

---

# **Software Requirements Specification**

**for**

# **Leaflings**

**Version 5.0 approved**

**Prepared by João Pedro Santos**

**Reviewed by:**

**Hugo Guimarães**

**Pedro Pinho**

**Sónia Oliveira**

**ESTG**

**Grupo 10**

**26-09-2024**

# Table of Contents

<b>1. Introduction.....</b>	<b>1</b>
1.1 Purpose.....	1
1.2 System Proposal.....	1
1.3 Product Scope.....	3
1.4 References.....	4
1.5 Product Perspective.....	4
1.6 Operating Environment.....	4
1.7 Design and Implementation Constraints .....	5
1.8 Assumptions and Dependencies.....	5
<b>2. External Interface Requirements .....</b>	<b>6</b>
2.1 User Interfaces.....	6
2.1.1 User Interfaces – Mobile Version – First Screens (Login and Register) .....	6
2.1.2 User Interfaces – Mobile Version – Homepage.....	7
2.1.3 User Interfaces – Mobile Version – User Homepage.....	8
2.1.4 User Interfaces – Mobile Version – Edit Account Screens .....	9
2.1.5 User Interfaces – Web Browser – First Screens (Login and Register).....	10
2.1.6 User Interfaces – Web Browser – Homepage.....	11
2.1.7 User Interfaces – Web Browser – User Homepage.....	12
<b>3. Use Case Scenarios .....</b>	<b>14</b>
3.1 User Management .....	14
3.2 Plant Database and Matchmaking .....	18
3.3 Administration.....	22
3.4 Other Functional Requirements .....	26
3.5 Product Backlog .....	28
<b>4. Other Nonfunctional Requirements.....</b>	<b>31</b>
4.1 Security Requirements .....	31
4.2 Software Quality Attributes .....	32
<b>5. Diagrams.....</b>	<b>33</b>
5.1 Package Diagram .....	33
5.2 Sequence Diagrams .....	34

5.3	Class Diagram.....	41
5.4	Architecture Diagram.....	42
<b>6.</b>	<b>Database Design and Normalization.....</b>	<b>43</b>
6.1	ER Diagram.....	43
6.2	Identify Entities and relationships.....	44
6.3	Assign keys to entities .....	46
6.4	Solve redundancies .....	46
6.5	Conceptual Model.....	47
6.6	Data Dictionary.....	48
6.7	Logical Model.....	51
<b>7.</b>	<b>Use Case.....</b>	<b>53</b>
7.1	User Management .....	53
7.2	Administration.....	54
7.3	Plant Matchmaking.....	55
<b>8.</b>	<b>Evidence of Implementation .....</b>	<b>56</b>
8.1	Proof of netcore.....	56
8.2	Proof of tokens for authentication .....	59
8.3	Proof of EF .....	60
<b>9.</b>	<b>Tests.....</b>	<b>62</b>
9.1	Unit Tests.....	62
9.2	Postman Tests.....	66
<b>10.</b>	<b>Proof of CI/CD.....</b>	<b>67</b>
10.1	Introduction to CI/CD .....	67
10.2	CI/CD Tools .....	67
<b>11.</b>	<b>Sprint Backlogs .....</b>	<b>70</b>
11.1	Sprint Backlogs since milestone3 .....	70
<b>12.</b>	<b>Online Resources.....</b>	<b>76</b>
12.1	API.....	76
12.2	Frontend for Webbrowser.....	76
12.3	Backend for Webbrowser .....	77
<b>13.</b>	<b>Conclusion.....</b>	<b>78</b>

## Figure Index

Figure 1 - Revision History .....	7
Figure 2- Login/register screens    Register Screen    Login Screen .....	6
Figure 3- Homepage Screen .....	7
Figure 4- User Homepage Screens    Check Plant Screen    My Plant Diary .....	8
Figure 5 - Edit Account and Preferences Screen .....	9
Figure 6 - Browser LandingPage .....	10
Figure 7 - Register Screen .....	10
Figure 8 - Login Screen.....	10
Figure 9 - Homepage.....	11
Figure 10 - Card presentation .....	11
Figure 11 - profile homepage .....	12
Figure 12- check a user owned plant.....	12
Figure 13 - Change Preferences Page .....	13
Figure 14 - Package Diagram .....	33
Figure 15- Register Sequence Diagram .....	34
Figure 16 - Fill Profile Sequence Diagram.....	35
Figure 17 - Edit User Profile Sequence Diagram .....	36
Figure 18 - Get Matching Plant Sequence Diagram .....	37
Figure 19 - Add New Plant Sequence Diagram.....	38
Figure 20 - Remove Plant Sequence Diagram.....	39
Figure 21 - View User's Plants Sequence Diagram .....	40
Figure 22- Class Diagram.....	41
Figure 23 - Architecture Diagram.....	42
Figure 24 - First ER Diagram .....	43
Figure 25- conceptual model .....	47
Figure 26 - Data Dictionary.....	50
Figure 27 - Logical Diagram, foreign keys are represented with the letter U .....	52
Figure 28 - Use Case Register and authentication .....	53
Figure 29 - use case administration.....	54
Figure 30 - use case Plant Matchmaking.....	55
Figure 31 -. NetCore dependency injection.....	56
Figure 32 - builder object in program.cs .....	57
Figure 33 - app object in program.cs .....	58
Figure 34 - proof of JWT tokens for authentication .....	59
Figure 35 – JWT key.....	59
Figure 36 - token generator .....	59
Figure 37 - adding Database Context.....	60
Figure 38 - mapping tables .....	60
Figure 39 - mapping entites .....	61
Figure 40 - webApi context for tests.....	63
Figure 41 - example of service testing class.....	64
Figure 42 - snippet of excel file for test cases .....	64
Figure 43 - Testing Controllers file.....	65
Figure 44 - snippet of postman requests.....	66

Figure 45 - dockerfile.....	68
Figure 46 - docker creation.....	68
Figure 47 - docker running .....	68
Figure 48 - example of pipeline passed.....	69
Figure 49- pipeline configuration .....	69
Figure 50 - sprint backlog 19/10-26/11 .....	70
Figure 51 - sprint backlog 26/10-02/11 .....	71
Figure 52- sprint backlog 02/11-09/11.....	72
Figure 53- sprint backlog 17/11-26/11.....	73
Figure 54-sprint backlog 26/11-02/12.....	74

## Table Index

Table 1- Register User Use Case .....	14
Table 2 - Func Requirement Login And Register.....	14
Table 3 - Use Case Scenario Login.....	15
Table 4- Func Requirement Login and Register.....	15
Table 5 - Use Case Scenario Create User Profile .....	16
Table 6 - Func Requirement Profile Creation .....	16
Table 7- Use Case Scenario Alter User Profile .....	17
Table 8 - Func Requirement Profile Alteration .....	17
Table 9- Use Case Scenario Access Plant Directory .....	18
Table 10 - Func Requirement Plant Directory Access.....	18
Table 11 - Func Requirement Plant Info Display .....	19
Table 12 - Func Requirement Plant Info Display .....	19
Table 13 - Func Requirement Searching And Filtering .....	20
Table 14 - Use Case Scenario Match Plants To Users.....	20
Table 15 - Func Requirement Plant Matchmaking .....	21
Table 16 - Use Case Scenario Adding Plants to Users Favorites .....	21
Table 17 - Func Requirement Add Plant To Garden .....	22
Table 18 - Use Case Scenario Remove User .....	22
Table 19 - Func Requirement Remove User .....	23
Table 20 - Use Case Scenario Access User Information .....	23
Table 21 - Func Requirement User Information Access.....	24
Table 22 - Use Case Scenario Plant Creation.....	24
Table 23 - Func Requirement Plant Creation .....	25
Table 24 - Use Case Scenario Edit Plant Information .....	25
Table 25 - Func Requirement Plant Edition .....	26
Table 26 – Func Requirement Reminder System .....	26
Table 27 - Func Requirement Streaks .....	27
Table 28 - Non Func Requirement Password Encryption.....	31
Table 29 - Non Func Requirement Token Authentication .....	31
Table 30 - Non Func Requirement Ease of Use .....	32
Table 31 - Identify Entities and Relationships .....	44
Table 32 - Multiplicity of relationships.....	45

## Revision History

Name	Date	Reason For Changes	Version
João Santos	03/10/2024	Corrections to	<b>1.1</b>
João Santos	16/10/2024	Adding Use Cases	<b>1.2</b>
João Santos	17/10/2024	Added Diagrams and Requirements	<b>2.0</b>
João Santos	22/10/2024	Introduce figure index, introduced table index, introduced new ER diagram, introduced conceptual drawing and normalization process, changed Scope, changed Class Diagram, changed use cases, introduced product backlog to the report. Added new class Diagram.	<b>3.0</b>
João Santos	6/11/2024	Introduced Evidence of using .netcore.; Introduced Evidence of using tokens for authentication; Introduced Evidence of using EF ( code-first ); Introduced proof of unit tests, and postman tests; Introduced Evidence of CI/CD; Introduced Sprint Backlogs; Introduced current sprint backlog.	<b>4.0</b>
João Santos	5/12/2024	Introduced more sprint backlogs; Introduced non implemented backlog; Introduced online sources and credentials; Introduced conclusion.	<b>5.0</b>

Figure 1 - Revision History

# 1. Introduction

## 1.1 Purpose

This document is dedicated to the 1.0 release of the application called **Leaflings**, by Group 10, for the course unit of software development lab at Escola superior de tecnologia e gestão (ESTG).

This document will go over the main functionalities to be implemented in this release, that can be consulted at “**1.5 Product Scope**”.

Group 10 is formed by the following students:

- . **João Pedro Santos** - 8220256
- . **Hugo Ricardo Guimarães** - 8220337
- . **Sónia Raquel Degtyareva** - 8220114
- . **Pedro Santos Pinho** - 8220307

## 1.2 System Proposal

### **Overview**

*The system our developing team proposes in this document aims to match users to houseplants best suited for them, having in mind certain characteristics, ranging from available time for taking care of the plants, to conditions such as natural light. The system aims to tackle the problem a lot of beginners face when it comes to starting to care for plants, which is a lack of experience leading to having their plants die, wasting their money and time on subpar care for houseplants. The system's main goal is to ease users into the world of caring for plants, aiming to create a higher consciousness for wildlife and nature in general, while also helping said wildlife by preventing users from killing plants they thought would fit them, and preventing said users from also wasting their money and resources.*

### **System Features**

*This system will primarily focus on easing the journey of a beginner plant enthusiast into the world of caring for plants and gardening. as such its core functionalities cover:*

- . **User profile:** a customizable user profile, where the user will be able to input their living conditions, fields such as “available space “, humidity, natural lighting, time available for the plants. It would also provide a dedicated space for plants that the user currently has selected and is assumed to be taking care of in their house, where they'll be able to track their progress across time.
- . **Matching Plants to Users:** Based on the user's parameters, the system will provide the user with a list of potential matches, from which the user will be able to select the ones they desire to try and cultivate. This is done with the intent of matching houseplants to their favored environments, raising the likelihood of said plants thriving.



. **Plant Almanac:** The system will provide a sort of plant almanac, accessible to users through a graphic interface, containing all plants available in the database. Through navigating this almanac, users will be able to filter, select and analyze the data regarding each plant, which should include a general description of the plants, one or multiple pictures, general tips on how to care for it, and its necessities.

. **Social Aspect** The system will provide a form of community engagement, through the implementation of public streaks (tracking how often a user cares for their plants and how well), leaderboards (tracking the number of plants a user successfully takes care of), and a graveyard feature where all the plants a user has not taken care well of will be recorded at.

. **Streaks** – much like Duolingo, the app will have a notification system that reminds the user to water their plants, and will keep track of how many days **in a row** a user has successfully done so.

. **XP system and Leaderboards** – Users will be pushed to compete based on who maintains their plants the best, or who maintains the most plants.

## **Target Audience**

We aim to build our system's user base targeting people that fit the following descriptions:

. **Beginners:** People who are looking to get into gardening or add some green to their house life but lack the experience and confidence to do so alone.

. **Plant Experts:** people with more experience and comfortable with houseplants, possibly looking for a more exotic experience.

. **Plant Shops:** Potential partners with our app, there is the possibility of suggesting shops to our user base depending on their location.

## **Conclusion**

*Our app aims to revolutionize how people interact with the prospect of caring for plants, through personalized plant recommendations and easy access to information about their plants.*

*The initial development phase will focus on creating the foundational components of the application, with the potential for future enhancements that could include a larger plant database.*

## 1.3 Product Scope

### Scope Description

The team is committed to implementing the platform in two distinct versions, a website and a mobile application.

The features to be developed in this initial phase are as such:

- **Customizable user profiles**, in which the user will be able to describe their home environment, available time for plant caretaking, and other preferences;
- **Create, view, and edit** plants, tasks, and ads entries by an admin;
- **An extensive dictionary of plants**, complete with care guides for each entry;
- **An algorithm which will match a user to suitable plants**, through the analysis of the user profile, matching it with the needs and characteristics of the available plants at our database, such as luminosity, difficulty of care, water needs and such;
- **The option to select plants and associate them to a user profile**, in order to track the process of their plants.
- **A plant diary**, when a user selects a plant and has it associated to their profile, each plant has a diary which consists of a series of entry logs written by a user.
- **Paid and Free versions**, when a user pays for their subscription, they're allowed to have more than a certain number of plants and have ads removed from the platform.
- **An ad system**, where an admin can create Ads that are shown to free users.
- **A Warning system**, in order to warn users of dangerous weather events through the use of notifications.

**All other features described in system proposal are therefore out of scope, and not to be developed in the following 3 months for the course unit of software development lab.**

## 1.4 References

**Robert C. Martin Series, Clean Architecture A craftsman's Guide to Software Structure and Design, 1<sup>st</sup> edition, Prentice Hall.**

**Mark Richards, Fundamentals of Software Architecture, 1<sup>st</sup> edition, O'Reilly.**

**Philip A. Laplante, Requirements Engineering for Software and Systems, 3<sup>rd</sup> edition, CRC Press.**

## 1.5 Product Perspective

Our group noticed that for many activities, like learning a language, hiking, and others, there are plenty of apps that are aimed at helping the process and tracking said process of its user base.

However, we noticed that we had never heard of such an app for when it comes to caring for plants, and after searching, although we noticed there are tracking apps for this effect, none of them really "gamifies" the process, in aims of making it more engaging and fun.

The above description summarizes where the idea for the application came from; With further discussion the group decided it should be a self-contained product with the potential for future expansion through the addition of new features, much alike the application used as inspiration for the project "Duolingo ". It is intended as an app that the userbase would interact with on a day-to-day basis, where people can get educated, and share their experiences with the gardening world.

We found it to be a fun to work in project that would, at the same time, bring a positive weight to the social landscape, through engaging people in a community revolving around plants, making it so there would be more places around filled with greens, investing people into caring more heavily for plant life, and doing all this while using the game aspect of the system in order to drive engagement instead of lecturing the general population.

## 1.6 Operating Environment

The **Leaflings** application will be available in a web app format, ensuring accessibility on various operating environments, including Windows, Linux, and macOS. It is designed with the following web browsers in mind:

- . **Google Chrome (latest two versions)**
- . **Mozilla Firefox (latest two versions)**

Other than the webapp format, the group will be developing a mobile version of the system destined to the android platforms.

## 1.7 Design and Implementation Constraints

Our project will be subject to several constraints, which might limit our options:

. **Time Constraints:** The team only has **3 months** to complete the project, necessitating a focus on delivering a Minimum Viable Product (MVP) with core functionalities.

. **Budget Limitations:** Due to a lack of funds, we will not utilize any paid interfaces, libraries, or web services.

We will be using **react** for the webapp version and **Kotlin** for the mobile version so anything outside of the capability of these frameworks will be beyond the scope our project, it may also cause the application to not be compatible with old browser versions, or older android and iOS versions.

## 1.8 Assumptions and Dependencies

The development process of the Leafings application will be affected by and based on multiple *assumptions and dependencies which influence the flow of development, depending on how accurate these turn out to be.*

*The assumptions are as follows:*

- *It is assumed that the future users of the platform have sufficient understanding of technology (namely, web and mobile applications) in order to navigate the system without the need of extensive training beyond short guides and/or tutorials;*
- *We assume that users will have access to hardware and software environments that allow for the application to run smoothly, such as OS compatibility, system capability and stable network connection;*
- *It is assumed that the initial database (to be implemented upon release) will be sufficiently comprehensive, providing users with information most pertinent to them, which is to say, include the most common houseplants recommended for ownership;*
- *It is assumed that all members of the development team will be available for meetings, willing to collaborate, and aid in decision making throughout the progress of the project, as well as possess all the technical skills to effectively implement the platform.*

*The dependencies are as such:*

- *The choice of React and React Native for the development of the project, demands that these frameworks and the tools necessary for their support are available and stable during the development period and beyond;*
- *Any external API's or resources used will be presumed to remain available for the foreseeable future, in order to maintain all functionality during and post development and deployment;*

- *As GitLab will be used for version control and collaboration, these elements rely heavily on its availability for all team members;*
- *Lastly, the project will be heavily influenced by user feedback during testing and prototyping, hence it will be dependent on the users who will be submitting their thoughts and any issues encountered. Any delays encountered during these phases will impact the development.*

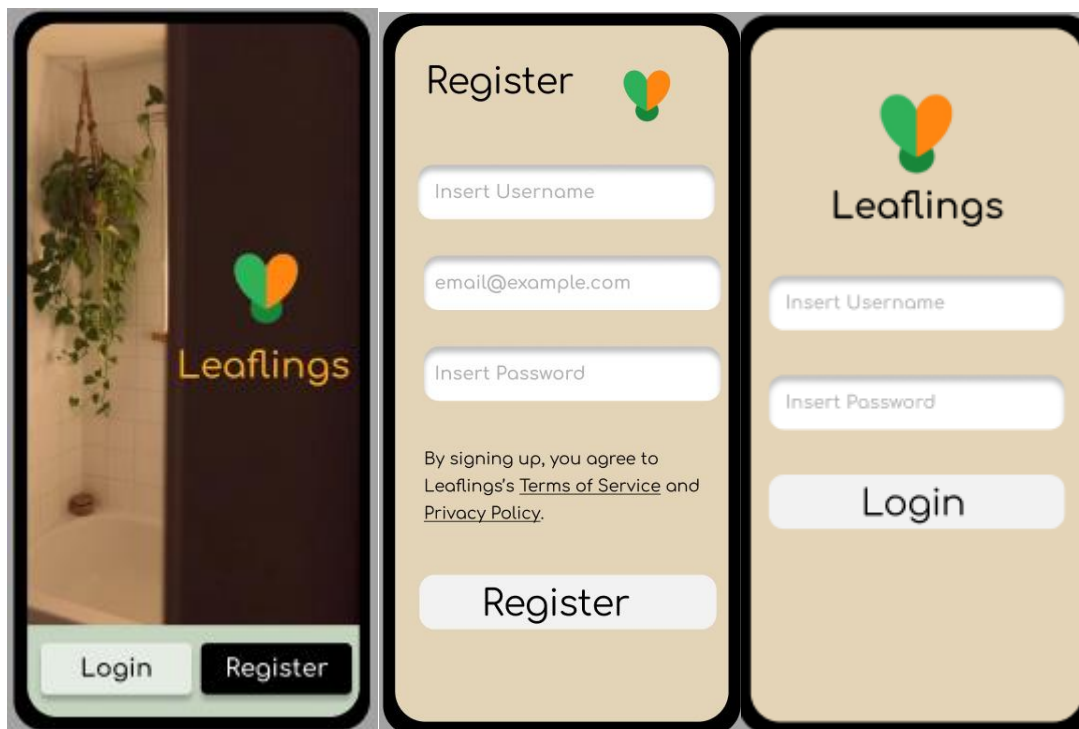
## **2. External Interface Requirements**

### **2.1 User Interfaces**

#### **2.1.1 User Interfaces – Mobile Version – First Screens (Login and Register)**

The leaflings mobile user interface aims to be simple and intuitive, in terms of it's design the team is aiming for something that looks simple, easy to use and comprehensive. The reasons for this are multi-faceted, but to point out the most important reasons, we have the development time constraints, the fact that it will be a cross-platform system, and we also have the fact that app design nowadays is tending towards becoming more and more simplistic.

*These screens play an important role in the system, as they're meant to provide the user with a preliminary experience to the user of what the aesthetic of the system looks like.*



*Figure 2- Login/register screens*

*Register Screen*

*Login Screen*

## 2.1.2 User Interfaces – Mobile Version – Homepage



The leafings homepage serves as the central page for a user to easily navigate through the app's key features. It was designed to combine both simplicity and the navigation of the app.

**Navigation:** At the top of the screen, users have clickables to **access to their profile and plant collection**, keeping personalized content readily available.

**Almanac Section:** The almanac provides an overview of the plants in the database, displaying each plant's name alongside visual clues that indicate important metrics such as water levels, sunlight exposure, and experience necessary. This layout allows users to discover new plants every time they enter the app, while keeping the access to personalized content easily accessible. The almanac is supposed to represent an endless scrollable, which will load more and more plant entries the more a user scrolls.

**Plant Cards:** Each Plant is represented inside a “card”. Each card is composed of a clear image, picked by admins of the platform, a name, and the bars that indicate the plant's needs in a quick fashion.

Figure 3- Homepage Screen

### 2.1.3 User Interfaces – Mobile Version – User Homepage

The User Homepage is what a user can access by clicking the profile option.

In the profile screen, the user is presented with a section dedicated to the user's preferences (where a user may change their preferences which include, the sun exposure they get for their plants, the amount of time they have to care for plants and the experience they have). They are also presented with a scrollable element where the user's plants will be presented in "card" elements as well when a user clicks, they'll be presented with a new screen, with the information for their plant, an editable text wall representing a plant diary, and a button that allows a user to remove the plant from their personal plants.



Figure 4- User Homepage Screens      Check Plant Screen      My Plant Diary

## **2.1.4 User Interfaces – Mobile Version – Edit Account Screens**

The following screens are accessible by both the “Edit Account” button in the User Homepage and the “My preferences” clickable in the same page.

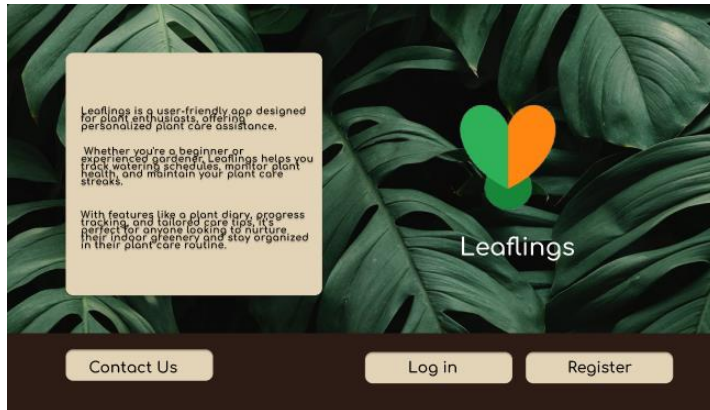
Edit Account allows a user to change their email and password while the “My Preferences” screen allows a user to change their availability settings.

The image displays two side-by-side mobile application screens. The left screen, titled "Edit Account", features four input fields: "Enter email", "confirm email", "Insert Password", and "confirm Password". A "Complete" button is positioned at the bottom. The right screen, titled "My Preferences", includes a "Time to Water" input field, an "Experience with plants" input field, and a "Sun Exposure at home" section with three radio button options: "Low", "Medium", and "High". A "Complete" button is also at the bottom of this screen. Both screens have a light beige background and rounded corners.

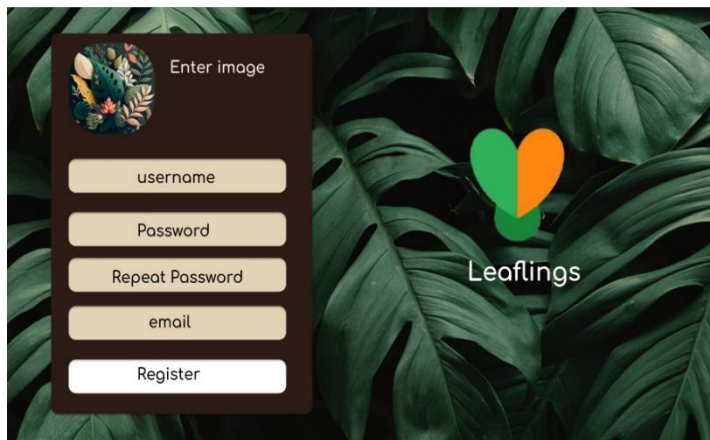
*Figure 5 - Edit Account and Preferences Screen*



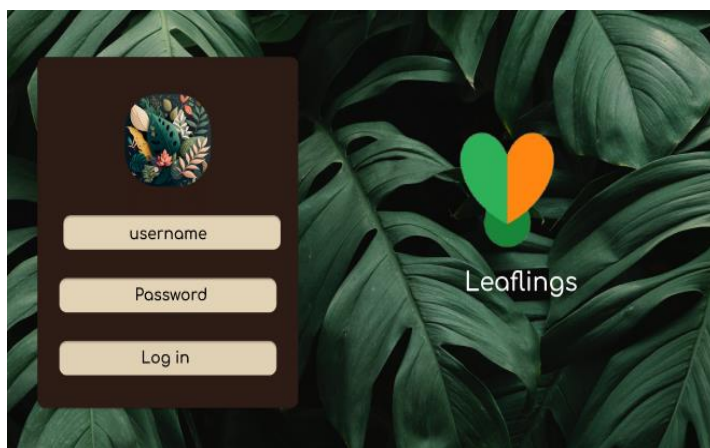
## 2.1.5 User Interfaces – Web Browser – First Screens (Login and Register)



*Figure 6 - Browser LandingPage*



*Figure 7 - Register Screen*



*Figure 8 - Login Screen*

1. LandingPage: This screen presents the entry point of a user into the browser version of our app.

2. Register: This screen presents the form a user has to fill with their information before being allowed to access the platform.

3. Login: This screen presents the user who has already registered with the platform the form they must fill before entering their account.

## 2.1.6 User Interfaces – Web Browser – Homepage

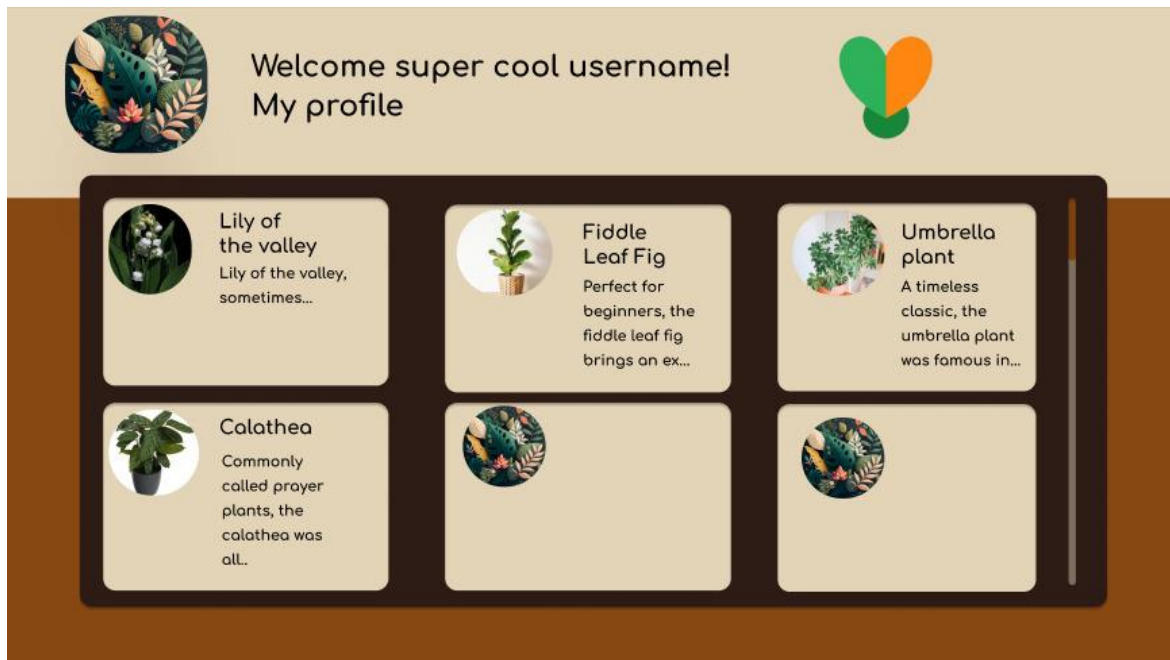


Figure 9 - Homepage

The homepage is designed to be simplistic, and allow the user to be presented with everything without much strain on the eye. The Plant Manual is untitled as it fills most of the screen, it is also designed to be implemented as an infinite scrollable. The clickable element “My Plants” will guide the user to his personal page. The plants are still represented by “cards”, which, when clicked will present a detailed pop up with all the information of a plant.

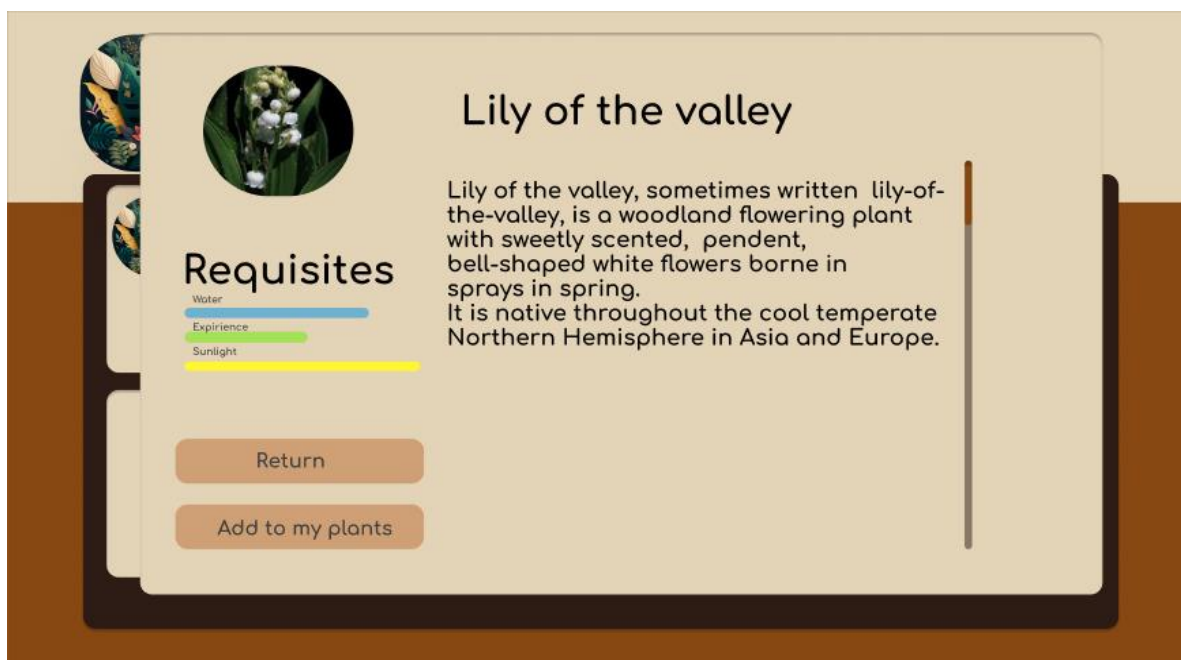


Figure 10 - Card presentation

## 2.1.7 User Interfaces – Web Browser – User Homepage

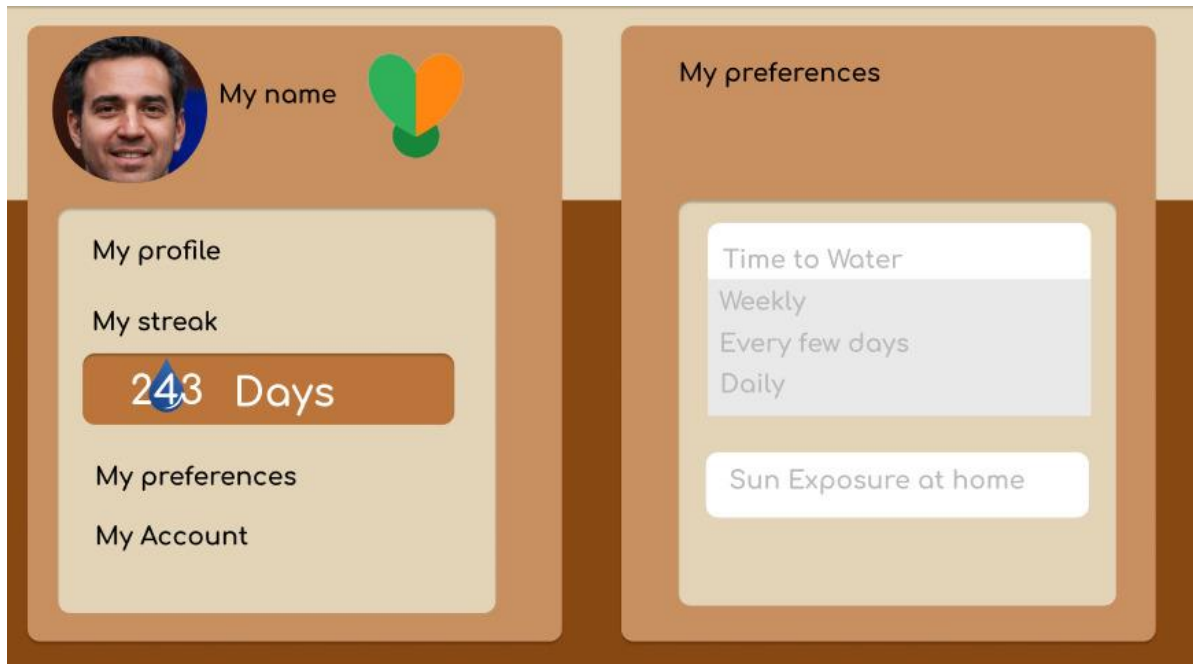


Figure 11 - profile homepage

The user homepage is where a user arrives once they choose the option “My Profile” in the app homepage. It holds their streak (number of days logged into the app count as having watered your plants), and it also holds a second screen card which will change depending on the option chosen by the user. If they select “My Preferences” then the my preferences screen will show up, but if they click their name again, their plants will be displayed again. To go back to the app homepage, they should click the leaflings logo.



Figure 12- check a user owned plant



*Figure 13 - Change Preferences Page*

### 3. Use Case Scenarios

#### 3.1 User Management

##### 3.1.1 Register User

Name	Register User
Actors	User
Pre-Conditions	Users must have access to the platform.
Typical-Flow	A new user will access the platform and choose the <b>Register</b> option, upon which they will be prompted to complete a form with the basic information needed to create an account. After completing the form, the user may then select the <b>Submit</b> button, effectively creating their account.
Alternative-Flow	<b>Improper form data:</b> In case the user fails to properly fill out the form, they will not be able to submit their information, until the form is correctly filled out. <b>Redundancy:</b> If a user's unique identifier (email) coincides with one already present in the database, the user will be notified, and will not be able to create an account.
Extensions	N/A
Post-Conditions	The user will now have access to all of the app's functionalities, and will be able to log in with the credentials they have chosen.

Table 1- Register User Use Case

##### 3.1.1.1 Functional Requirements

Category	User Management
ID	FRUM-01
Name	Login and Registration
Description	Users must be able to register in the application, upon which their data will be securely kept in the database. If a user is registered, they will then be able to log into their account, allowing them to use the platform.
Priority	1 - Essential
State	Proposed
Restrictions	N/A

Table 2 - Func Requirement Login And Register

### 3.1.2 Login

Name	Login
Actors	User, Admin
Pre-Conditions	Users must be registered.
Typical-Flow	A registered user will arrive at the entry screen, and, if they aren't logged in, they can then enter their credentials, upon which they will be granted access to their respective dashboard.
Alternative-Flow	<b>Improper form data:</b> In case the user fails to properly fill out the form, they will not be able to submit their information, until the form is correctly filled out.
Extensions	<b>N/A</b>
Post-Conditions	The user will now have access to all of the app's functionalities, and will be able to log in with the credentials they have chosen.

Table 3 - Use Case Scenario Login

#### 3.1.2.1 Functional Requirements

Category	User Management
ID	FRUM-01
Name	Login and Registration
Description	Users must be able to register in the application, upon which their data will be securely kept in the database. If a user is registered, they will then be able to log into their account, allowing them to use the platform.
Priority	1 - Essential
State	Proposed
Restrictions	N/A

Table 4- Func Requirement Login and Register

### 3.1.3 Create User Profile

Name	Create User Profile
Actors	User
Pre-Conditions	Users must have access to the platform, and must be registered
Typical-Flow	Once a user is registered, they will be able to complete their profile with their caretaking information, which is to say, describe their environment and conditions (by filling out a multiple-choice form). To do this, a user will access their profile, <b>Edit Profile</b> , and fill out all of the relevant fields. Once complete, the user will select <b>Submit</b> in order to persist their data.
Alternative-Flow	<b>Improper form data:</b> In case the user fails to properly fill out the form, they will not be able to submit their information, until the form is correctly filled out.
Extensions	A user will be able to change and update their profile at any given time.
Post-Conditions	The user will now have access to a customized feed of suitable plants for their environment.

Table 5 - Use Case Scenario Create User Profile

#### 3.1.3.1 Functional Requirements

Category	User Management
ID	FRUM-02
Name	Profile Creation
Description	Registered users should be able to create a profile detailing the available environment for plant care (caretaker profile).
Priority	1 - Essential
State	Proposed
Restrictions	Only registered users should have access to this functionality

Table 6 - Func Requirement Profile Creation



### 3.1.4 Alter User Profile

Name	Alter User Profile
Actors	User
Pre-Conditions	Users must have access to the platform, and must be registered
Typical-Flow	Once a user has a profile created, they can then access it, selecting the <b>Edit</b> option, and updating any information they deem necessary.
Alternative-Flow	<b>Improper form data:</b> In case the user fails to properly fill out the form, they will not be able to submit their information, until the form is correctly filled out.
Extensions	<b>N/A</b>
Post-Conditions	The user will now have access to a customized feed of suitable plants for their environment.

Table 7- Use Case Scenario Alter User Profile

#### 3.1.4.1 Functional Requirements

Category	User Management
ID	FRUM-02.1
Name	Profile Alteration
Description	A user with an existing profile should be able to change it at any time, updating their information, as well as their caretaker profile
Priority	1 - Essential
State	Proposed
Restrictions	Only registered users with an existing profile should have access to this functionality.

Table 8 - Func Requirement Profile Alteration



## 3.2 Plant Database and Matchmaking

### 3.2.1 Access Plant Directory

Name	Access Plant Directory
Actors	User, Admin
Pre-Conditions	Users must have access to the platform, and must be registered, and the system and database must be available.
Typical-Flow	User selects the <b>Almanac</b> section of the application.
Alternative-Flow	<b>N/A</b>
Extensions	Once in the <b>Almanac</b> page, a user can use the search bar to look for specific plants. A user can also access a specific plant, and explore the <b>Plant Care Guide</b> or simply read the plant's description
Post-Conditions	The user will now have access to a feed of all the plants available in the database, along with their information.

Table 9- Use Case Scenario Access Plant Directory

#### 3.2.1.1 Functional Requirements

Category	Plant Database and Matchmaking
ID	FRP-01
Name	Plant Directory Access
Description	Users should be able to browse the entirety of the plant catalog, accessing information about each entry.
Priority	1 - Essential
State	Proposed
Restrictions	Only registered users should have access to this functionality

Table 10 - Func Requirement Plant Directory Access

Category	Plant Database and Matchmaking
ID	FRP-01.1
Name	Plant Information Display
Description	Upon selection of a plant present in the catalog, a user should be able to inspect its characteristics.
Priority	1 - Essential
State	Proposed
Restrictions	Only registered users should have access to this functionality

*Table 11 - Func Requirement Plant Info Display*

Category	Plant Database and Matchmaking
ID	FRP-01.2
Name	Caretaking Guide
Description	Associated with each plant, besides their general information and characteristics, there must be a caretaking guide, detailing the steps and precautions to take when caring for this plant.
Priority	1 - Essential
State	Proposed
Restrictions	Only registered users should have access to this functionality, the user's caretaker profile must be completed, and there must be plants in the database capable of thriving according to user's profile.

*Table 12 - Func Requirement Plant Info Display*

Category	Plant Database and Matchmaking
ID	FRP-01.3
Name	Searching and Filtering
Description	Users should be able to look for plants within the database, applying filters in regards to the type of plant.
Priority	2 - High Priority
State	Proposed
Restrictions	Only registered users should have access to this functionality

Table 13 - Func Requirement Searching And Filtering

### 3.2.2 Matching Plants to Users

Name	Matching Plants to Users
Actors	User
Pre-Conditions	Users must have access to the platform, and must be registered, their profile must be complete, and the system and database must be available.
Typical-Flow	User accesses the <b>Almanac</b> and selects the <b>For Me</b> option. The system will compare the user's available environment and the plant's needs, and create a feed containing the plants suitable for the user.
Alternative-Flow	<b>No suitable plants:</b> If for some reason the user's profile does not match with any plants in any capacity, the filter will return no results.
Extensions	Much like in the <b>Almanac</b> page, a user can use the search bar to look for specific plants. A user can also access a specific plant, and explore the <b>Plant Care Guide</b> or simply read the plant's description
Post-Conditions	The user will now have access to a customized feed of suitable plants for their environment.

Table 14 - Use Case Scenario Match Plants To Users

### 3.2.2.1 Functional Requirements

<b>ID</b>	FRP-02
<b>Name</b>	Plant Matchmaking
<b>Description</b>	Through the use of a matchmaking algorithm, and taking into account the user profile and plant characteristics, the program should be able to provide a personalized catalog containing only plants which abide by the user's caretaker profile specifications.
<b>Priority</b>	1 - Essential
<b>State</b>	Proposed
<b>Restrictions</b>	Only registered users should have access to this functionality, the user's caretaker profile must be completed, and there must be plants in the database capable of thriving according to the user's profile.

Table 15 - Func Requirement Plant Matchmaking

### 3.2.3 Adding Plants to User Favorites

<b>Name</b>	Adding Plants to User Favorites
<b>Actors</b>	User
<b>Pre-Conditions</b>	Users must have access to the platform, and must be registered, their profile must be complete, and the system and database must be available.
<b>Typical-Flow</b>	A user accesses a specific plant's page through the <b>Almanac</b> . Then, within the plant page, a user can select the <b>Add</b> option, which will add a plant to the user's favorites ( <b>Garden</b> ), making it easily accessible to check any information.
<b>Alternative-Flow</b>	<b>N/A</b>
<b>Extensions</b>	<b>N/A</b>
<b>Post-Conditions</b>	The plant is added to the user's collection, for easier access, and will allow the user to enable reminders to care for it, if they effectively own this plant.

Table 16 - Use Case Scenario Adding Plants to Users Favorites

### 3.2.3.1 Functional Requirements

Category	Plant Database and Matchmaking
ID	FRP-03
Name	Add Plant to Garden
Description	If the user decides to effectively care for one of the plants present in the catalog, they can then add it to their <b>Garden</b> , which will be a separate collection of plants selected by the user. Here, their information will be readily available.
Priority	1 - Essential
State	Proposed
Restrictions	Only registered users should have access to this functionality, and the user must have selected at least one plant to care for, in order for the Garden to be available.

Table 17 - Func Requirement Add Plant To Garden

## 3.3 Administration

### 3.3.1 Remove User

Name	Remove User
Actors	Admin
Pre-Conditions	User must have administrative permission and be authenticated
Typical-Flow	An admin will access the <b>User Directory</b> , and select the user they wish to remove. They will then be prompted with a confirmation message, and upon answering affirmatively, the user's account will be <b>Deactivated</b> .
Alternative-Flow	<b>Negative confirmation:</b> In case the admin does not confirm the deletion, the action will not be performed.
Extensions	<b>N/A</b>
Post-Conditions	The user's account will no longer be available.

Table 18 - Use Case Scenario Remove User

### 3.3.1.1 Functional Requirements

Category	Administration
ID	FRA-01
Name	User Deletion
Description	The platform admins will be able to remove user accounts from the platform
Priority	5 - Essential
State	Proposed
Restrictions	Only authorized admins have access to this functionality

Table 19 - Func Requirement Remove User

### 3.3.2 Access User Information

Name	Access User Information
Actors	Admin
Pre-Conditions	User must have administrative permission and be authenticated
Typical-Flow	An admin will access the <b>User Directory</b> , upon which they will be able to navigate the list of registered users.
Alternative-Flow	<b>N/A</b>
Extensions	An admin will be able to view a specific user's information.
Post-Conditions	<b>N/A</b>

Table 20 - Use Case Scenario Access User Information

### 3.3.2.1 Functional Requirements

Category	Administration
ID	FRA-02
Name	User Information Access
Description	The platform admins will have access to user information, except passwords
Priority	5 - Essential
State	Proposed
Restrictions	Only authorized admins have access to this functionality

Table 21 - Func Requirement User Information Access

### 3.3.3 Plant Creation

Name	Plant Creation
Actors	Admin
Pre-Conditions	User must have administrative permission and be authenticated
Typical-Flow	An admin will access the <b>Plant Directory</b> , and select the <b>Add new Plant</b> option. They will then be prompted to fill out a form detailing the plant information. Once the form is completed, they will be able to <b>Submit</b> the new plant, upon which it will be persisted in the database.
Alternative-Flow	<b>Improper form data:</b> In case the user fails to properly fill out the form, they will not be able to submit the plant, until the form is correctly filled out. <b>Redundancy:</b> If a plant's unique identifier (scientific name) coincides with one already present in the database, the user will be notified, and will not be able to proceed with the addition.
Extensions	N/A
Post-Conditions	The plant will be available in the directory and ready for operations.

Table 22 - Use Case Scenario Plant Creation

### 3.3.3.1 Functional Requirements

Category	Administration
ID	FRA-03
Name	Plant Creation
Description	The platform admins will be able to add plants to the database, without directly accessing it
Priority	5 - Essential
State	Proposed
Restrictions	Only authorized admins have access to this functionality

Table 23 - Func Requirement Plant Creation

### 3.3.4 Edit Plant Information

Name	Edit Plant Information
Actors	Admin
Pre-Conditions	User must have administrative permission and be authenticated
Typical-Flow	An admin will access the <b>Plant Directory</b> , and select the plant they wish to edit, and edit the fields as they see fit. After this is done, they can <b>Submit</b> the update.
Alternative-Flow	<b>Improper form data:</b> In case the user fails to properly fill out the form, they will not be able to submit the plant, until the form is correctly filled out. <b>Redundancy:</b> If a plant's unique identifier (scientific name) coincides with one already present in the database, the user will be notified, and will not be able to proceed with the edition.
Extensions	<b>N/A</b>
Post-Conditions	The plant's information is now updated and will be used in the platform.

Table 24 - Use Case Scenario Edit Plant Information



### 3.3.4.1 Functional Requirements

Category	Administration
ID	FRA-04
Name	Plant Edition
Description	The platform admins will be able to edit/update plant information
Priority	5 - Essential
State	Proposed
Restrictions	Only authorized admins have access to this functionality

*Table 25 - Func Requirement Plant Edition*

## 3.4 Other Functional Requirements

### 3.4.1 Engagement

Category	Engagement
ID	FRE-01
Name	Reminder System
Description	Given the user's Garden collection, reminders will be sent out, to remind the user of when they need to water their plant, or simply log the day into the application.
Priority	3 - Medium Priority
State	Proposed
Restrictions	Only registered users should have access to this functionality, and the user must have selected at least one plant to care for.

*Table 26 – Func Requirement Reminder System*

Category	Engagement
ID	FRE-02
Name	Streak System
Description	With continuous logging of caretaking into the application, the system will award the user by recording the consecutive entries, incentivizing consistency and responsibility.
Priority	3 - Medium Priority
State	Proposed
Restrictions	Only registered users should have access to this functionality, and the user must have selected at least one plant to care for.

*Table 27 - Func Requirement Streaks*

### **3.5 Product Backlog**

## **LOGIN AND REGISTRY**

User story 1 - AS A DEVELOPER I want the API to effectively save information regarding users (new and old) in the database FOR usage in the system.

User story 2 - AS A USER I want to be able to register and maintain said register in the app SO I CAN use across a long period of time with my personalized history.

User story 3 - AS AN ADMIN I want to be able to authenticate myself IN ORDER TO have access to admin privileges to the platform.

### **Create Profile**

User story 4 - AS A USER I want to be able to create my profile with the info necessary IN ORDER TO find my ideal plant matches;

### **Edit Profile**

User story 5 - AS A USER I want to be able to edit my profile SO I CAN update the info regarding my life conditions and receive better plant suggestions;

User story 6 - AS AN ADMIN I want to be able to remove users TO avoid scenarios like multiple accounts and offensive user names.

## **ACCESS PLANT DIRECTORY (MANUAL)**

User story 7 - AS A USER I want to be able to access the plant manual SO I CAN discover new plants, and check information I find important;

User story 8 - AS A DEV I want to create the plant manual in a visually appealing manner for both the mobile and web browser apps IN ORDER TO make the final product more attractive to the user;

User story 9 - AS AN ADMIN I want to be able to access the plant manual SO I CAN verify information and verify that the information changes I implemented were correctly applied;

## **INFORMATION DISPLAY**

User story 10 – AS AN ADMIN I want to be able to create ads IN ORDER to post them to the platform.

User story 11 - AS A DEV I want to create the plant manual in a manner that it isn't visually overwhelming to the user, for that end I should have an initial display that serves as a plant presentation card, and only when said card is selected should the rest of the information be displayed. I want this IN ORDER TO avoid overwhelming a user visually.

User story 12 - AS AN ADMIN I want the system to force me to fully complete the form to add new plants to the platform IN ORDER TO avoid half registries in the final product plant manual;

## **PLANT CARETAKING GUIDE**

User story 13 - AS A USER I want to be able to access care guides SO I CAN better take care of the plants I've selected as my own;

User story 14 - AS AN ADMIN I want the information for the guides to be easy and intuitive to access SO I CAN better serve the user;

User story 15 - AS AN ADMIN I want to be able to edit the info relative to plants and their guides so I can keep the information in the platform up-to-date;

## **SEARCH AND FILTER**

User story 16 - AS A USER I want to be able to search specific names in the manual SO I CAN be quicker with my searches.

User story 17 - AS AN USER I want to be able to search specific names in the manual SO I CAN be quicker with my searches.

User story 18 - AS A USER I want to be able to filter my plant matches SO I CAN better find plants that I both like, and fit my home environment.

## **PLANT MATCHMAKING**

User story 19 - AS A USER I want to find plants that match my profile's preferences SO I CAN keep them in my house with the help of the guides provided by the app

User story 20 - AS A DEV I want to implement an algorithm that'll match plants to user profile preferences IN ORDER TO provide the main attraction feature of the app

User story 21 - AS A DEV I want to be able to provide the user with the option for a paid subscription SO THAT I CAN make a bigger profit.

User story 22 – AS A USER I want to be able to receive warnings on days where weather might be dangerous to any plants I may have outside.

## **PLANT GARDEN (My Plants section)**

User story 23 - AS A USER I want to be able to save the plants that I find relevant to me, IN ORDER TO have their information available to me at moment's notice.

User story 24 - AS A USER I want to be able to write in my plant diary SO THAT I CAN keep track of the progress I've been making with my plant.

User story 25 - AS A USER I want to have access to multiple plants in my garden SO THAT I can keep multiple plants at home.

## 4. Other Nonfunctional Requirements

### 4.1 Security Requirements

Category	Safety and Security
ID	NFRS - 01
Name	Password Encryption
Description	The user's password will never be directly kept in the database, without first being encrypted, for security reasons, ensuring a robust layer of protection.
Priority	1 - Essential
State	Closed
Restrictions	N/A

Table 28 - Non Func Requirement Password Encryption

Category	Safety and Security
ID	NFRS - 02
Name	Token Authentication
Description	The authentication will be done with the use of access tokens, providing security and efficiency, with server side keys.
Priority	1 - Essential
State	Closed
Restrictions	N/A

Table 29 - Non Func Requirement Token Authentication

## 4.2 Software Quality Attributes

Category	Accessibility
ID	NFRA - 01
Name	Ease of Use
Description	It is crucial that the system is developed in such a way that users will find it intuitive and easy to navigate. As such, the development team will use the <a href="http://www.acessibilidade.gov.pt">www.acessibilidade.gov.pt</a> website in order to gauge the level of accessibility as the application gets developed, to ensure the best possible experience.
Priority	2 - High Priority
State	Closed
Restrictions	N/A

*Table 30 - Non Func Requirement Ease of Use*

## 5. Diagrams

### 5.1 Package Diagram

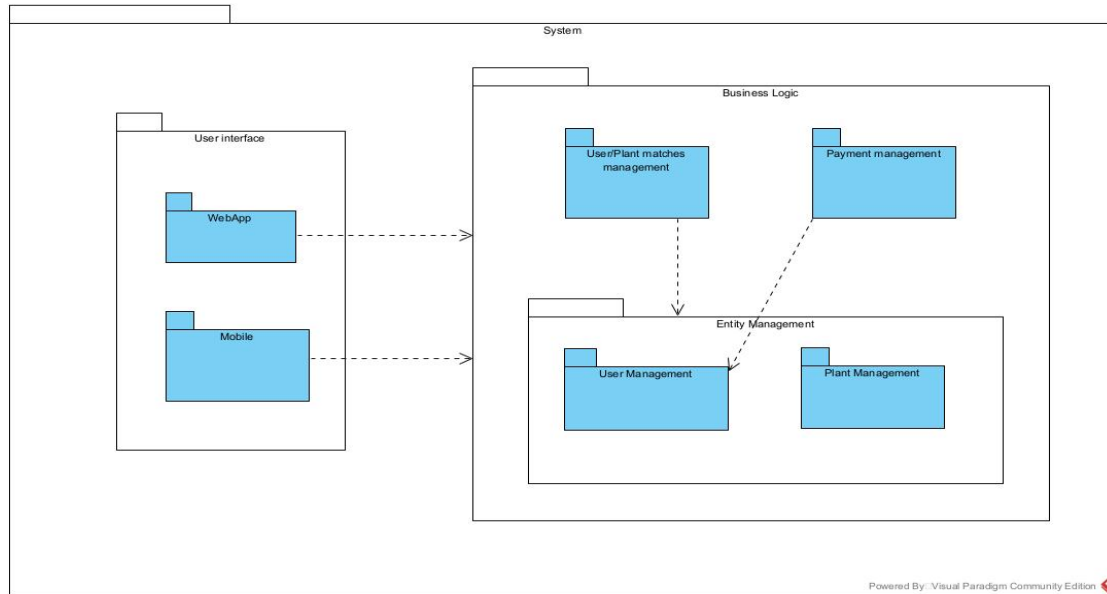


Figure 14 - Package Diagram

**Mobile and Webapp** – Responsible for handling the user interface, presenting the system’s functionality to the user.

**User Management** - responsible for handling user-related tasks, such as authentication, profile editing, and permission handling.

**Plant Management** – responsible for handling plant related tasks.

**UserPlant Matches** – dependent on both user and plant management, it’s a package that bridges the information shared by the two packages.



## 5.2 Sequence Diagrams

### Register User

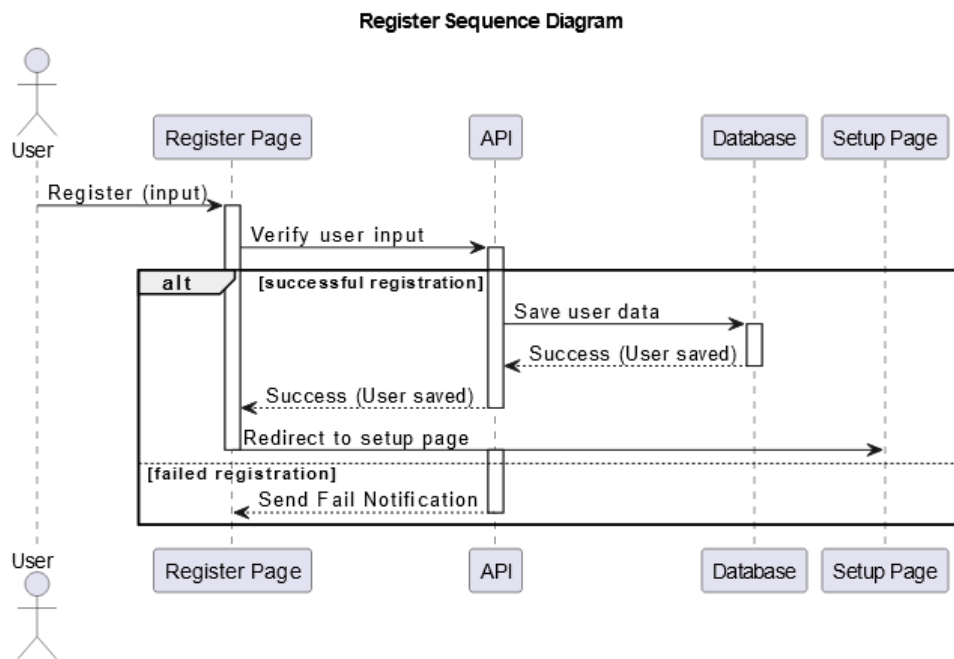


Figure 15- Register Sequence Diagram

When registering, a user will introduce fill out a register form in the platform, which will then send said form to the API for verification. If the information is verified correctly, the API will save the information into the database, and the user will receive a notice saying that they were registered and redirected to the setup page.

If the user input is not validated, the API will send a failure notification and the user will receive a notice that they must correct the inputs in the platform.

## User Fill Profile

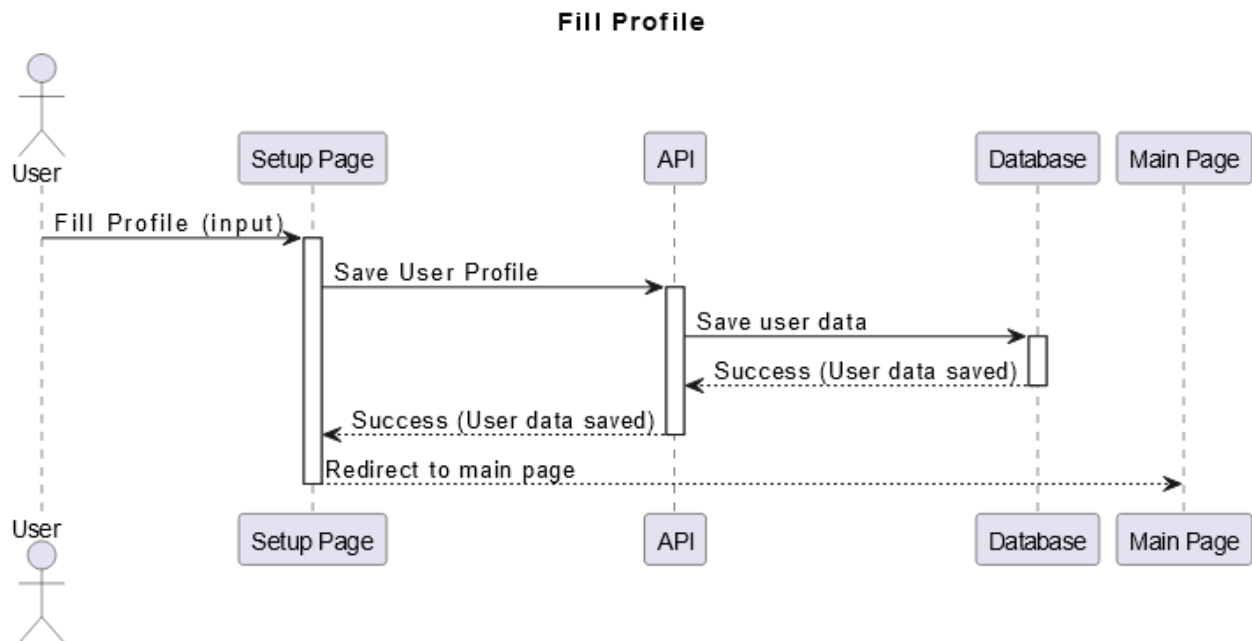


Figure 16 - Fill Profile Sequence Diagram

When filling a profile, a user will have the possibility to fill their needs through optional values regarding light availability at home, time they have to dedicate etc. Once they've filled the form, the API will save the information of the user into the database, and the end user will be redirected to the main page.

## User Edit Profile

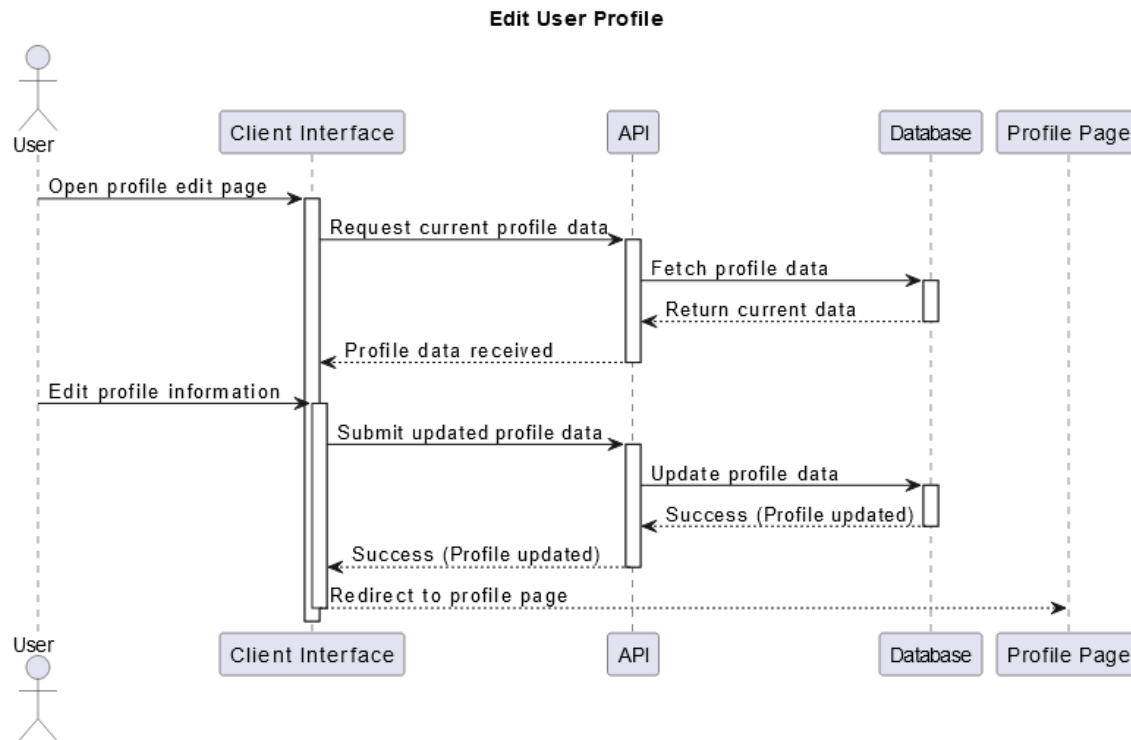


Figure 17 - Edit User Profile Sequence Diagram

For a user to edit their information, they must access the pages dedicated to the functionality, the API will then fetch their current profile data and present it in fields for the user so that they can then change the inputs in the different fields; Once done, the user will request the platform to save the new information, and the API will save it into the information, and then redirect the end user to their profile page.

## Get Matching Plants

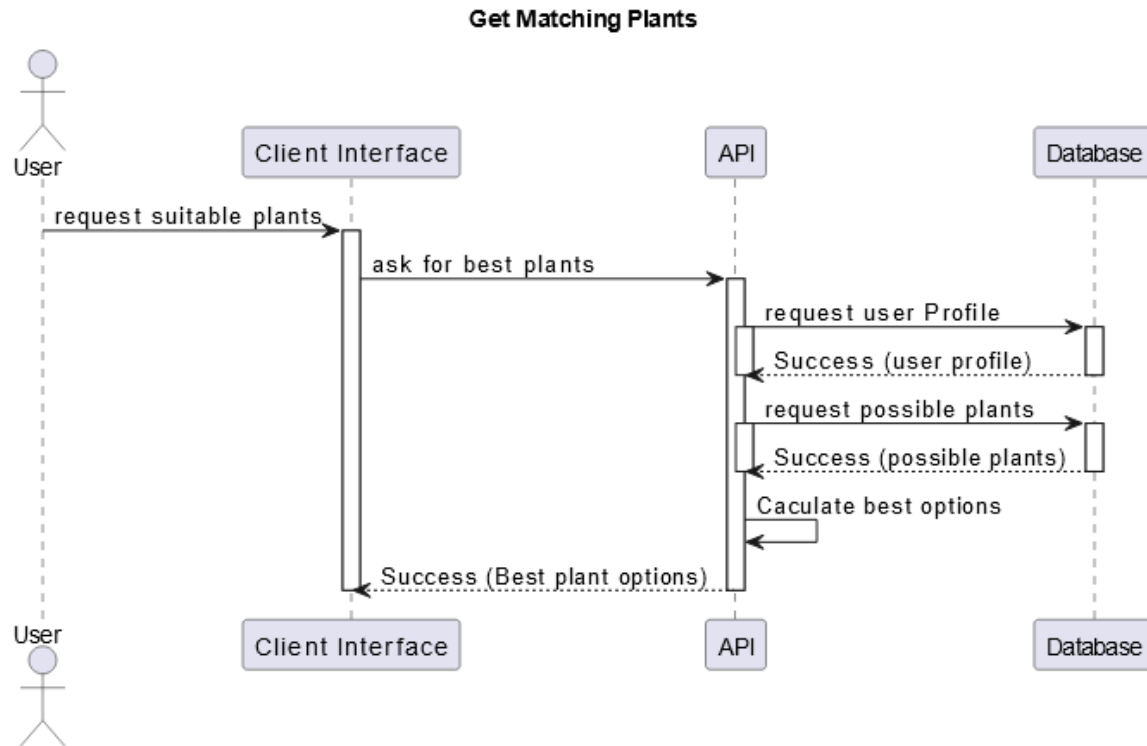


Figure 18 - Get Matching Plant Sequence Diagram

For a User to get their matching plants, a request will be done to the API, which will then fetch the User qualitative information, and the qualitative information of the plants. It will then run an algorithm that matches both users and plants, and will then present the result matches to the end user.

## Add New Plant

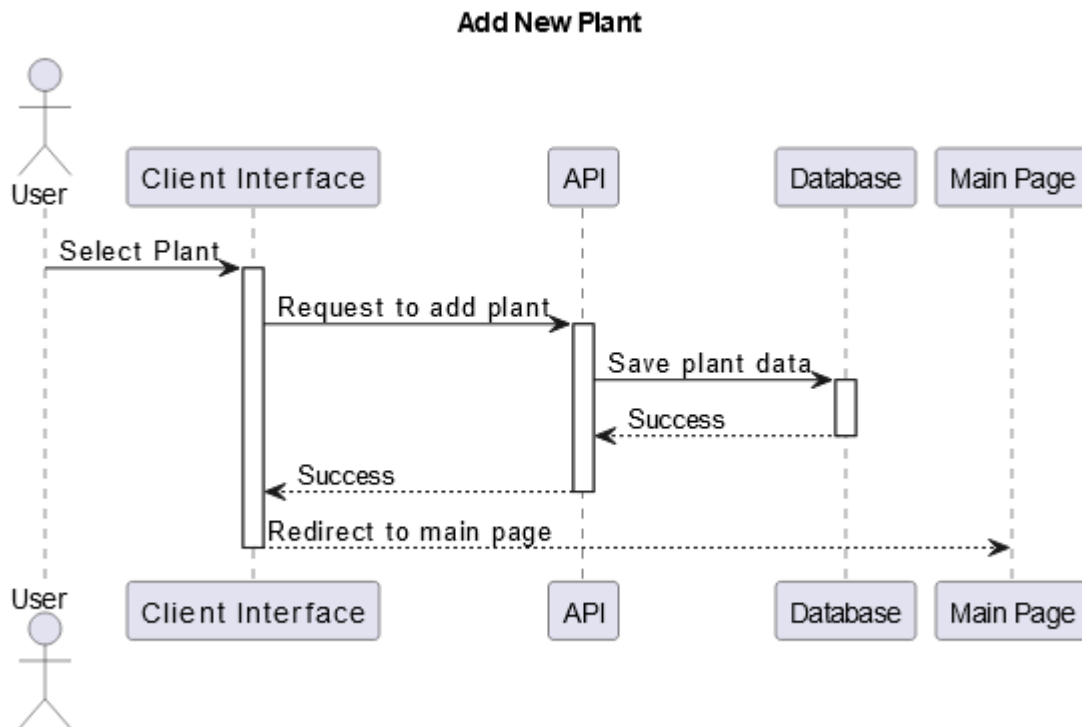


Figure 19 - Add New Plant Sequence Diagram

When adding a new plant, the end user will select a plant they wish to have associated to them, make a request to associate them to their profiles, and the Api will then save the new UserPlant data, and redirect the user to the main page.

## Remove Plant

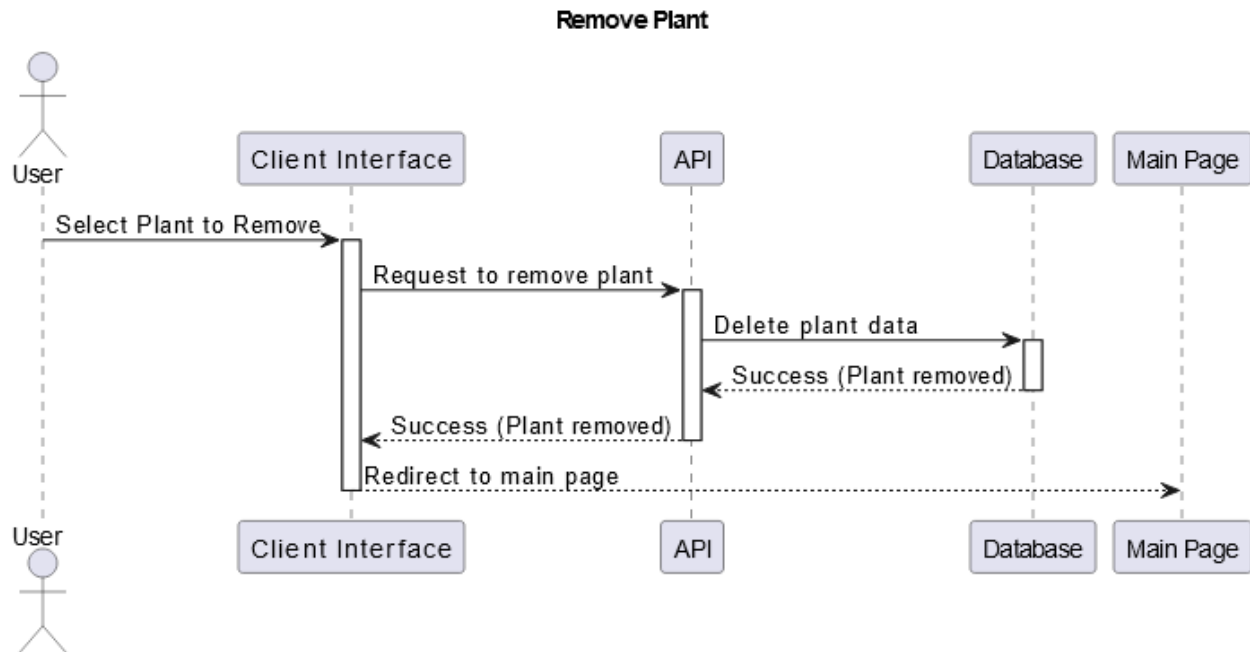


Figure 20 - Remove Plant Sequence Diagram

When a user wishes to remove a plant from their profile, they make a request on the selected plant, and the API will then delete the entry of the UserPlant from the database, and free the space for the end user to be able to select another plant in the future.

## View User Plants

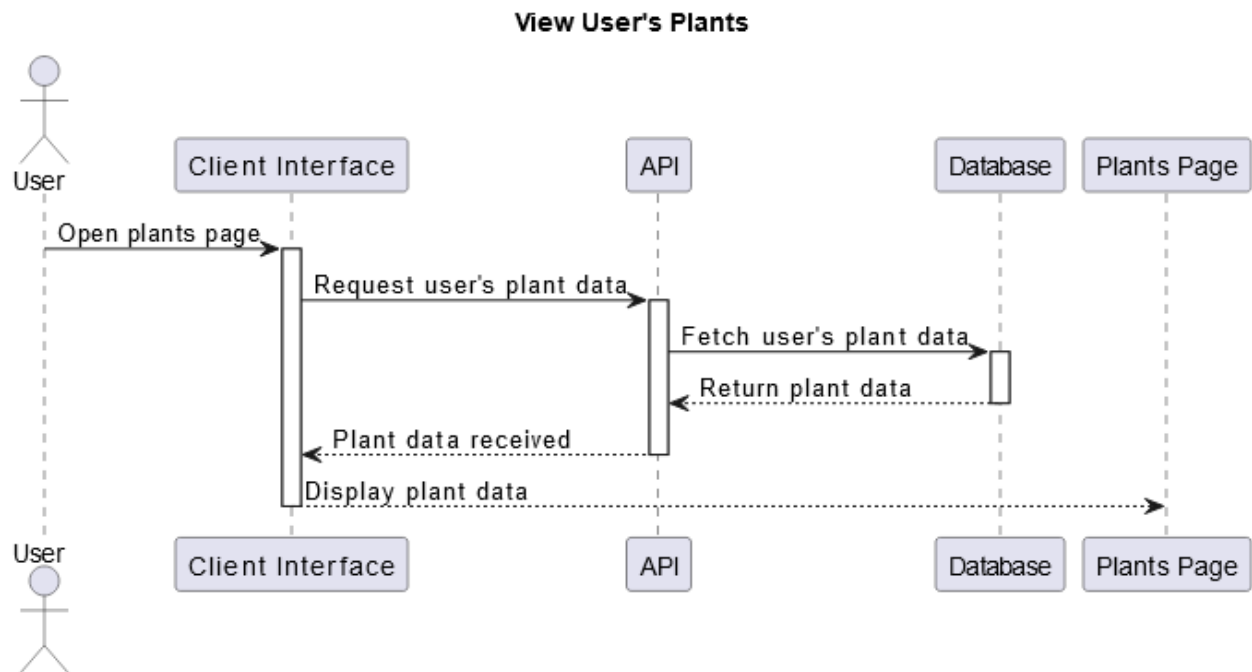


Figure 21 - View User's Plants Sequence Diagram

When a user wishes to consult one of their plants, they must open their plant page, and the API will fetch the data of that Users plants and return it.

### 5.3 Class Diagram

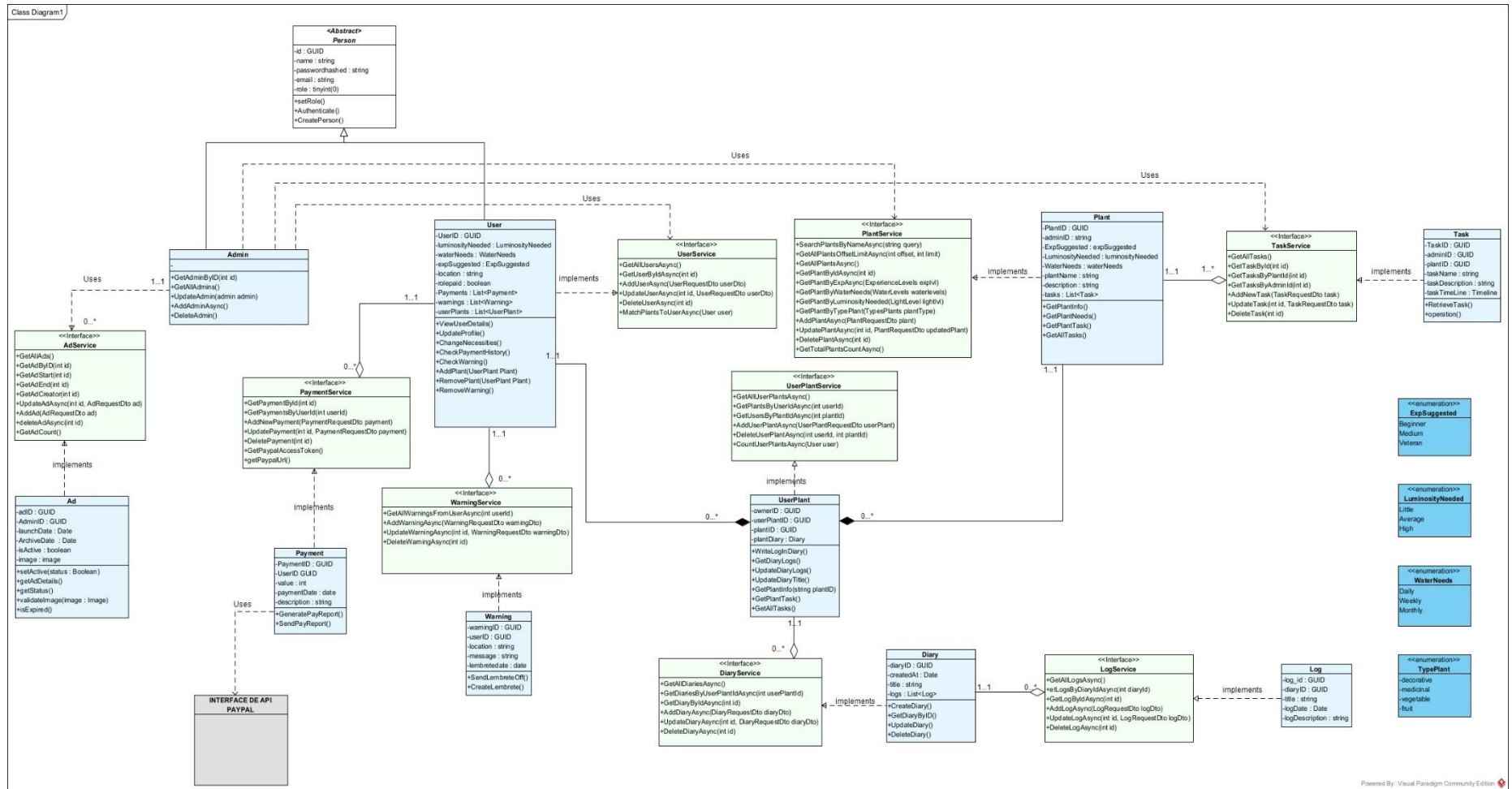


Figure 22- Class Diagram



## 5.4 Architecture Diagram

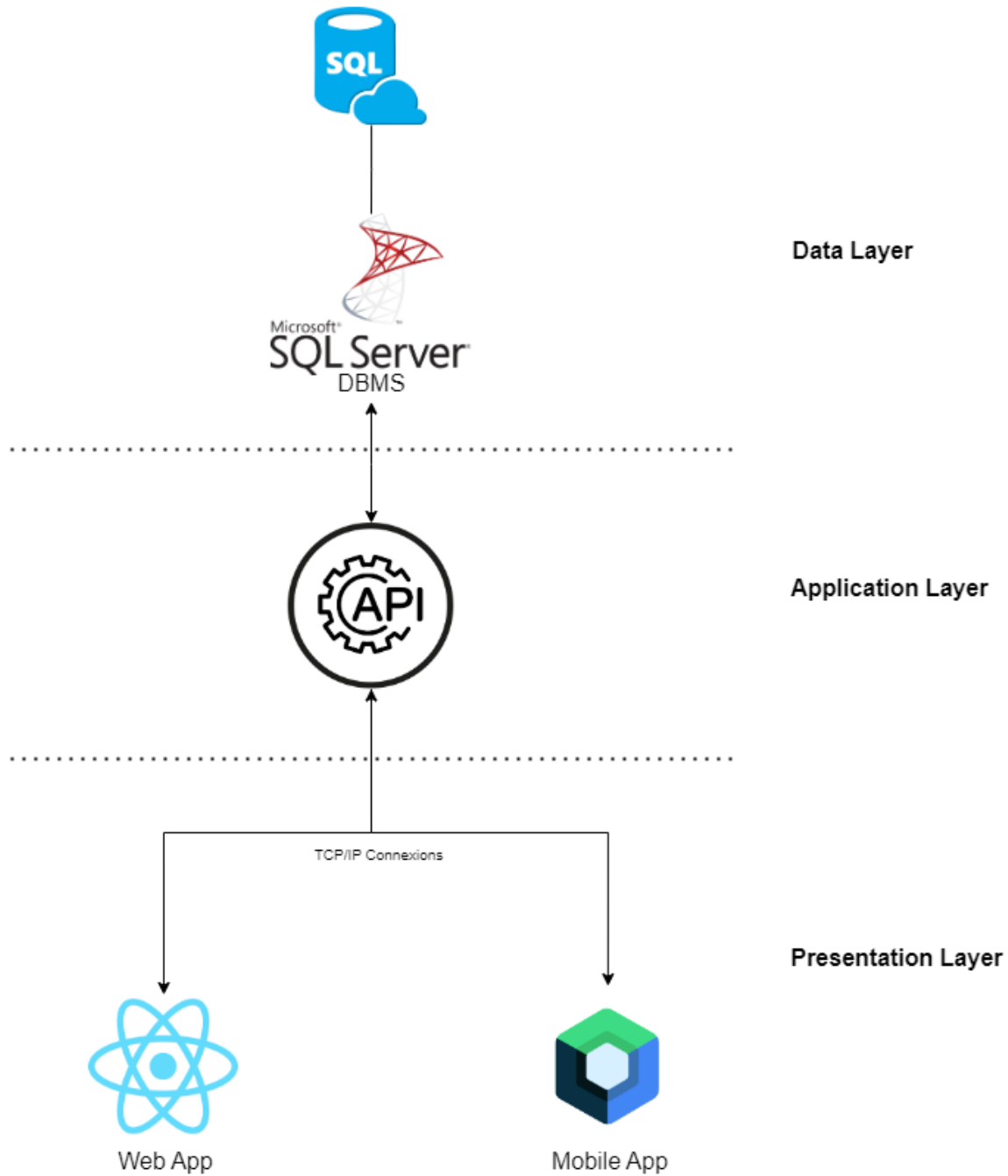


Figure 23 - Architecture Diagram

## 6. Database Design and Normalization

The database supports the entirety of the system through maintaining all the information and relationships between the information necessary for the system's functionality. The database was designed for this end, and MicrosoftSQL server was chosen as the database management platform.

### 6.1 ER Diagram

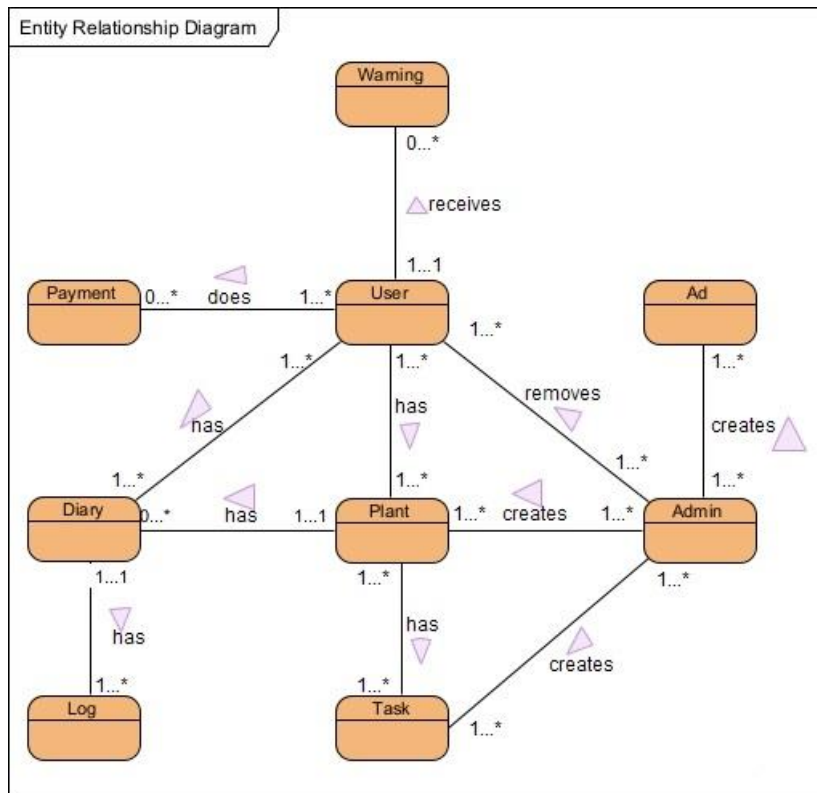


Figure 24 - First ER Diagram

This first ER diagram is supposed to represent an initial, high-level view of the database entities and the relationships that exist between said entities. In the following page we identify each entity, when they occur and their relationships.

## 6.2 Identify Entities and relationships

Entity	Description	Aliases	Occurrences
<b>User</b>	Represents an end user that uses the platform	Client, platform member.	A user occurs every time an end user signs up for the platform
<b>Admin</b>	Represents a staff member that is responsible for managing certain processes within the platform.	Staff, Employee	The platform has a pre-determined number of admins
<b>Plant</b>	Represents a plant registry (all the information regarding a plant) that is available in the platform	N/A	An admin can create one or multiple plants and add them to the platform
<b>Task</b>	Represents a unique task that belongs to a plant.	To do's	A plant can have one or more tasks, they are also admin created.
<b>Diary</b>	Represents a collective of log entries, dedicated to the end user, so they can keep track of the progress of their plant's growth, or for other ends.	Journal, record	A diary occurs whenever a User picks a plant to add to their garden.
<b>Log</b>	Represents a single entry in the diary	Entry, register	A log occurs whenever an end user writes a new entry to their diary
<b>Payment</b>	Represents a payment that users can make to the platform	Transaction	A user can make one or multiple payments to the platform.
<b>Warning</b>	Warning represents a notification sent to users when weather conditions in a certain location justify it.	Notifications	A user receives warnings when the weather conditions justify warning a user to be careful of their plants conditions
<b>Ad</b>	Ad represents an ad to be shown to users.	Advertisements	An admin can create and post one or multiple ads.

Table 31 - Identify Entities and Relationships

### 6.2.1 Multiplicity of relationships

entity	multiplicity	Relationship	Multiplicity	entity
<b>User</b>	1...1 1...1 1...1 1...1 1...1 1...1	<b>Receive</b> <b>Receives</b> <b>Does</b> <b>Has</b> <b>Has</b> <b>Does</b>	1...* 1...* 1...* 1...* 1...* 1...*	<b>Ads</b> <b>Warnings</b> <b>Tasks</b> <b>Plants</b> <b>Diaries</b> <b>Payments</b>
<b>Admin</b>	1...1 1...1 1...1 1...1	<b>Manages</b> <b>Manages</b> <b>Manages</b> <b>Removes</b>	1...* 1...* 1...* 1...*	<b>Ads</b> <b>Plants</b> <b>Tasks</b> <b>Users</b>
<b>Plant</b>	1...* 1...1	<b>Belong to</b> <b>Contains</b>	1...* 1...*	<b>Users</b> <b>Tasks</b>
<b>Task</b>	1...1	<b>Belongs to</b>	1...1	<b>Plant</b>
<b>Diary</b>	1...1 1...1	<b>Belong to</b> <b>Belong to</b>	1...1 1...1	<b>User</b> <b>Plant</b>
<b>Log</b>	1...1	<b>Belongs to</b>	1...1	<b>Diary</b>
<b>Payment</b>	1...1	<b>Is associated to</b>	1...1	<b>User</b>
<b>Warning</b>	1...1	<b>Is Sent to</b>	1...*	<b>User</b>
<b>Ad</b>	1...1	<b>Is Served to</b>	1...*	<b>User</b>

Table 32 - Multiplicity of relationships

### 6.3 Assign keys to entities

Entity	Key Value
User	userID
Admin	adminID
Plant	plantID
Task	taskID
Diary	diaryID
Log	logID
Payment	paymentID
Warning	warningID
Ad	adID

Table 33 - Assign keys to entities

### 6.4 Solve redundancies

#### 6.5.1 Examine (1:1) relationships

After analyzing all the 1 to 1 relationships, the group concluded that they do not cause redundancy of data.

#### 6.5.2 Solve (\*: \*) relationships

We have one occurrence of a many to many relationship;

To solve the issue between the User and Plant relationship, where a user can hold multiple plants, and multiple plants can belong to multiple users at the same time we decided to create a third table to hold this relationship called UserPlant. The UserPlant table solves this issue by bridging the relationship, maintaining 1 userID and 1 plantID per plant that belongs to a user.

## 6.5 Conceptual Model

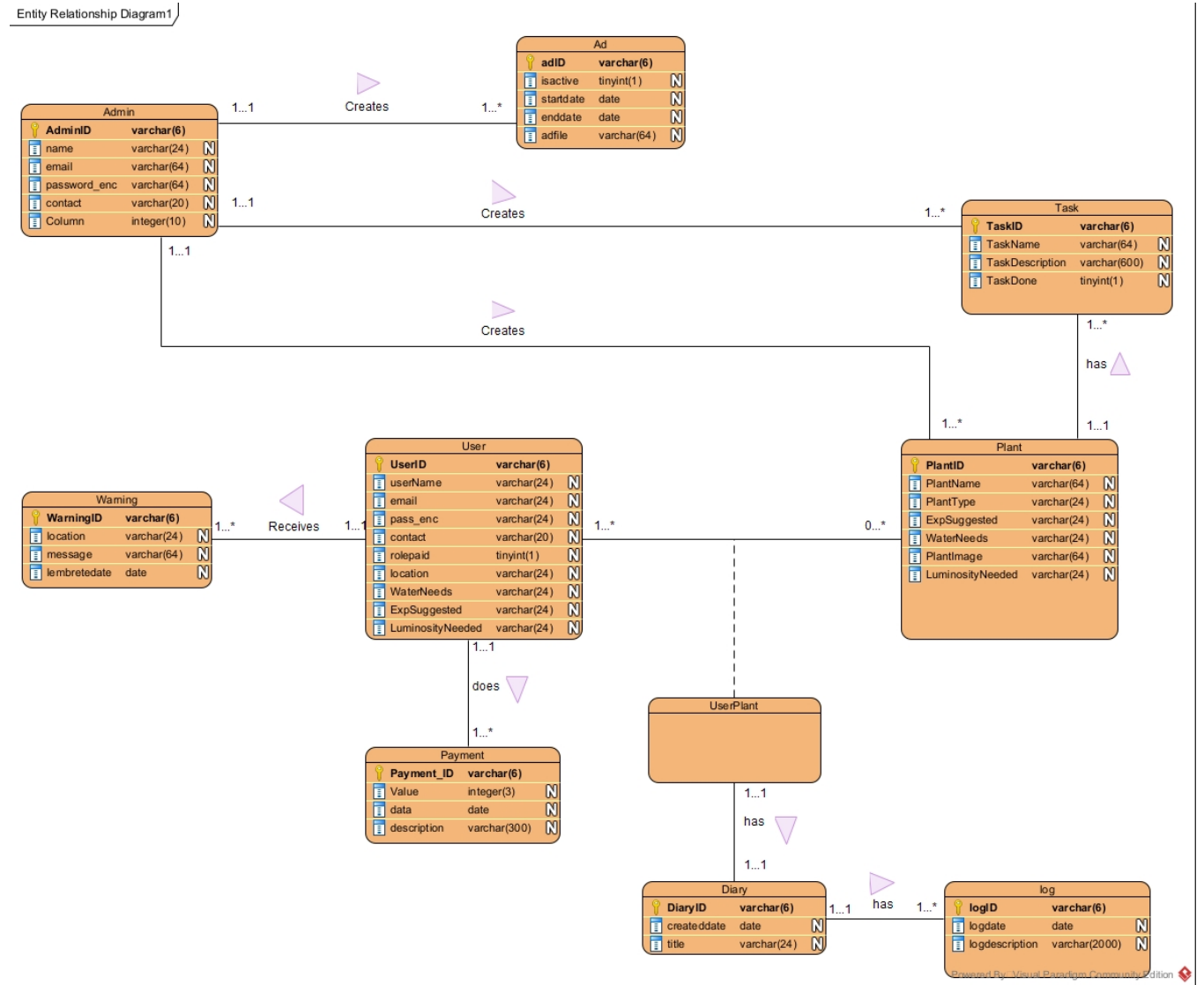


Figure 25- conceptual model

## 6.6 Data Dictionary

Entity	Attribute	Description	Datatype	Character Length	Acceptable Values	Required	Null
Admin	<u>AdminID</u>	Unique Admin identifier	varchar	6	N/A	yes	no
	Name	Admin's name	varchar	24	N/A	yes	no
	Email	Admin's contact email	varchar	64	N/A	yes	no
	Password	Encrypted password	varchar	64	N/A	yes	no
	Contact	Admin alternate contact	varchar	20	N/A	yes	no
Ad	<u>AdID</u>	Unique ad identifier	varchar	6	N/A	yes	no
	<u>AdminID</u>	Identifier of the admin that created the ad	varchar	6	N/A	yes	no
	IsActive	Whether or not the ad is active	tinyint	1	1 0	yes	no
	StartDate	Starting date for the ad activity	Date	N/A	N/A	yes	no
	EndDate	Ending date for the ad activity	Date	N/A	N/A	yes	no
	AdFile	File containing ad information	varchar	64	N/A	yes	no
Task	<u>TaskID</u>	Unique identifier for a task	varchar	6	N/A	yes	no
	<u>AdminID</u>	Admin that created the task	varchar	6	N/A	yes	no
	<u>PlantID</u>	Plant to which the task belongs	varchar	6	N/A	yes	no
	TaskName	Name of the task	varchar	64	N/A	yes	no
	TaskDescription	Description of the task	varchar	600	N/A	yes	no
Plant	<u>PlantID</u>	Unique identifier of the plant	varchar	6	N/A	yes	no
	<u>AdminID</u>	Admin responsible for creating the plant	varchar	6	N/A	yes	no
	PlantName	Name of the plant	varchar	64	N/A	yes	no
	PlantType	Type of plant	varchar	24	"Decorative", "Fruit", "Vegetable", "Fungus", "Cacti", "Succulent"	yes	no

	<b>ExpSuggested</b>	Suggested experience for caretaking	varchar	24	“High”, “Intermediate”, “Low”	yes	no
	<b>WaterNeeds</b>	Plant’s need for water	varchar	24	“High”, “Intermediate”, “Low”	yes	no
	<b>LuminosityNeeded</b>	Plant’s need for light	varchar	24	“High”, “Intermediate”, “Low”	yes	no
	<b>PlantImage</b>	Sample image of the plant	varchar	64	N/A	yes	no
<b>Warning</b>	<b><u>WarningID</u></b>	Unique warning identifier	varchar	6	N/A	yes	no
	<b><u>UserID</u></b>	Unique identifier of user to receive the warning	varchar	6	N/A	yes	no
	<b>Location</b>	User’s location for the warning	varchar	24	List of cities	yes	no
	<b>Message</b>	Message contained in the warning	varchar	64	N/A	yes	no
	<b>LembreteDate</b>	Date the warning is sent out	date	N/A	N/A	yes	no
<b>User</b>	<b><u>UserID</u></b>	Unique user identifier	varchar	6	N/A	yes	no
	<b>Username</b>	User’s username	varchar	24	N/A	yes	no
	<b>Email</b>	User’s email	varchar	24	N/A	yes	no
	<b>Password</b>	User’s encrypted password	varchar	24	N/A	yes	no
	<b>Contact</b>	User’s contact	varchar	20	N/A	yes	no
	<b>Rolepaid</b>	Whether or not user has paid for premium	tinyint	1	1 0	yes	no
	<b>Location</b>	User’s location	varchar	24	List of cities	yes	no
	<b>WaterAvailability</b>	Availability the user has to water their plants	varchar	24	“High”, “Intermediate”, “Low”	yes	no
	<b>CareExperience</b>	User’s experience in plant care	varchar	24	“High”, “Intermediate”, “Low”	yes	no
	<b>LuminosityAvailability</b>	User’s luminosity availability	varchar	24	“High”, “Intermediate”, “Low”	yes	no



<b>UserPlant</b>	<b><u>UserID</u></b>	Plant's owner identifier	varchar	6	N/A	yes	no
	<b><u>PlantID</u></b>	Plant identifier	varchar	6	N/A	yes	no
	<b>UserPlantID</b>	Identifies a specific plant belonging to a user	varchar	6	N/A	yes	no
<b>Payment</b>	<b><u>PaymentID</u></b>	Unique identifier of the payment	varchar	6	N/A	yes	no
	<b><u>UserID</u></b>	Unique identifier of the paying user	varchar	6	N/A	yes	no
	<b>Value</b>	Value present in the payment	integer	3	N/A	yes	no
	<b>Date</b>	Payment date	date	N/A	N/A	yes	no
	<b>Description</b>	Payment description	varchar	300	N/A	yes	no
<b>Diary</b>	<b><u>DiaryID</u></b>	Unique identifier of the diary	varchar	6	N/A	yes	no
	<b><u>UserPlantID</u></b>	User's plant to which the diary belongs	varchar	6	N/A	yes	no
	<b>CreatedDate</b>	Date the diary was created	date	N/A	N/A	yes	no
	<b>Title</b>	Diary title	varchar	24	N/A	yes	no
<b>Log</b>	<b><u>LogID</u></b>	Unique identifier for the log	varchar	6	N/A	yes	no
	<b><u>DiaryID</u></b>	Unique identifier for the diary the log belongs to	varchar	6	N/A	yes	no
	<b>LogDate</b>	Date the log was created	date	N/A	N/A	yes	no
	<b>LogDescription</b>	Content of the log	varchar	2000	N/A	yes	no

Figure 26 - Data Dictionary

## **6.7 Logical Model**

### **Normalization**

The process of normalization is done to validate the relationships obtained in the prior steps. This process is done by working on the dependencies between the attributes on the both ends of a relationship.

**First Normative Form** – in this first level of normalization, each line must correspond to an entry and each column should correspond to only one attribute. By analyzing the previous steps, we conclude that the database is already in this form.

**Second Normative Form** – in this level of normalization, each non key attribute must be completely dependent on the totality of the primary key, and not only to part of the primary key. By analyzing the previous steps, we conclude that the database is in this form.

**Third Normative Form** – in this level of normalization, no non key attribute should be functionally dependent of another attribute that is not primary key. If there are, they should be separated into new tables. After analyzing the previous steps, we conclude that the database is in this form.

Each next step of the normalization presumes that the previous step has been achieved.

## Presentation of the logical diagram

Entity Relationship Diagram1

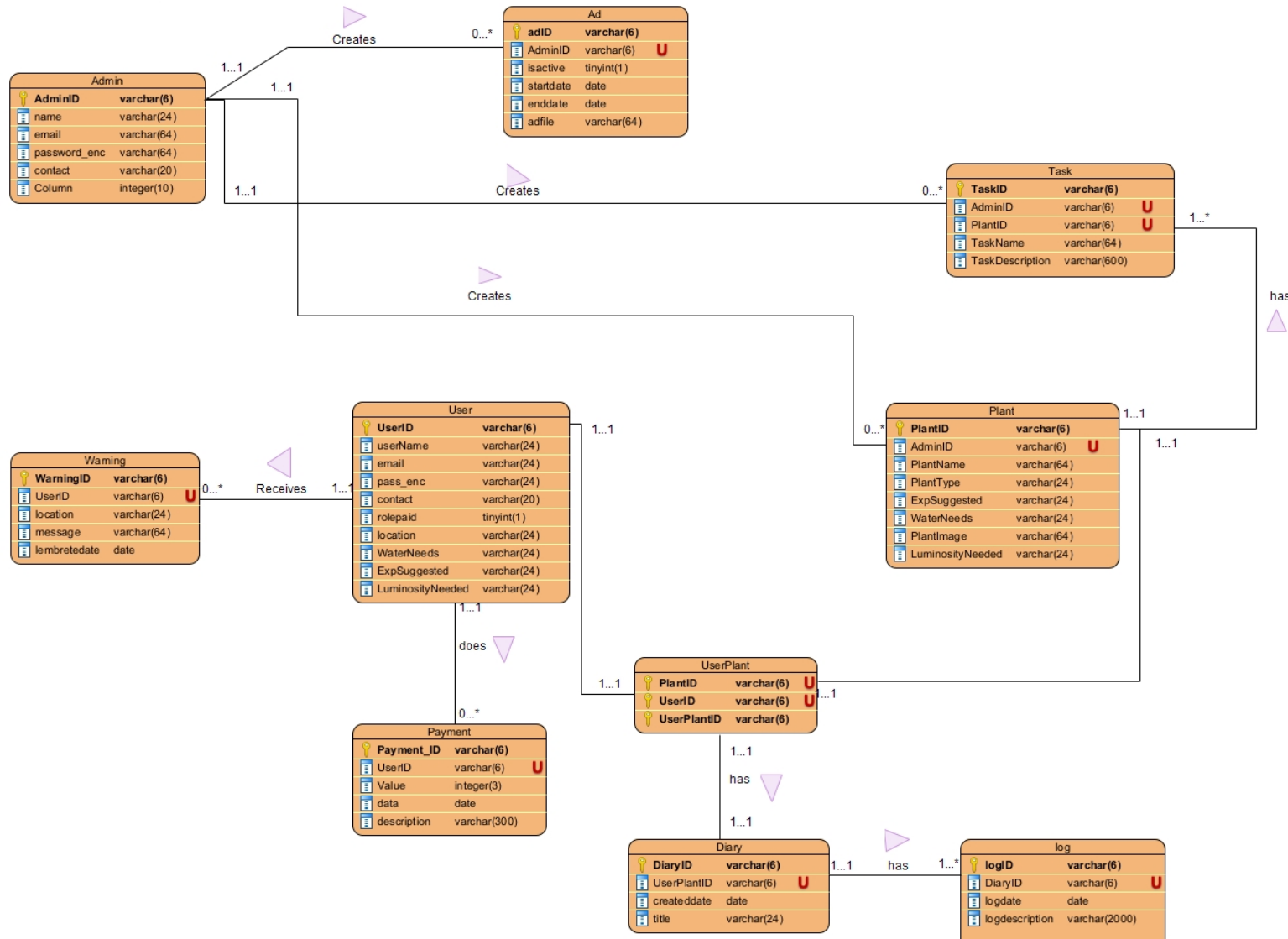


Figure 27 - Logical Diagram, foreign keys are represented with the letter U

## 7. Use Case

### 7.1 User Management

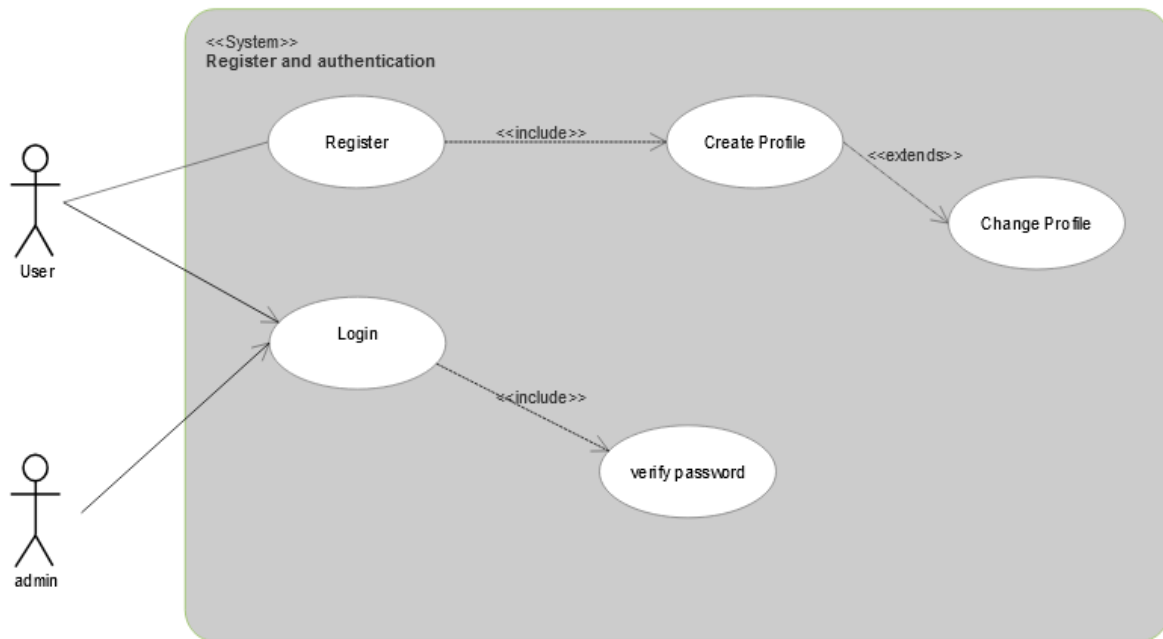


Figure 28 - Use Case Register and authentication

This use case illustrates the register and authentication process for both users, and admins.

**Register:** Only users may register, during the register process, a user will be invited to **create their profile** in order to set them up to use all the functionalities of the system. The user may later **change their profile** if they desire to do so.

**Login:** Both user and admins have access to the login feature. It Includes **verifying password** as part of the authentication process.

## 7.2 Administration

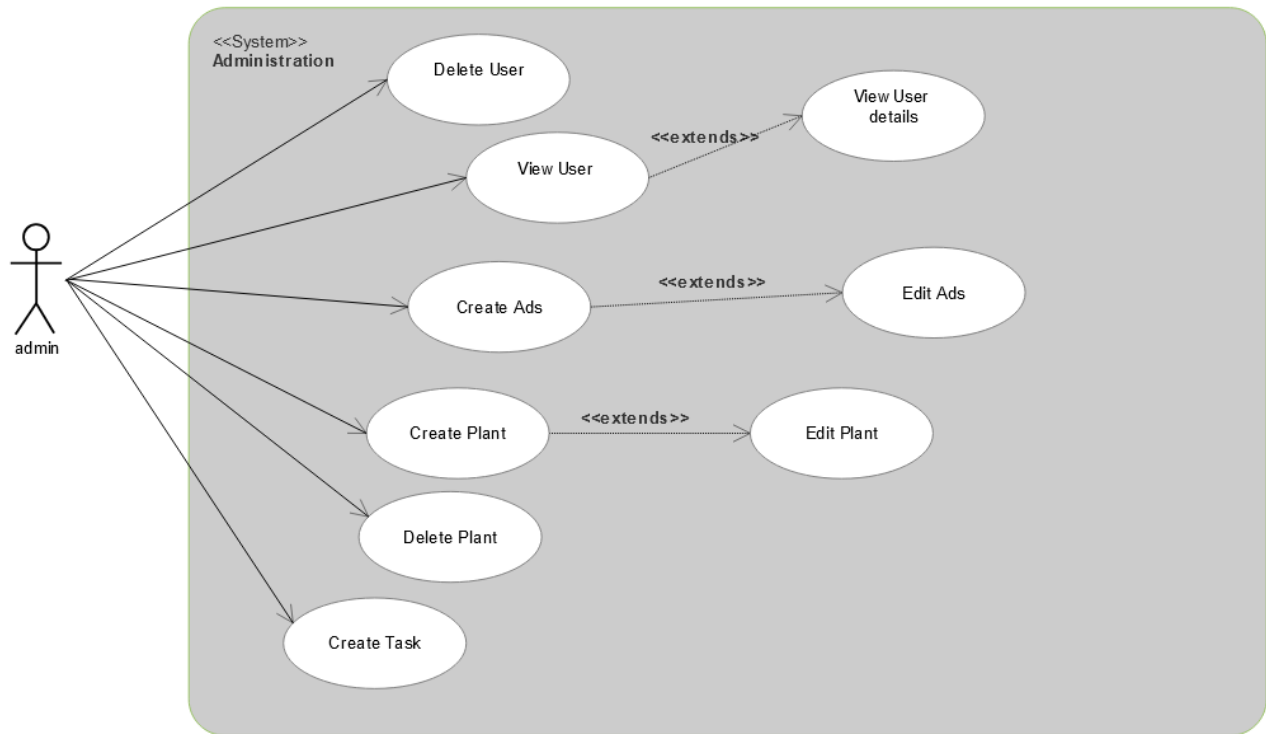


Figure 29 - use case administration

This use case illustrates the administration processes an admin has access to.

**Delete User:** Only admins can delete users, for this, they must select the user they wish to delete.

**View User:** An admin can get the information of a user, like their username for example, this case extends a View User Details, in which more detailed information is given.

**Create plant:** Admins are the type of user capable of creating plant registries. They may later on choose to edit said plant registries.

**Delete plant:** Since admins are able to create plants, they are also able to delete the plant registries from the system.

**Create Task:** Admins are able to create tasks and associate them to plants.

**Create Ad:** Admins are able to create ads, and edit them later on.

### 7.3 Plant Matchmaking

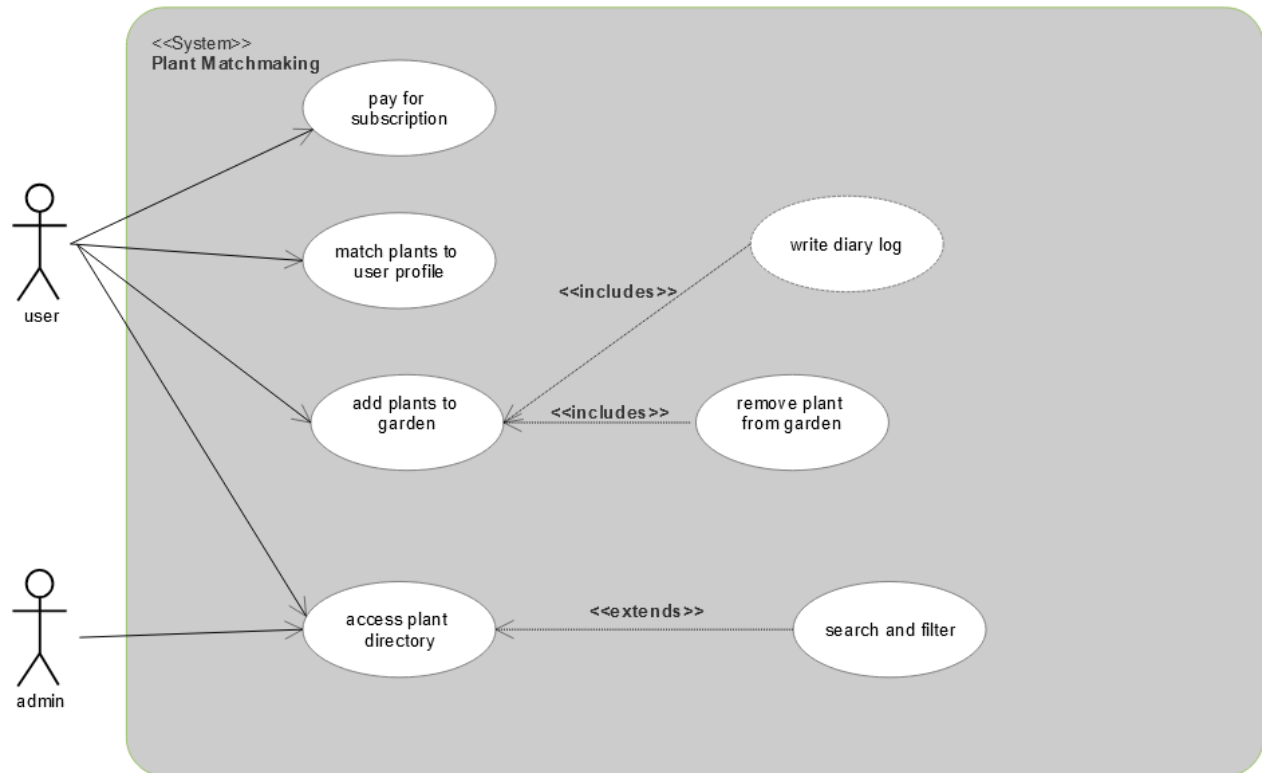


Figure 30 - use case Plant Matchmaking

This case illustrates what both users and admins can do in the front end of the platform.

**Pay For subscription:** Only users are given the chance to pay for a subscription to the platform.

**Match plants to user profile:** Users can access their plant matches based on their profiles.

**Add plants to garden:** A user may “add a plant to their garden” which is to say, associate a plant registry with their profile, linking both the user and the plant into a user owned plant. They can later **remove** this plant, or **write in their plant diary**.

**Access plant directory:** Both admins can search the plant directory, they are also given the change to search and filter it.

## 8. Evidence of Implementation

In this section of the document, we'll be providing all the evidence required in the deliverables for milestone4, from proof that we used .Net core, tokens for authentication, and proof of usage of EF framework.

### 8.1 Proof of netcore

This project uses .NET Core, a cross-platform, high-performance framework ideal for building modern web APIs.

#### Key Components

- **Dependency Injection (DI):** Services, such as **UserService** and **AdminService**, are injected into controllers using .NET Core's built-in DI container.
- **Middleware Pipeline:** Custom middlewares, such as **CustomExceptionMiddleware**, are added to handle exceptions globally.
- **Authentication and Authorization:** JWT-based authentication is configured, leveraging .NET Core's support for JWT bearer tokens.
- **Entity Framework Core:** This project uses EF Core as the ORM, with code-first migrations to manage the database schema.

#### Proof

##### Dependency Injections

Using .NET Core, services are registered in the **Program.cs** file using **builder.Services**. Below is a snippet of the registration of a few of the core services:

```
// Registro do PlantService no container de DI
builder.Services.AddScoped<PlantService>();
builder.Services.AddScoped<UserService>();
builder.Services.AddScoped<AdminService>();
builder.Services.AddScoped<WarningService>();
builder.Services.AddScoped<UserPlantsService>();
```

*Figure 31 -. NetCore dependency injection*

## Middleware And Routing configuration

This is where Cors policies, Swagger, json, and JWT authentication options went.

```
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAllOrigins", builder =>
    {
        builder.AllowAnyOrigin() // Permite qualquer origem
        .AllowAnyMethod() // Permite qualquer método (GET, POST, etc..)
        .AllowAnyHeader(); // Permite qualquer cabeçalho
    });
});

// Configurações do Swagger
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "WebAPI", Version = "v1" });

    // Configuração para JWT Auth
    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        Name = "Authorization",
        Type = SecuritySchemeType.Http,
        Scheme = "Bearer",
        BearerFormat = "JWT",
        In = ParameterLocation.Header,
        Description = "Insira 'Bearer' [espaço] e o token JWT na caixa de texto abaixo.\n\nExemplo: 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...'";
    });

    c.AddSecurityRequirement(new OpenApiSecurityRequirement
    {
        {
            new OpenApiSecurityScheme
            {
                Reference = new OpenApiReference
                {
                    Type = ReferenceType.SecurityScheme,
                    Id = "Bearer"
                }
            },
            new string[] {}
        }
    });
});
```

*Figure 32 - builder object in program.cs*



## App object

Represents web application instance. Here we configure the request pipelines, a distinctive .net core design.

```
var app = builder.Build();

// Registra o middleware customizado
app.UseMiddleware<CustomExceptionMiddleware>();
// Registra o middleware de tratamento de erros
app.UseMiddleware<ErrorHandlingMiddleware>();

// Configura o pipeline para o Swagger
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "PlantMatchAPI v1");
    });
}

// Habilita o middleware CORS
app.UseCors("AllowAllOrigins");

app.UseHttpsRedirection();

// Adiciona o middleware para forçar URLs minúsculas
app.Use(async (context, next) =>
{
    context.Request.Path = context.Request.Path.Value.ToLowerInvariant();
    await next.Invoke();
});

// Adiciona o middleware de autenticação
app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app.Run();
```

Figure 33 - app object in program.cs

## 8.2 Proof of tokens for authentication

```
// Configuração para JWT Auth
c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
{
    Name = "Authorization",
    Type = SecuritySchemeType.Http,
    Scheme = "Bearer",
    BearerFormat = "JWT",
    In = ParameterLocation.Header,
    Description = "Insira 'Bearer' [espaço] e o token JWT na caixa de texto abaixo.\n\nExemplo: 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...'"
});

c.AddSecurityRequirement(new OpenApiSecurityRequirement
{
    {
        new OpenApiSecurityScheme
        {
            Reference = new OpenApiReference
            {
                Type = ReferenceType.SecurityScheme,
                Id = "Bearer"
            },
            new string[] {}
        }
    }
});
```

Figure 34 - proof of JWT tokens for authentication

```
{
  "Jwt": {
    "Key": "d4f7b7a56c34b4c34a78b26f62fdb0f7",
    "Issuer": "WebAPI",
    "Audience": "WebAPIUsers" // Público da aplicação
  },
}
```

Figure 35 – JWT key

```
/// <summary>
/// Generates a JSON Web Token (JWT) for the given claims.
/// </summary>
/// <param name="claims">The claims to include in the token.</param>
/// <returns>A string representing the generated JWT.</returns>
1 reference
private string GenerateToken(IEnumerable<Claim> claims)
{
    // Create a symmetric security key using the secret key from configuration
    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));

    // Create signing credentials using the security key and the HMAC SHA-256 algorithm
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    var token = new JwtSecurityToken(
        issuer: _configuration["Jwt:Issuer"],
        audience: _configuration["Jwt:Audience"],
        claims: claims,
        expires: DateTime.Now.AddMinutes(60),
        signingCredentials: creds
    );

    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

Figure 36 - token generator

## 8.3 Proof of EF

### Configuration of EF Core

```
// adiciona o serviço do dbcontext com sql server
builder.Services.AddDbContext<DataContext>(options =>
{
    options.UseSqlServer(builder.Configuration.GetConnectionString("defaultconnection"));
});
```

Figure 37 - adding Database Context

### DbContext Class

```
using Microsoft.EntityFrameworkCore;
using System.Drawing;
using WebAPI.DTO;
using WebAPI.Entity;

namespace WebAPI.Data
{
    public class DataContext : DbContext
    {
        1 reference
        public DataContext(DbContextOptions<DataContext> options) : base(options)
        {
        }

        // Tabelas da Base de Dados

        1 reference
        public DbSet<Person> Person { get; set; }
        11 references
        public DbSet<Admin> Admin { get; set; }
        16 references
        public DbSet<User> User { get; set; } // Tabela de utilizadores
        13 references
        public DbSet<Ad> Ad { get; set; }
        12 references | 1/1 passing
        public DbSet<Entity.Task> Task { get; set; } // Tabela de tarefas
        25 references
        public DbSet<Plant> Plant { get; set; } // Tabela de plantas
        10 references
        public DbSet<Warning> Warning { get; set; }
        14 references
        public DbSet<UserPlant> UserPlant { get; set; } // Tabela de junção para a relação muitos-para-muitos
        9 references
        public DbSet<Payment> Payment { get; set; }
        9 references
        public DbSet<Diary> Diary { get; set; }
        7 references
        public DbSet<Log> Log { get; set; }
```

Figure 38 - mapping tables

## Mapping Entities

```
// Configuração de mapeamento de entidade
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{

    // Set up TPH or TPT inheritance
    modelBuilder.Entity<Person>()
        .ToTable("Persons"); // Single table if using TPH

    modelBuilder.Entity<Admin>()
        .ToTable("Admins"); // Separate table if using TPT

    modelBuilder.Entity<User>()
        .ToTable("Users"); // Separate table if using TPT


    base.OnModelCreating(modelBuilder);

    // Configuração dos enums para User
    modelBuilder.Entity<User>()
        .Property(u => u.CareExperience)
        .HasConversion<string>();

    modelBuilder.Entity<User>()
        .Property(u => u.WaterAvailability)
        .HasConversion<string>();

    modelBuilder.Entity<User>()
        .Property(u => u.LuminosityAvailability)
        .HasConversion<string>();

    // Configurações de enum para Plant
    modelBuilder.Entity<Plant>()
        .Property(p => p.Type)
        .HasConversion<string>();

    modelBuilder.Entity<Plant>()
        .Property(p => p.ExpSuggested)
        .HasConversion<string>();

    modelBuilder.Entity<Plant>()
        .HasOne(p => p.Admin) // Each Plant has one Admin
        .WithMany(a => a.Plants) // An Admin can have many Plants
        .HasForeignKey(p => p.AdminID) // Foreign key in Plant
        .OnDelete(DeleteBehavior.Restrict); // Prevent cascade delete when Admin is deleted
}
```

Figure 39 - mapping entites

## **9. Tests**

The group decided to go with unit tests in order to test all functionalities from services and controller methods. We will also, along with the report and back-end, deliver a collection of postman requests that serve to test all the end points.

### **9.1 Unit Tests**

#### **WebAPI context**

All tests run on the information provided in WebAPIContext.cs, which serves as a temporary database, that is only used whenever we run tests, and then thrown away. It only exists on test run time.

```
15 namespace WebAPITests
16 {
17     28 references
18     internal class WebApiContext
19     {
20         private readonly IConfiguration _configuration;
21         private readonly Cryptography _cryptography;
22
23         18 references
24         public WebApiContext()
25         {
26             _configuration = new ConfigurationBuilder().AddJsonFile("appsettings.json").Build();
27             _cryptography = new Cryptography(_configuration);
28         }
29
30         18 references
31         public DataContext getWebApiContext()
32         {
33             var options = new DbContextOptionsBuilder<DataContext>()
34                 .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())
35                 .Options;
36             var databaseContext = new DataContext(options);
37             databaseContext.Database.EnsureCreated();
38             //if(await databaseContext)
39             if (databaseContext == null)
40             {
41                 throw new Exception("Database context is null.");
42             }
43
44             // populate temporary database
45             populateAdmins(databaseContext);
46             populatePlants(databaseContext);
47             populateUsers(databaseContext);
48             populateAds(databaseContext);
49             populateTasks(databaseContext);
50             populateUserPlants(databaseContext);
51             populatePayments(databaseContext);
52             populateWarnings(databaseContext);
53
54             return databaseContext;
55         }
56     }
57 }
```

Figure 40 - webApi context for tests

## Testing Services

Each service is tested in its designated class and these can be found inside the project in the **WebAPITests** section. Each of the members got delegated a few classes that they were in charge of designing tests for, and implementing said tests. Along with the codification of these, we also created excel files to document test cases.

```

AdminServiceTests.cs
WebAPITests
WebAPITests.services.AdminServiceTests

0 references
24 public AdminServiceTests()
25 {
26     var webApiContext = new WebApiContext();
27
28     _context = webApiContext.getWebApiContext();
29     _configuration = new ConfigurationBuilder().AddJsonFile("appsettings.json").Build();
30     _cryptograpy = new Cryptography(_configuration);
31     _adminService = new AdminService(_context, _cryptograpy);
32 }
33
34 0 references
35 public void Dispose()
36 {
37     _context.Database.EnsureDeleted();
38     _context.Dispose();
39 }
40
41 /// <summary>
42 /// Tests whether retrieving an admin by id returns successfully
43 /// </summary>
44 [Fact]
45 0 references
46 public async void GetAdminByIdAsyncShouldNotBeNull()
47 {
48     var expectedAdminId = 1;
49     var admin = await _adminService.GetAdminByIdAsync(expectedAdminId);
50
51     Assert.NotNull(admin);
52 }

```

Figure 41 - example of service testing class

These excel files will also be found in the delivery for milestone 4.

TEST CASE ID	TEST SCENARIO	TEST CASE	PRE-CONDITION	TEST STEPS	TEST DATA	EXPECTED RESULT	POST CONDITION	ACTUAL RESULT	STATUS (PASS/FAIL)
GetAllPaymentsAsyncShouldNotBeEmpty	Verify that the method getallpayments doesn't return an empty list	call the method getallpayment	there are payment in the db	1. Call the method	<valid info in the DB>	Successful return of a list with the payment information	N/A	Successful return of a list with the payment information	PASS
GetPaymentsByUserIdAsyncShouldBeCorrectForValidID	Verify that the method GetPaymentById returns a list of payments	Enter valid ID info	there are payments in the db associated with the userID	1. create ID 2. Call the method	var expectedPaymentID = 1;	Successful return of a list with the payment information	N/A	Successful return of a list with the payment information	PASS
GetPaymentsByUserIdAsyncShouldBeCorrectForInvalidID	Verify that the method GetPaymentById returns null when the ID is invalid	Enter invalid ID info	there are payments in the db not associated with the userID	1. create ID 2. Call the method	var expectedPaymentID = 10;	returns null	N/A	returns null	PASS
GetPaymentsByUserIdShouldNotBeNull	Verify that the method get GetPaymentsByUserId returns a list of payments	Enter valid UserID info	there are payments in the db associated with the userID	1. create ID 2. Call the method	var UserID = 4	returns a list of payment objects	N/A	returns a list of payment objects	PASS

Figure 42 - snippet of excel file for test cases

## Testing Controllers

Controller tests can be found in the same section (WebAPITests), in a file entitled “controllers”. Each controller tests has various services inside of it, and work with the same webApiContext as services. The test cases serve to test if the controllers are returning the right kind of responses to requests.

```
namespace WebAPITests.controllers
{
    public class AdminControllerTests : IDisposable
    {
        private readonly DataContext _context;
        private readonly IConfiguration _configuration;
        private readonly Cryptography _cryptography;
        private readonly AdminController _controller;
        private readonly WebApiContext _webApiContext;
        private readonly AdminService _adminService;

        public AdminControllerTests() {
            var webApiContext = new WebApiContext();
            _context = webApiContext.getWebApiContext();
            _configuration = new ConfigurationBuilder().AddJsonFile("appsettings.json").Build();
            _cryptography = new Cryptography(_configuration);
            _adminService = new AdminService(_context, _cryptography);
            _controller = new AdminController(_adminService);
        }

        public void Dispose()
        {
            _context.Database.EnsureDeleted();
            _context.Dispose();
        }

        /// <summary>
        /// Tests wether get all method corretly returns all admins
        /// </summary>
        [Fact]
        public async void Admin_GetAllAdmins_ReturnsList()
        {
            //Arrange
            var result = await _controller.GetAllAdmins();

            //Act
            var okResult = Assert.IsType<OkObjectResult>(result.Result);
            var data = okResult.Value as dynamic;

            //Assert
            Assert.NotNull(data);
            Assert.Equal(2, data.total);
            Assert.Equal("Admin", data.data[0].Username);
            Assert.Equal("Admin2", data.data[1].Username);
        }
    }
}
```

Figure 43 - Testing Controllers file



## 9.2 Postman Tests

in order to test the API, besides swagger, we have also created various postman requests to guarantee the complete cover of the endpoints. These can be found in files, where each of them contains a collective of necessary requests to test specific routes.

All that is necessary, is to execute the requests from top to bottom in order to obtain the desired output.

Each file is independent of the rest, therefore, there's no need to execute the endpoints of a file, in order to test another.

Along with the report there are two annexed json files:

The route file - contains all the API endpoints to be tested.

Environment variables files - includes all environment variables necessary for the routes to work correctly. To note that postman will only work if there is an admin registered into the database with the same credentials as in the environments variables file.

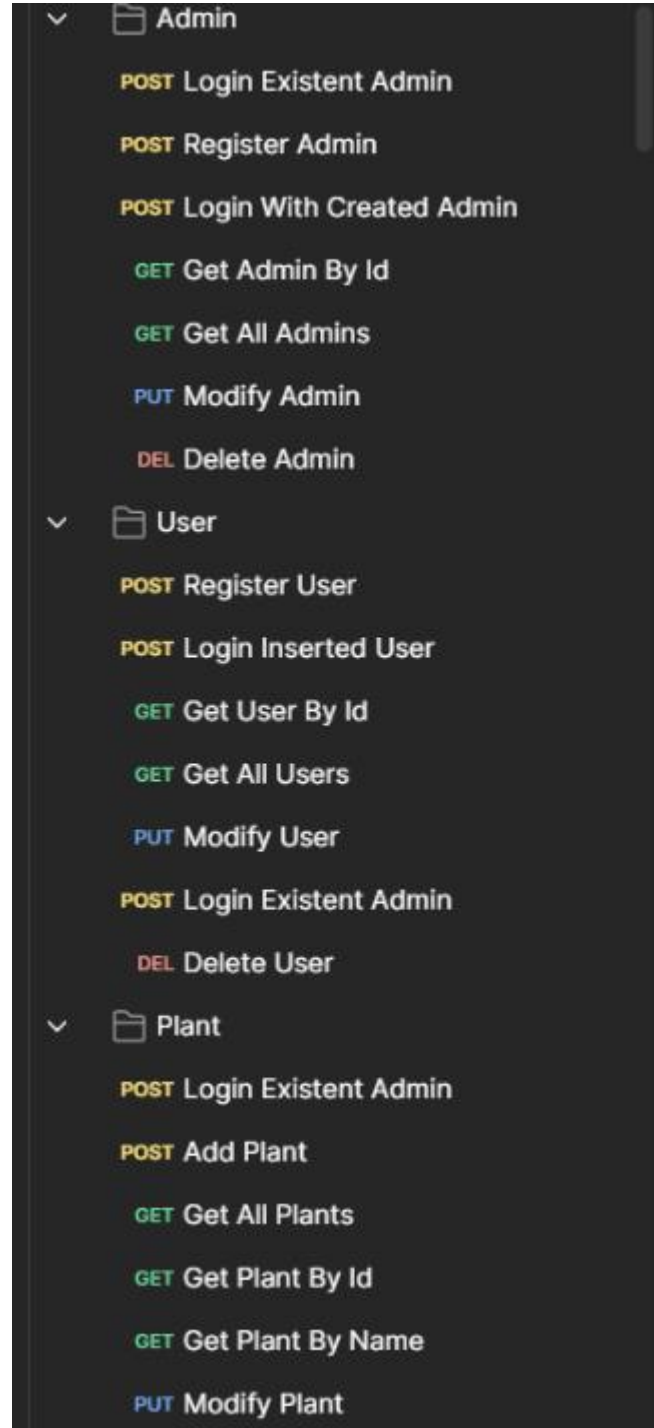


Figure 44 - snippet of postman requests

## 10. Proof of CI/CD

### 10.1 Introduction to CI/CD

Continuous integration and Continuous Deployment was used to streamline the process of building and testing our application, and any changes associated to it. This ensures that each update to the main app is automatically validated and deployed appropriately.

### 10.2 CI/CD Tools

**Dockerfile** — the dockerfile sets up a multi-stage build environment in the .net core project.~ It defines how the application is built, published and run in a contained environment, ensuring that it always runs in a stable and isolated environment.

It's composed of stages, **The base stage**, where we define the image for a runtime environment, in this case using .net core 8.0 asp.net.

The **Build stage**, where the .net sdk image is used to build the application. It'll copy the project file from our webAPI, install dependencies and then copy source code and then will build it. The build stage ensures that every time code changes, the pipeline can automatically compile the latest version in an isolated environment.

The **Publish Stage**, creates a self-contained set of files needed to run it. These can be found in /app/publish/. This stage produces optimized files for deployment. This step serves to make the final image smaller and faster to start up, by only keeping the necessary runtime files.

The **final Stage**, uses the base image to create a final runtime environment, copying the published files from the publish stage. This results in a smaller final image size by ignoring unnecessary build files and dependencies.

```
# Estágio base: Usado para rodar o container
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
#USER app
WORKDIR /app
EXPOSE 8080
EXPOSE 8081

# Estágio de build: Usado para compilar o projeto
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY["WebAPI/WebAPI.csproj", "WebAPI/"]
RUN dotnet restore "./WebAPI/WebAPI.csproj"
COPY . .
WORKDIR "/src/WebAPI"
RUN dotnet build "WebAPI.csproj" -c Release -o /app/build

# Estágio de publicação: Usado para gerar o artefato final que será usado no container de produção
FROM build AS publish
RUN dotnet publish "WebAPI.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=false

# estágio final: usado para rodar o container com a aplicação publicada
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "WebAPI.dll"]
```

Figure 45 - dockerfile

```
PS C:\Users\pedro\Documents\GitHub\lds_leafings\WebAPI> docker-compose up --build
time="2024-11-08T13:03:15Z" level=warning msg="C:\\Users\\pedro\\Documents\\GitHub\\lds_leafings\\WebAPI\\docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid
potential confusion"
[*] Building 0/7s (19/19) FINISHED
[+] Building 0.7s (19/19) FINISHED
=> [webapi internal] load build definition from Dockerfile
=> transferring dockerfile: 1.09kB
=> [webapi internal] load metadata for mcr.microsoft.com/dotnet/sdk:8.0
=> [webapi internal] load metadata for mcr.microsoft.com/dotnet/aspnet:8.0
=> [webapi internal] load .dockerignore
=> transferring context: 46kB
=> [webapi build 1/7] FROM mcr.microsoft.com/dotnet/sdk:8.0@sha256:ca0284cce7bc26d41855d9ac5859a69a8b75d9a201cd
=> [webapi internal] load build context
=> transferring context: 5.06kB
=> [webapi base 1/2] FROM mcr.microsoft.com/dotnet/aspnet:8.0
=> CACHED [webapi base 2/2] WORKDIR /app
=> CACHED [webapi final 1/2] WORKDIR /app
=> CACHED [webapi build 2/7] WORKDIR /src
=> CACHED [webapi build 3/7] COPY [WebAPI/WebAPI.csproj, WebAPI/]
=> CACHED [webapi build 4/7] RUN dotnet restore "WebAPI/WebAPI.csproj"
=> CACHED [webapi build 5/7] COPY .
=> CACHED [webapi build 6/7] WORKDIR /src/WebAPI
=> CACHED [webapi build 7/7] RUN dotnet build "WebAPI.csproj" -c Release -o /app/build
=> CACHED [webapi publish 1/1] RUN dotnet publish "WebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=false
=> CACHED [webapi final 2/2] COPY --from=publish /app/publish .
=> [webapi] exporting to image
=> exporting layers
=> writing image sha256:9afd68baf6e1bf2e3caf8427155398639bb861e2f2dce25d40f5fa317884dde
=> naming to docker.io/library/webapi-webapi
=> [webapi] resolving provenance for metadata file
[*] Running 1/0
Container webapi-webapi-1 Created
Attaching to webapi-1
webapi-1 | The command could not be loaded, possibly because:
webapi-1 | * You intended to execute a .NET application:
webapi-1 | * The application 'WebAPI.dll' does not exist.
webapi-1 | * You intended to execute a .NET SDK command:
webapi-1 | * No .NET SDKs were found.
webapi-1 |
webapi-1 | Download a .NET SDK:
webapi-1 | https://aka.ms/dotnet/download
webapi-1 |
webapi-1 | Learn about SDK resolution:
webapi-1 | https://aka.ms/dotnet/sdk-not-found
```

Figure 46 - docker creation

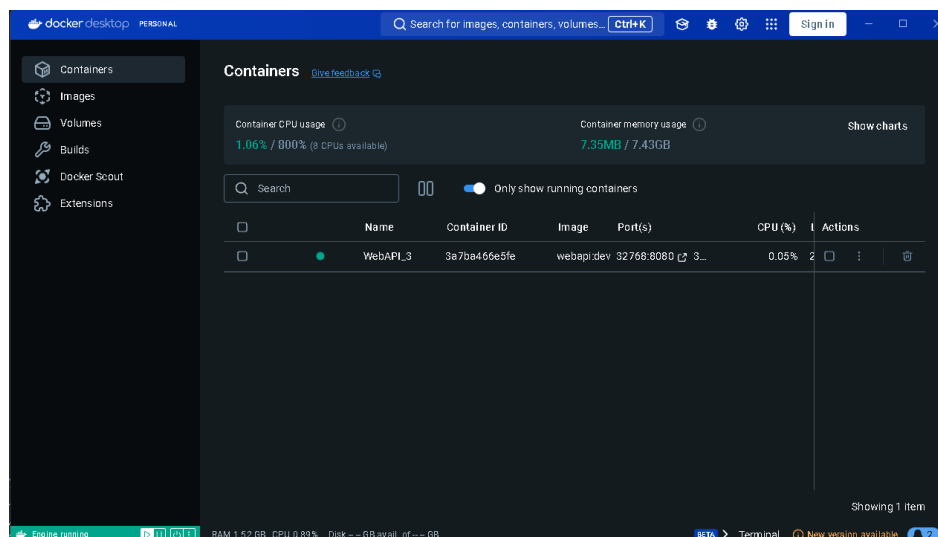


Figure 47 - docker running

## Pipeline –

Our CI/CD pipeline, can be found in .gitlab-ci.yml, includes two stages, Build and Test.  
**Build** compiles the application code to ensure it builds successfully.  
**Test** runs all the unit tests to verify that the application functions with all the tests in mind.  
Our pipeline uses the .net sdk image provided by Microsoft to run said stages.



Status	Pipeline	Created by	Stages
<div><div>Passed</div><div>00:01:34</div><div>6 days ago</div></div>	<div>Update .gitlab-ci.yml</div> <div>#1523597444 1 94938f86</div> <div>latest merge request</div>		<div><div>✓</div><div>✓</div></div> <div></div>

Figure 48 - example of pipeline passed

```
stages:
  - build
  - test

# define que a pipeline só vai executar quando for um merge
.restrictions:
  rules:
    #- if: '$CI_PIPELINE_SOURCE == "push"' # temporário, para testes
      #when: always
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"' # só executa quando for um merge
      when: always

.restrictions_dotnet:
  image: mcr.microsoft.com/dotnet/sdk:8.0
  before_script:
    - cd WebAPI
  extends: .restrictions

dotnet-build:
  stage: build
  script:
    - dotnet restore # restaura as dependências do projeto
    - dotnet build
  extends: .restrictions_dotnet
  tags:
    - dockerRunner

dotnet-tests:
  stage: test
  script:
    - dotnet restore
    - dotnet test --results-directory "test-results" --collect:"Code Coverage;Format=cobertura"
  artifacts:
    paths:
      - "**/test-results/**/*.xml" # vai recolher qualquer xml numa pasta "test-results"
    expire_in: 1 week
  extends: .restrictions_dotnet
  tags:
    - dockerRunner
```

Figure 49- pipeline configuration

## 11. Sprint Backlogs

### 11.1 Sprint Backlogs since milestone3

Since milestone3 we have done three sprints, we averaged between 20-40 story points depending on the sprint. The last sprint was lower on workload to ensure we had everything correct from previous sprints and ready for delivery.

#### 19/10 – 26/10 –

This sprint was marked by the start of the Implementation of the basics for the webAPI, Mapping out tables and entities using EF, creating Classes and implementing simple crud operations.

Most of the user stories implemented throughout This sprint regard operations like “creating ads “ or “I want to write in my plant diary (creating Logs)”.

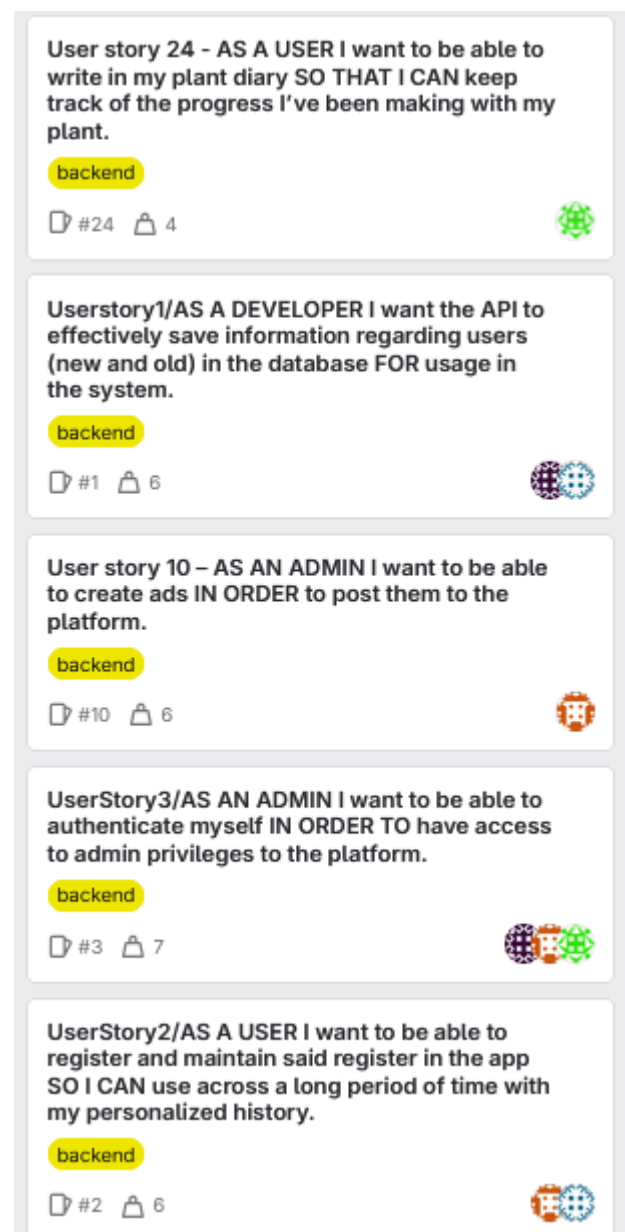


Figure 50 - sprint backlog 19/10-26/11

## 26/10 – 02/11 –

This sprint followed the footsteps of the previous Ones, we finished crud implementation and started Implementing more complex operations like:

- . Verifying that when a user adds a plant, Are they a free or paid user? If they're free, they Shouldn't be allowed to possess more than 3 user-Plants.
- . The matchmaking algorithm between users And plants depending on their profiles.
- . Authentication.

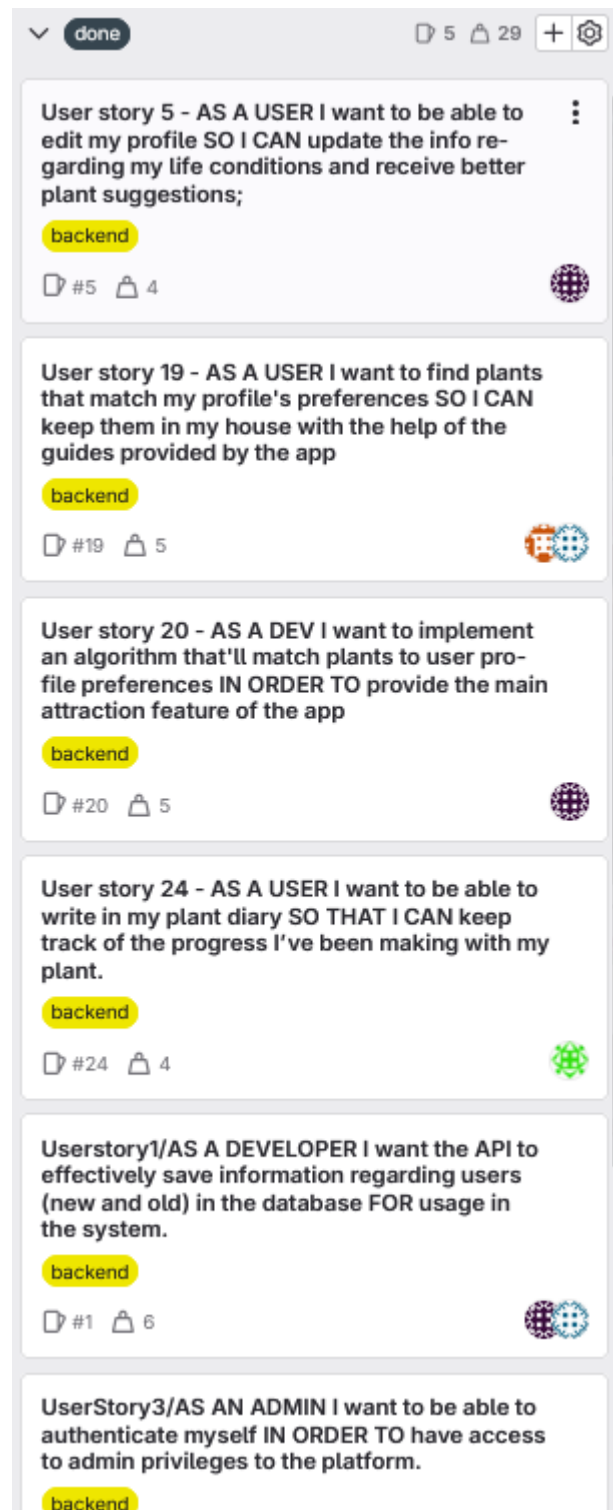


Figure 51 - sprint backlog 26/10-02/11

## 02/11 – 09/11 –

This sprint was focused on tweaking any problems Left from previous sprints, implementing easier Functionalities like search filters and name matching.

All the meeting minutes and meeting summons can be Found alongside the delivery files.

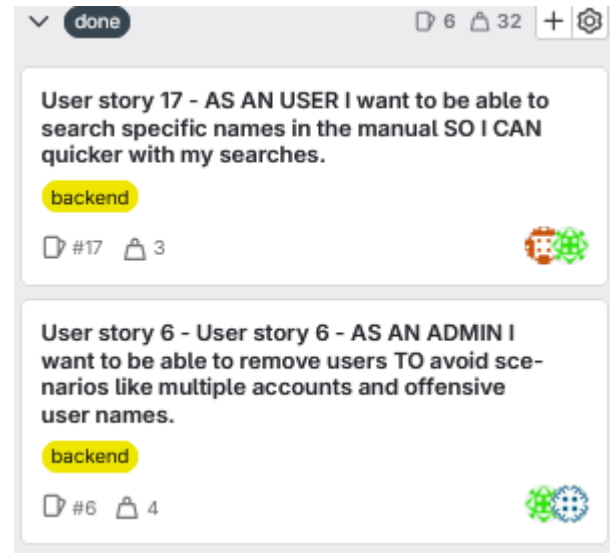


Figure 52- sprint backlog 02/11-09/11

## 17/11 – 26/11 –

There's a lack of a sprint between the 9<sup>th</sup> and 16<sup>th</sup> Because the group decided to focus on the 1<sup>st</sup> milestone For the Course Unit of mobile development (CMU).

This Sprint marked the development of the frontend, Both in the mobile and web browser apps. The group Was divided by 2 members for each frontend. The Backoffice was implemented in this sprint as well.

As mentioned before, the frontend for the web browser Was developed in react, while the mobile version was Built in react.

We implemented the barebones of the frontend, going Through the principal pages, like the dashboard with The plant manual, the display components for plant Information etc.

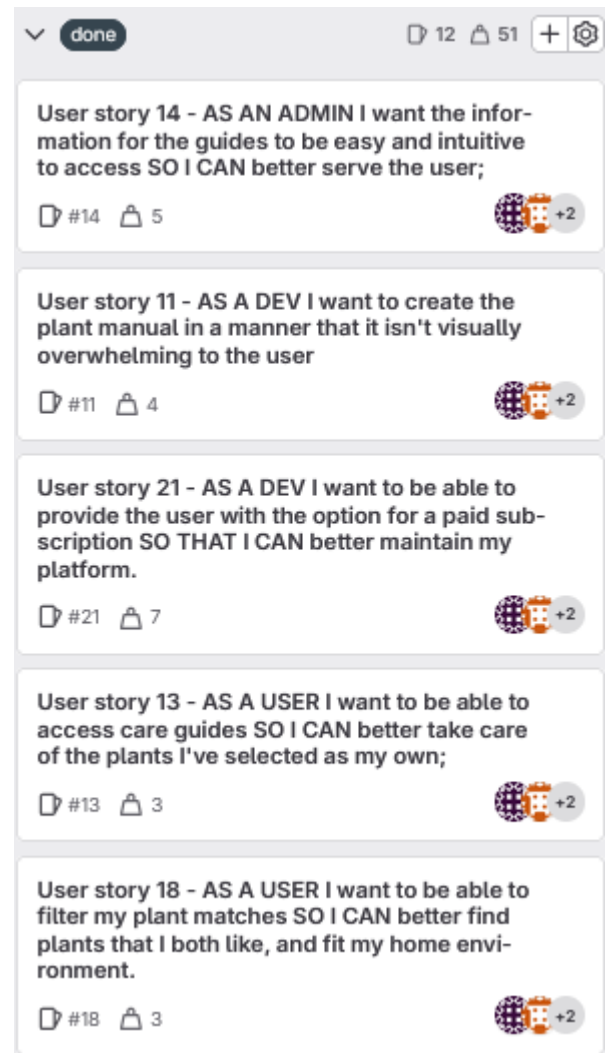


Figure 53- sprint backlog 17/11-26/11



**26/11 – 03/12 –**

At the last sprint the group focused on the adjacent Features to the core of the application.

We implemented user profile pages and the “my garden” Page, where a user is able to manage their personal plants.

We also implemented the diary feature, along with Displaying tasks for plants. Users are also capable of Removing and writing into their plant diaries through Logs.

We also implement the capability of paying for a Premium option in the web browser for the user side.

User story 7 - AS A USER I want to be able to access the plant manual SO I CAN discover new plants, and check information I find important;

 #7  3

User story 8 - AS A DEV I want to create the plant manual in a visually appealing manner for both the mobile and web browser apps IN ORDER TO make the final product more attractive to the user;

 #8  4


User story 12 - AS AN ADMIN I want the system to force me to fully complete the form to add new plants to the platform IN ORDER TO avoid half registries in the final product plant manual;

 #12  4

User story 15 - AS AN ADMIN I want to be able to edit the info relative to plants and their guides so I can keep the information in the platform up-to-date;

 #15  4

User story 23 - AS A USER I want to be able to save the plants that I find relevant to me, IN ORDER TO have their information available to me at moment's notice.

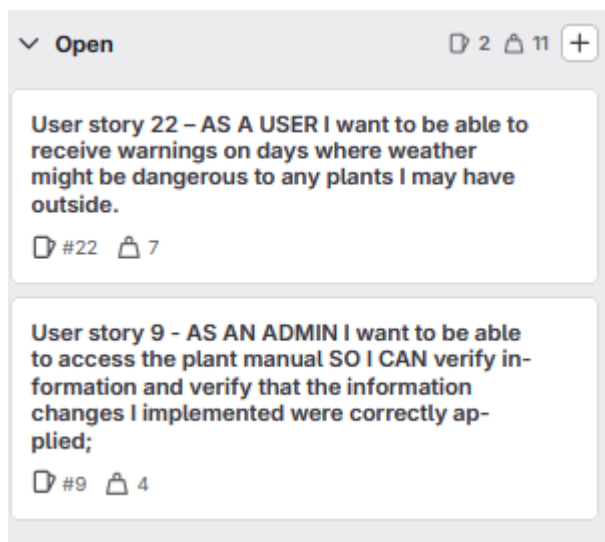
 #23  4

User story 25 - AS A USER I want to have access to multiple plants in my garden SO THAT I can keep multiple plants at home.

 #25  5

*Figure 54-sprint backlog 26/11-02/12*

**Remaining backlog** — Although we tried our best, there was a feature that's missing from the product scope is the warning feature. We had intended for users to receive warning notifications based on their location for adverse weather conditions but we couldn't find a good API to integrate, and it ended up being sent to the back of the product backlog each sprint. In the end we ended up not implementing it.



## 12. Online Resources

In this section we will be giving resources to consult the solution online and login credentials for both front and backend on the web browser for the sake of testing functionalities. We used Azure to host both the API and the solution.

**Due to an oversight during the development phase, we are unable to host live images in the online version of the solution. This is due the fact that we were saving the images locally within a file in the API and believed we could do the same once we had hosted it with azure. This was untrue, we'd have to create a files server or place the API within a server. Because of time constraints we were unable to implement this.**

### 12.1 API

The API is hosted at link :

<https://leaflingsapi-gfgeehhqa3edcgbv.spaincentral-01.azurewebsites.net/swagger/index.html>

### 12.2 Frontend for Webbrowser

The frontend can be used by users to access multiple functionalities, from the dashboard, where they're able to consult all the plants available in the application, filter the plants through multiple button filters and search bar, get their plant matches based on their profile information and the plants various information.

The Profile, where they're able to changed their information and preferences at their will.

The Premium page, where they're able to use PayPal to do a one-time payment to the platform, to be able to save more than 3 plants to their garden page, and to remove ads from the platform.

The My garden page, where they're able to consult the plants, they've selected and chosen to take care of, where they can create diaries to keep track of their plants progress, and where they're able to consult the tasks for the plants they chose to care for.

The Client app can be accessed at:

<https://leaflingsfrontoffice-f5crehbfcbcdf9ax.spaincentral-01.azurewebsites.net/>

some credentials for logging in and testing can be found here:

Paypal Dummy account (for testing the PayPal payment function):

Email - sb-vbova33718726@personal.example.com

password – v+D8D?R?

Free user:

email - joao@joao.com

password – joaojoao

Paid user:

Email – [joao2@joao.com](mailto:joao2@joao.com)

Password – joaojoao2

### **12.3 Backend for Webbrowser**

The backend can be used by platform admins to create and delete all kinds of entities, from plants, to ads, plants tasks and deleting users. **Be wary to not test create plants or ads with images since it will not work.**

The admin app can be accessed at:

<https://leafingsbackoffice-g3htcmakfcg0gfeh.spaincentral-01.azurewebsites.net/>

some credentials for logging in and testing can be found here:

email - [admin@admin.com](mailto:admin@admin.com)

password - adminPassword

## **13. Conclusion**

Concluding our leafings projects, we were aiming to create an application that would serve all users in their quest for finding and maintaining house plants.

As a group, we believe we have achieved this goal. We managed, as a team to implement a full application, from the database to the Backoffice, to the user frontend, that provides the principal functionalities we had planned. We implemented an interface that intuitively presents users with plants that might come off as interesting to them; Users are then able to filter, search and customize their results in said interface through multiple filter buttons and a search bar option.

We implemented an algorithm that'll match plants based on their need to users. We allow users to Add plants to their profiles, making a link between a plant and a user; We allowed the users to write on their plant's records through diaries, that are customizable.

We've implemented paid and free profiles, created ad mocks that are shown to free users for the sake of realism.

Through this process we had a lot of challenges and hurdles we had to go through, mainly learning the languages that we used to implement the solution; We believe that the hardest part of the project was hitting the balance between all the languages and frameworks we had to use to implement multiple parts of the project, from the API, to the webapp, to the mobile version to the testing frameworks such as vitest. We had never done frontend tests so that was also part of the learning process.

We believe through the implementation of this project we have learned to work better as a team, using the SCRUM methodology and mixing in a bit of unorthodox organizing since every member of the group 10 has known each other since 1<sup>st</sup> year and we're pretty well organized as a team by now. We managed to improve our knowledge of the implementation process, from the design to the making phase, and we have also evolved in our CI/CD skills, in our programming skills and our communication skills.

We would've liked to have implemented a warning system, but we believe that we truly didn't have the time to do so due to the vast number of projects required this semester. However, we believe we did sufficient work and are proud of our results.