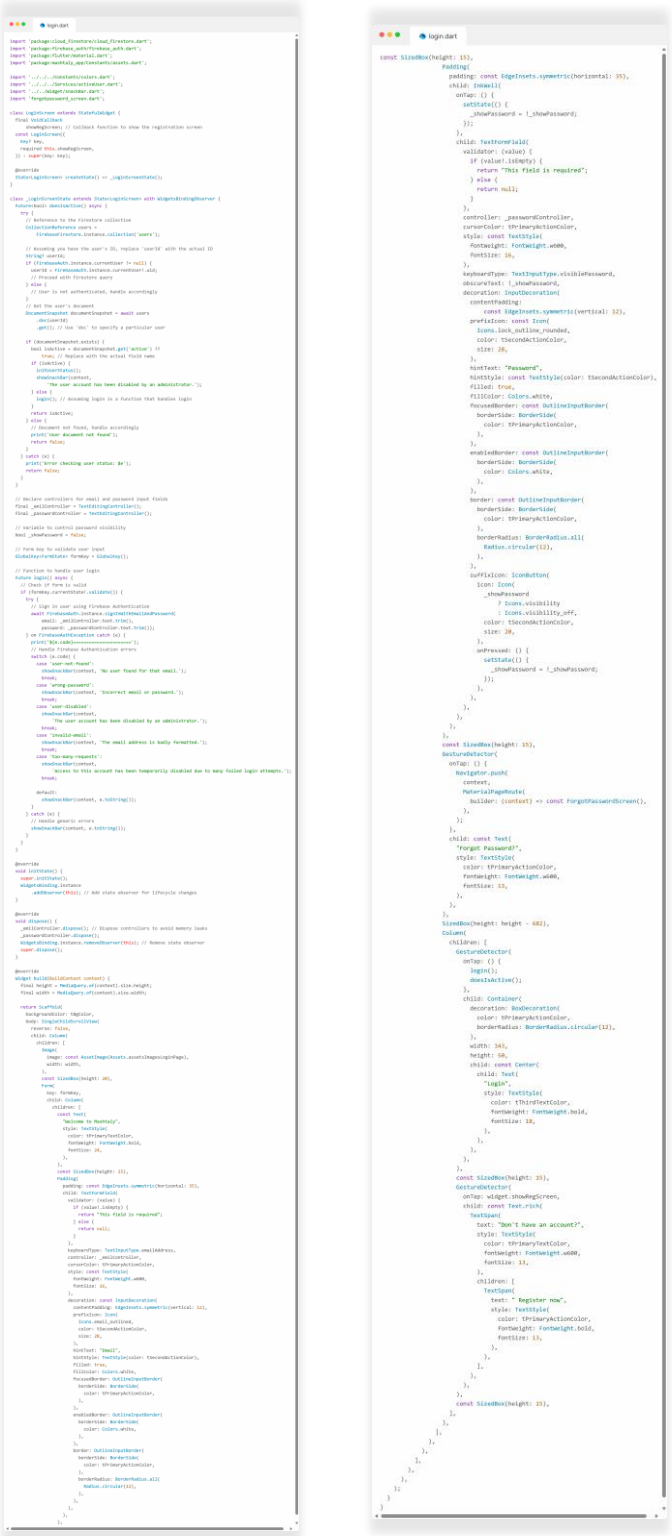


Figure 5.4: Login Screen

CHAPTER 6: TESTING PLAN

6.1 Introduction

This chapter presents an in-depth examination of the **Mashtaly** application, specifically focusing on testing its fundamental structure, functionality, and design. By scrutinizing these pivotal aspects, a nuanced understanding of the application's operational mechanisms and diverse capabilities emerges. The subsequent segments will detail the objectives and strategies employed to ensure the robust quality of the **Mashtaly** application.

6.2 Objectives

- **Comprehensive Overview**

  The primary objective is to conduct a thorough examination of the **Mashtaly** application, including its architecture, functionality, and design. This comprehensive overview will serve as the foundation for subsequent testing phases.

- **Critical Aspect Analysis**

  This objective focuses on the identification and testing of critical elements within the **Mashtaly** application. The goal is to ensure that crucial features operate seamlessly and to identify potential vulnerabilities.

- **Clarity and Understanding**

  Facilitate a clear and comprehensive understanding of the **Mashtaly** application among stakeholders through detailed documentation, knowledge-sharing sessions, and visual aids.

6.3 Testing Context

This section plays a pivotal role in framing the parameters for testing activities within the dynamic landscape of **Mashtaly** application development. This segment elucidates the scope of testing, outlining which functions will undergo scrutiny and which will not.

6.3.1 Test Scope

The testing scope is defined across various dimensions, encompassing the internal scope of the primary application, the internal scope specific to the application's dashboard, and the external scope applicable to Mashtaly. This comprehensive approach ensures a thorough examination of both the core functionality within the application and its associated dashboard, as well as the external interactions and dependencies that impact the overall performance and reliability of Mashtaly.

In scope (Mashtaly app)	In scope (Mashtaly dashboard)	Out scope
Login	Login	Scan plant API
Register	Approve posts	Weather API
See notification	Manage users accounts	**Mashtaly** sensor
See weather	Add plants	
Compare environment	See statistics	
Add plant	Review reports	
Manage plant		
Browse plants		
Publish an article		
Sale plants		
Manage profile		
Logout		

Table 6.3.1: Test Scope **Mashtaly** System

6.3.2 Test item

In the realm of software testing, a "test item" within the **Mashtaly** application context signifies a discrete component, feature, or facet that undergoes meticulous scrutiny. As the smallest unit in the testing process, it is subject to examination to validate its functionality, behavior, or performance. These test items may include specific functions, modules, interfaces, or other elements essential for validation within the broader testing endeavor. The overarching goal of testing these items is to ensure their seamless alignment with specified requirements and the fulfillment of their intended functionality.

Test item	Description
1. Login	The gateway for registered plant breeders and admins, users enter their unique email and password.
2. Register	New plant breeders enter their name, email and password.
3. Add plant	Enable plant breeders to add and record data using **Mashtaly's** database or the sensor system.
4. See weather	Plant breeders access real-time weather conditions via weatherapi.
5. Compare environment	Plant breeders can compare current weather conditions with optimal conditions for plants.
6. Browse plants	Plant breeders can swiftly access plant information by searching for names or scanning plants using the **Mashtaly** app's camera.
7. Publish an article	Plant breeders can share their expertise by creating and publishing articles on diverse plant topics.
8. Sale plants	Plant breeders can showcase and sell their plants.

Table 6.3.2.1: Test Item **Mashtaly** Application

Test item	Description
1. Approve posts	Admins can review and approve/reject posts by assessing content against set criteria and standards.
2. Manage users accounts	Admins oversee and adjust user settings, addressing guideline violations by disabling accounts if necessary. They assist with password issues.
3. Add plants	Admins efficiently manages the database by adding and updating plants.
4. See statistics	Admins can view sensor, user, and plant counts for performance monitoring.
5. Review reports	Admins can review the modified post and take appropriate action, which may involve either removing the post or retaining it based on the evaluation

Table 6.3.2.2: Test Item **Mashtaly** Dashboard

Test item	Description
1. **Mashtaly** sensor	Soil moisture sensor measures soil water content, sending data to the app via Wi-Fi.
2. Weather API	Identifies the user's location and retrieves weather data.
3. Scan plant API	Utilizes a camera for plant image capture and identification.

Table 6.3.2.3: Test Item Third Parties

6.4 Testing Approach

6.4.1 Unit Testing (White-box)

In the context of white-box testing, the internal structure, logic, and pathways of the units are examined comprehensively. This method requires an understanding of the code's internal workings.

- **Mashtaly app**

- **Login**

The following test cases aim to validate and enhance the login functionality by thoroughly testing all potential input scenario.

Test Case	Input	Expected Output	Actual Output
Valid credentials.	Email: valid email Password: valid password	Enter to main screen.	Enter to main screen of the application.
Invalid credentials.	Email: invalid email Password: valid password	Show error message.	Show message 'No user found for that email.'.
Invalid credentials.	Email: valid email Password: invalid password	Show error message.	Show message 'Incorrect in email or password.'.
After multiple failed login attempts.	Email: valid email Password: invalid password	Show error message.	Show message 'Access to this account has been temporarily disabled due to many failed login attempts.'.

Table 6.4.1.1: Test Case for Login

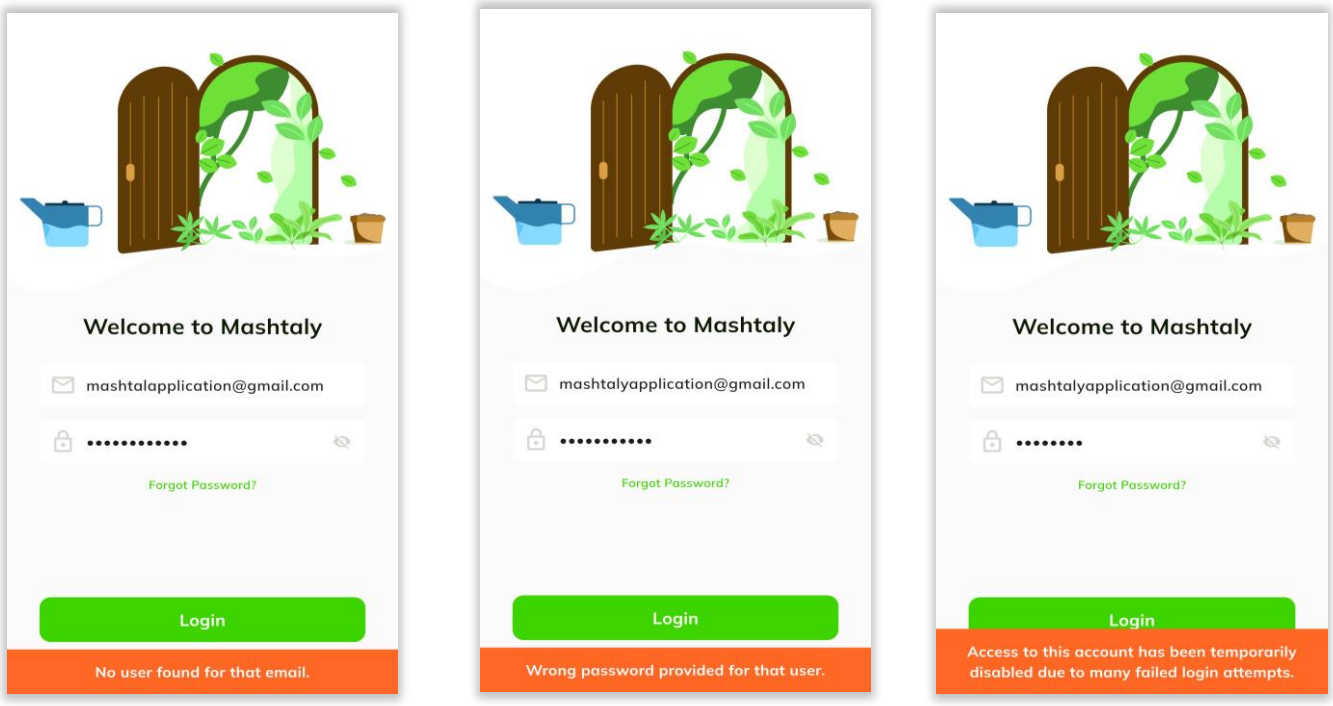


Figure 6.4.1.1: Handle Test Case from **Mashtaly** App

- **Register**

  The following test cases aim to validate and enhance the register functionality by thoroughly testing all potential input scenarios.

Test Case	Input	Expected Output	Actual Output
Valid credentials.	Name: valid name Email: valid email Password: valid password	Send Verify email.	Send Verify your email to the entered email and then enter the main screen.
Duplicate email registrations.	Name: valid name Email: same valid email Password: valid password	Show error message.	Show message 'The account already exists for that email.'.
Incomplete.	Name: Incomplete Email: valid email Password: valid password	Show error message.	Show message 'This field is required.'.
Invalid data.	Name: valid name Email: same valid email Password: invalid password	Show error message.	Show message 'The password is not match.'.

Table 6.4.1.2: Test Case for Register

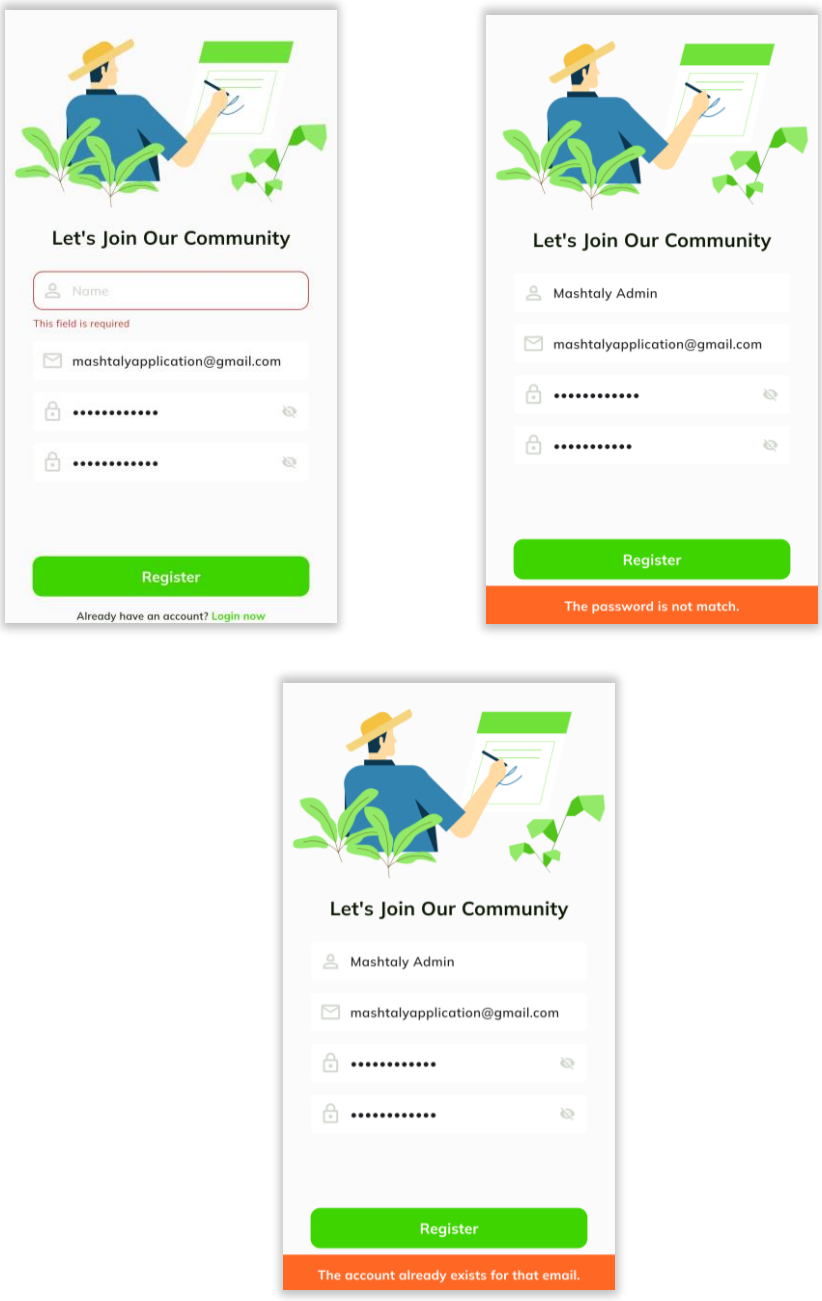


Figure 6.4.1.2: Handle Test Case from **Mashtaly** App

- **Add Plant**

  The following test cases aim to validate and enhance the add Plant functionality by thoroughly testing all potential input scenarios.

Test Case	Input	Expected Output	Actual Output
Validate successful addition of a plant to the database.	Plant photo: uploaded Plant Name: fetch from API Amount of water: optional From date: optional Until date: optional Delated watering conditions: optional Time schedule: optional	Input plant information and update main screen.	Sending plant information to the database and listing it under the user id and in the my plants field, and the main screen has been updated.
Test the handling of errors when adding a plant with incomplete.	Image: uploaded Plant Name: incomplete Amount of water: optional From date: optional Until date: optional Delated watering conditions: optional Time schedule: optional	Show error message.	Show message ‘Please enter plant name.’.

Table 6.4.1.3: Test Case for add plant

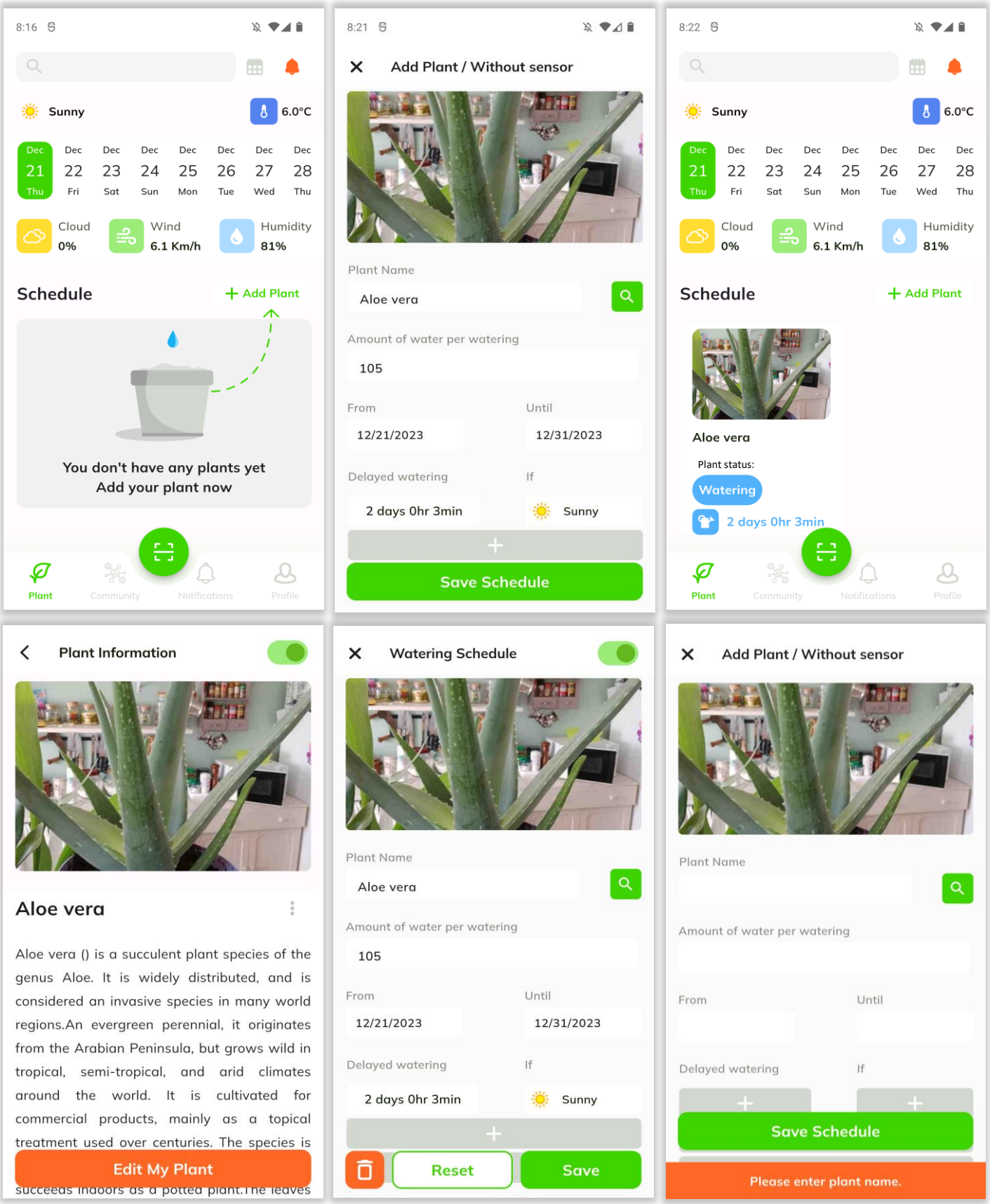


Figure 6.4.1.3: Handle Test Case from **Mashtaly** App

- **See weather**

  The following test cases aim to validate and enhance the see weather functionality by thoroughly testing all potential input scenarios.

Test Case	Input	Expected Output	Actual Output
Validate real-time weather data retrieval.	Latitude: current location Longitude: current location	Show weather data at real-time.	Show data from postman in bellow.
Confirm accurate display of weather information on the user interface.	Latitude: current location Longitude: current location	Show weather information at the top of the home screen.	Show weather information at the top of the home screen.

Table 6.4.1.4: Test Case for See Weather

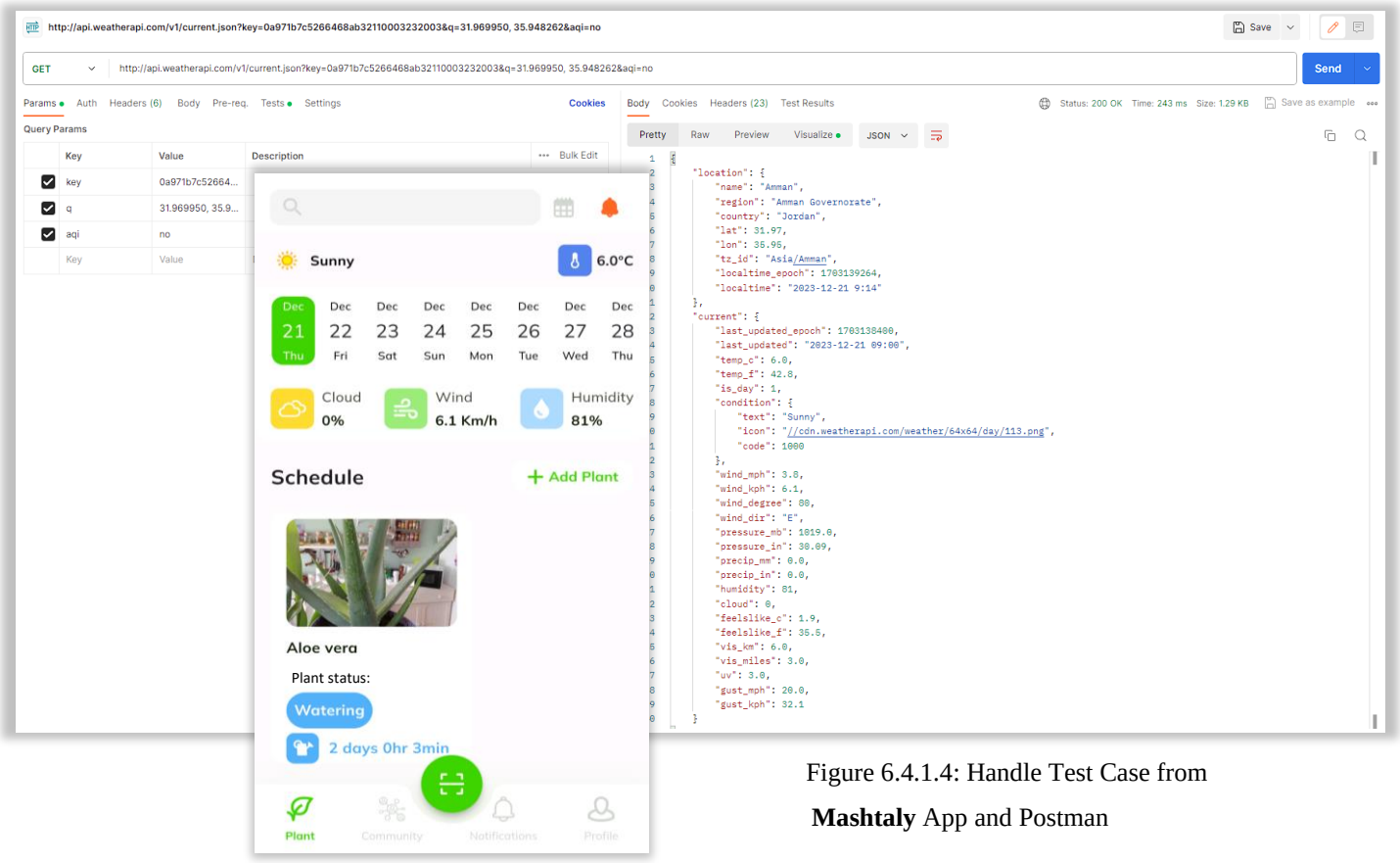


Figure 6.4.1.4: Handle Test Case from **Mashtaly** App and Postman

- **Publish an Article**

  The following test cases aim to validate and enhance the publish an article functionality by thoroughly testing all potential input scenarios.

Test Case	Input	Expected Output	Actual Output
Verify successful creation and publication of an article.	Article Photo: upload at least 3 picture Title: valid title Content: valid content	Show message the post has been uploaded.	Show message ‘Post submitted! Admin review in progress’.
Incomplete content.	Article Photo: upload at least 3 picture Title: valid title Content: incomplete	Show error message.	Show message ‘Please enter content for article.’.

Table 6.4.1.5: Test Case for Publish an Article

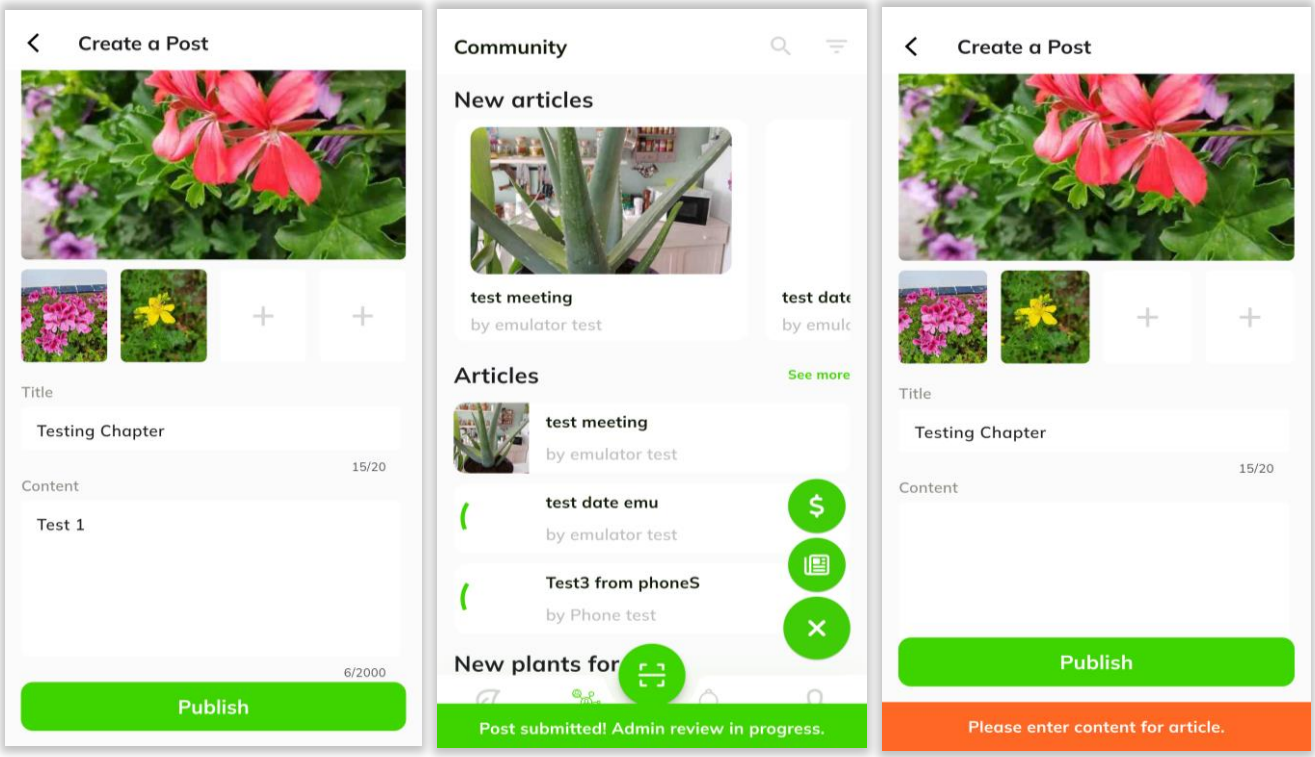


Figure 6.4.1.5: Handle Test Case from **Mashtaly** App

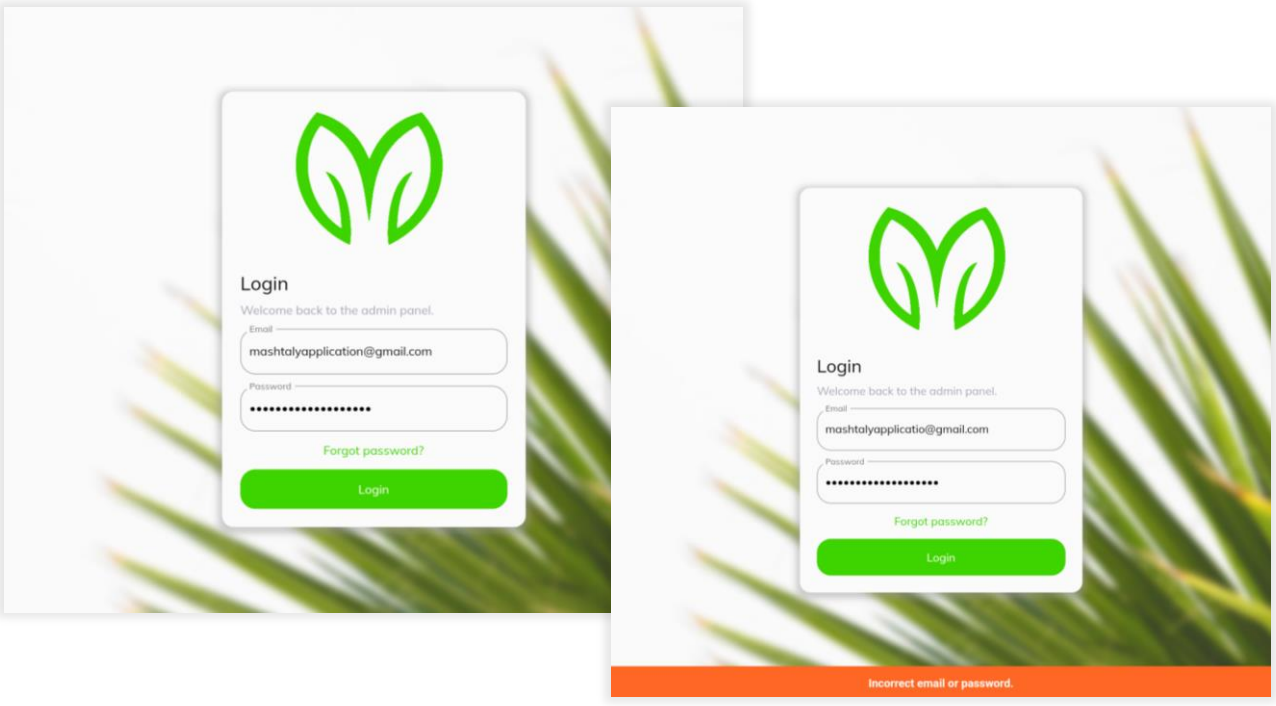
- **Mashtaly Dashboard**

  - **Login**

The following test cases aim to validate and enhance the login functionality by thoroughly testing all potential input scenarios.

Test Case	Input	Expected Output	Actual Output
Valid credentials.	Email: valid email Password: valid password	Enter to main screen.	Enter the main screen of the application.
Invalid credentials.	Email: invalid email Password: valid password	Show error message.	Show message 'Incorrect email or password.'.
Invalid credentials.	Email: valid email Password: invalid password	Show error message.	Show message 'Incorrect email or password.'.
After multiple failed login attempts.	Email: valid email Password: invalid password	Show error message.	Show message 'Access to this account has been temporarily disabled due to many failed login attempts.'.

Table 6.4.1.6: Test Case for Login



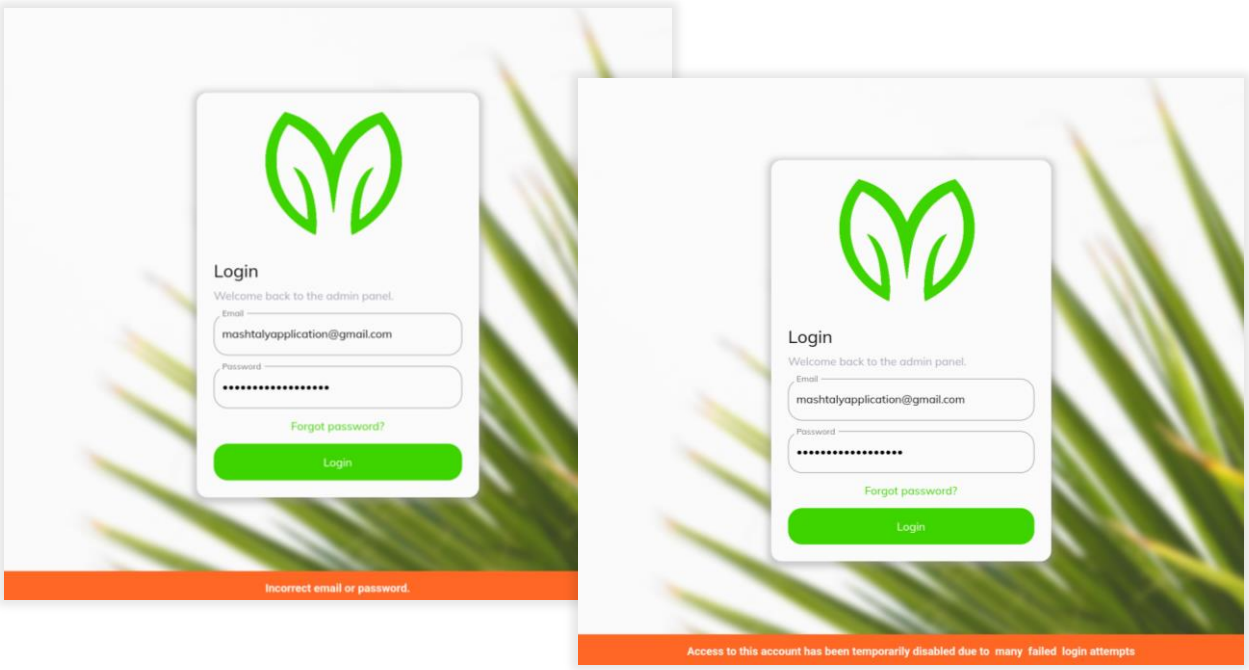


Figure 6.4.1.6: Handle Test Case from Mashtaly Dashboard

- **Approve posts**

The following test cases aim to validate and enhance the approve posts functionality by thoroughly testing accept and reject scenarios.

Test Case	Input	Expected Output	Actual Output
Test Approve post.	Post status: Accepted	Send notification to user and publish his post.	Enable notifications to receive timely updates. Instantly view new posts in the main app's article screen.
Test Approve post.	Post status: Rejected	Send notification to user and do not share his post.	Enable notifications to receive timely updates.

Table 6.4.1.7: Test Case for Approve Post

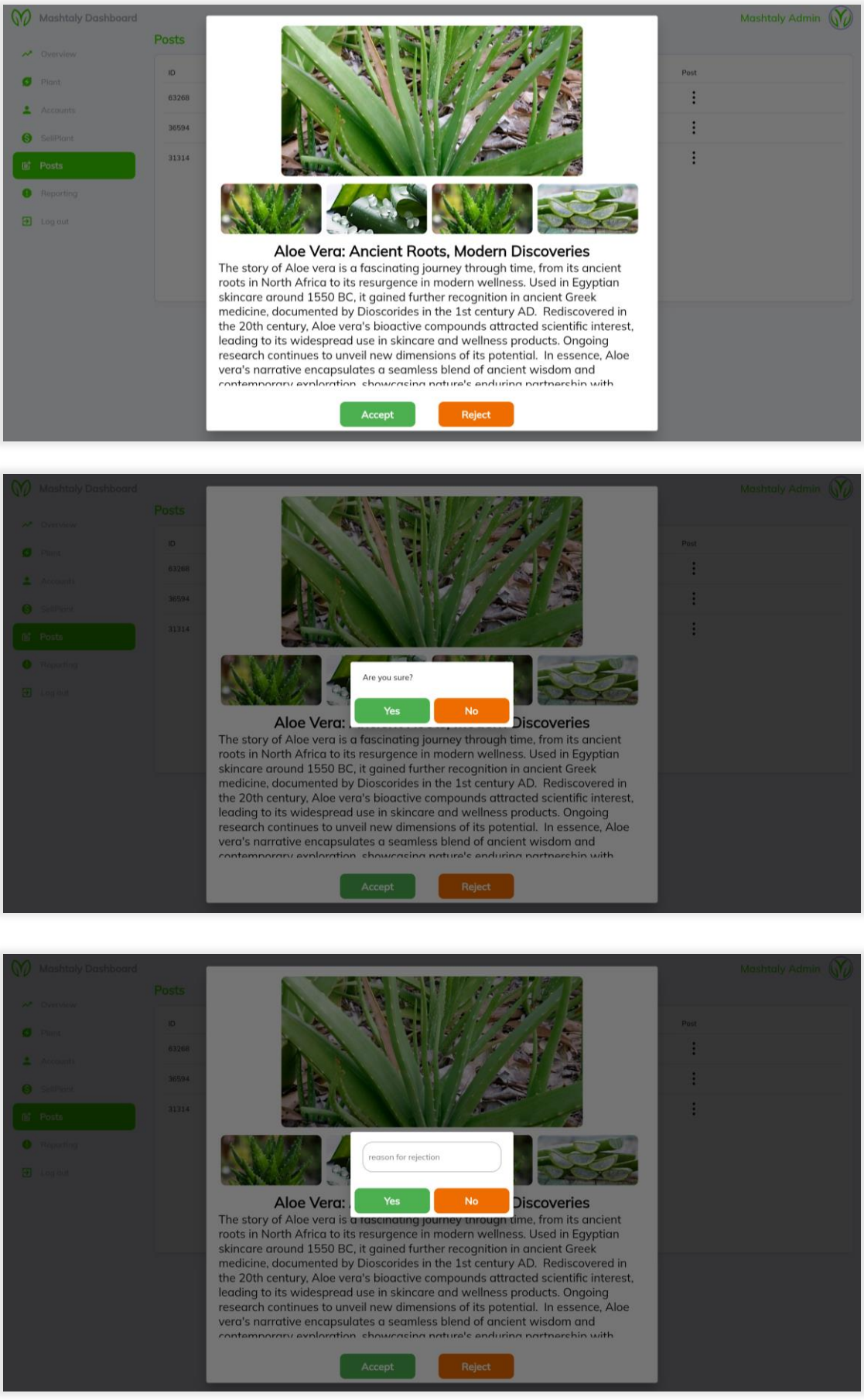


Figure 6.4.1.7: Handle Test Case from Mashtaly Dashboard

6.4.2 Integration Testing (Black-box)

The objective of Integration Testing is to ensure the seamless collaboration of different components within the **Mashtaly** application. This phase focuses on validating the interaction and data flow between integrated modules.

- **API Integration**

  Involves connecting and interacting with external to enhance the functionality of a software application. In the context of **Mashtaly**, which involves plant-related functionalities, API Integration can include connecting with external services to fetch data related to weather, plant databases, or image recognition.

API Integration in Mashtaly:

- **Weather API:**

  The application uses weatherapi to fetch real-time weather data based on the user's exact location, ensuring constant alignment with current meteorological conditions. Automatic updates keep users promptly informed of any weather changes, enhancing the user experience and showcasing the application's commitment to delivering accurate, timely, and location-specific weather forecasts.

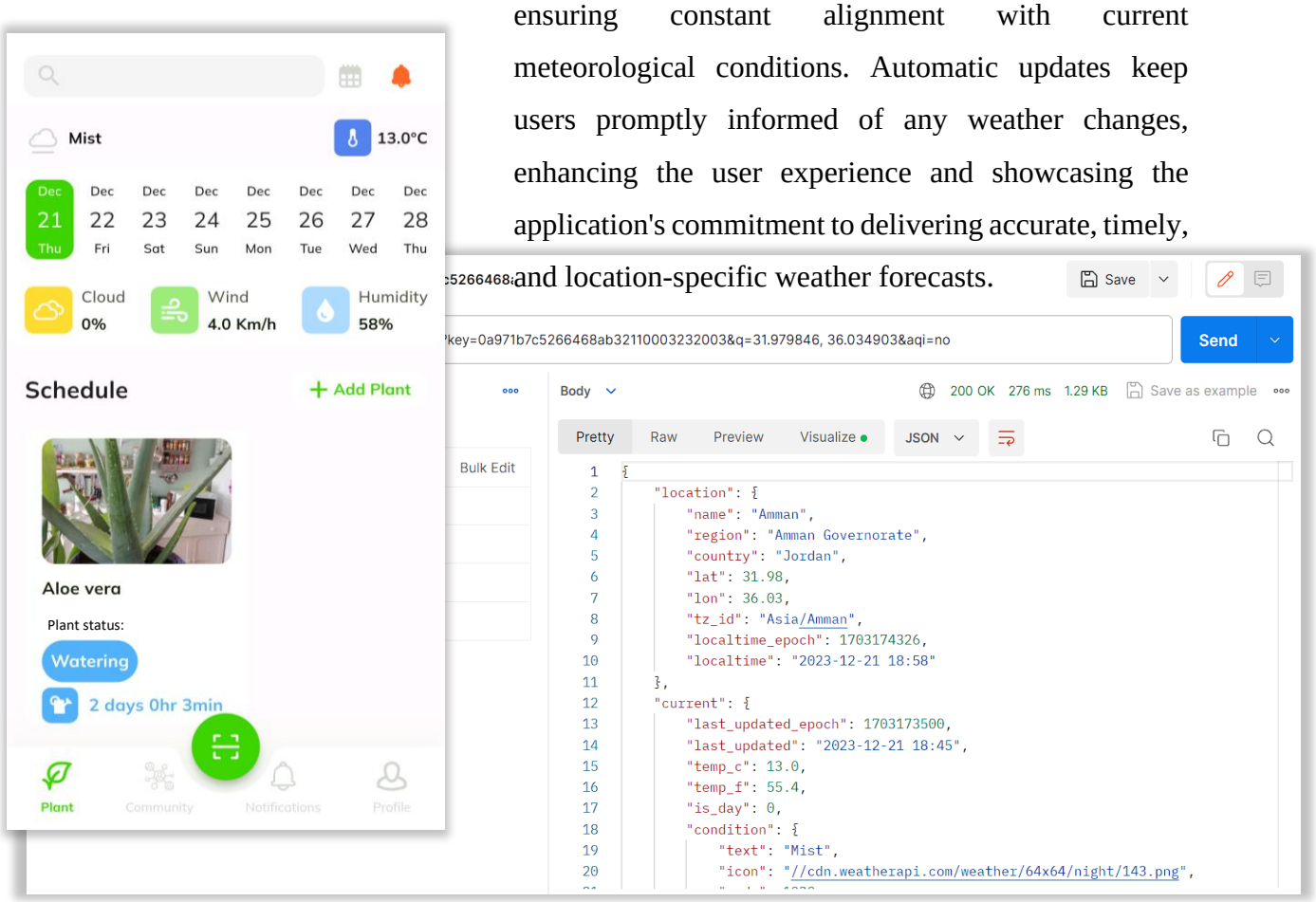


Figure 6.4.2.1: Handle Test from Mashtaly App and postman

- **Plant Database API Integration with Image Recognition API:**

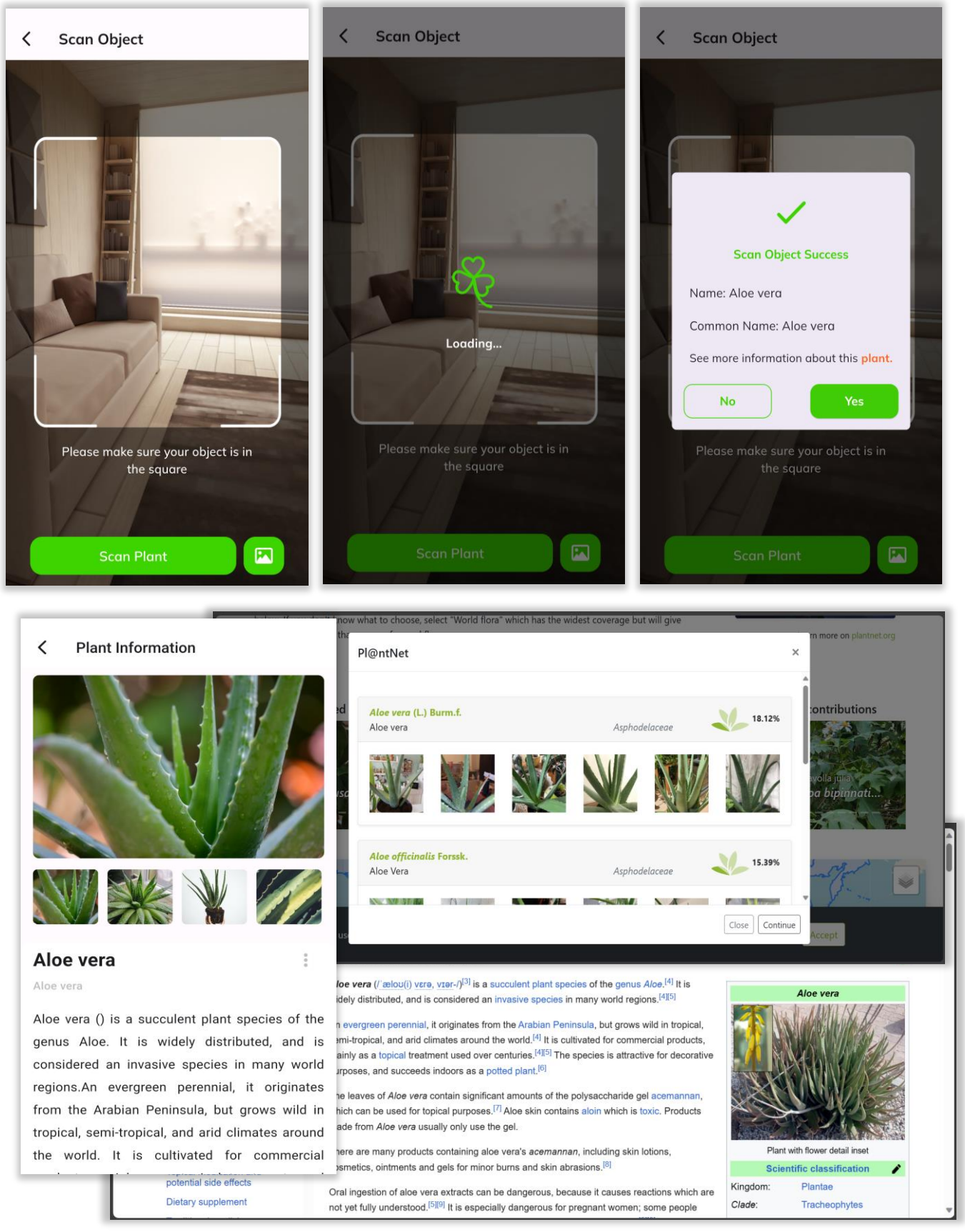


Figure 6.4.2.2: Handle Test from **Mashtaly** App and Pl@ntNet API and WIKIPEDIA

6.5 Test Environments

In the dynamic landscape of mobile applications, thorough testing across diverse environments is paramount to ensuring the robustness and reliability of the **Mashtaly** application. The Test Environments section encompasses a comprehensive strategy that addresses the variability in devices, operating systems, and network conditions. This multifaceted approach is designed to identify potential issues, enhance user experience, and guarantee seamless functionality across different scenarios.

6.5.1 Devices

To guarantee widespread accessibility, **Mashtaly** undergoes testing across an array of Android and iOS devices. This rigorous device testing protocol aims to verify compatibility, responsiveness, and visual fidelity. By ensuring that the application performs optimally on various screen sizes, resolutions, and hardware specifications, **Mashtaly** can confidently cater to a diverse user base.

6.5.2 Operating Systems

Diversity in mobile operating systems is a hallmark of the contemporary digital landscape. **Mashtaly** embraces this diversity by undergoing testing across different versions of both Android and iOS platforms. This approach acknowledges the nuanced variations and updates present in each operating system, ensuring that **Mashtaly** remains adaptable to evolving OS-specific features and requirements.

6.5.3 Network Conditions

The user experience within **Mashtaly** is intricately linked to network performance. Therefore, the testing strategy incorporates the simulation of various network conditions, including 3G, 4G, and Wi-Fi environments. By subjecting the application to different network speeds and types, the testing process evaluates its resilience under diverse connectivity scenarios. This meticulous assessment aims to identify potential bottlenecks, optimize data transfer, and enhance the overall responsiveness of **Mashtaly** across varying network conditions.

CHAPTER 7: CONCLUSION AND RESULTS

7.1 Summary of Accomplished Project

In today's society, technology and the internet have become integral components of our daily lives, enabling us to perform a plethora of tasks from the comfort of our own homes. In light of this, the **Mashtaly** was developed to ensure that our house plants receive the best possible care, there by securing a promising future for them. The platform achieves this by providing breeders with specific details about their plants and ensuring that they receive the optimal care they require.

Additionally, we strive to minimize water waste by utilizing water in the most efficient way possible. Our platform incorporates a range of features designed to improve the process of plant breeding, including a soil moisture sensor that sends notifications to plant breeder, alerting them when their plants require watering. Moreover, **Mashtaly** camera facilitates plant identification, providing users with detailed information about their plants.

7.2 Future Work

As we look towards the future of **Mashtaly**, there are several goals that we aim to achieve, including but not limited to:

- We will expand our services beyond house plants to include plant nurseries.
- Upgrading our soil moisture sensor to enable automatic watering of plants when required.
- Publish **Mashtaly** in different stores like Google Play, Huawei Gallery, and App Store.
- Enhancing the quality of our services to deliver an even better experience to our users.
- Developing the capability to detect plant diseases through our camera technology.

By striving towards these objectives, we aim to continually improve and evolve our platform, providing plant breeders with the most advanced and comprehensive tools for plant care and breeding.

REFERENCES

[1]: Planta - Care for Your Plants

https://Cutt.Us/GooglePlay_Planta

[2]: Plant Parent

https://Cutt.Us/GooglePlay_Plant_Parent

[3]: Blossom

https://Cutt.Us/GooglePlay_Blossom

[4]: Pl@ntNet

https://Plantnet.Org/En/

[5]: weatherapi

https://Www.Weatherapi.Com/

[6]: WiFiManager

https://Github.Com/Tzapu/Wifimanager

[7]: Adobe Xd

https://Helpx.Adobe.Com/Xd/Help/Adobe-Xd-Overview.Html

[8]: Flutter

https://Flutter.Dev/

[9]: Dart

https://Dart.Dev/

[10]: Visual Studio Code (VS Code)

https://Code.Visualstudio.Com/Docs

[11]: Firebase

https://Firebase.Google.Com/

[12]: Pub.Dev

https://Pub.Dev/

[13]: MVVM

https://Cutt.Us/Model-View-ViewModel

[14]: Soil Moisture Sensor

https://Cutt.Us/Soil_moisture_sensor

[15]: Arduino IDE

https://Cutt.Us/Arduino_IDE