**Software Design Document Template**

|  |  |
| --- | --- |
| Project Name: RaspberryPlant | Team ID: 29 |
| Team Members:Mayank Thakur, Chris Bosman, Louisa Hafferl, Meenakshi Girish Nair, Vithursika Vinasiththamby, Gyum Cho | Mentor(s): Priya Naguine and Radu Basarabá |

1. **Introduction**

Our application aims at helping plant enthusiasts take care of their plant from afar. The sensors in our application monitors the factors that affect the plant and give advice to the user on how to maintain a healthy plant. Our application also has an automatic plant watering system which waters the plant on the interval specified by the user. An addition to the functional requirement is: The amount of water used in watering should be customizable.

1. **Functional/Non Functional Requirements**

The functional and non-functional requirements are mostly the same as the ones we had in Sprint 1. However, we have decided to change the priorities of some of them taking into consideration the time we have. These changes are highlighted below.

- I)” The functional requirements of a Raspberry Plant are:

* **[H]** The access from the user will be protected.
* **[H]** The system will collect the environment data and store in database, the data will be encrypted.
* **[H]** The system will notify user about changes to make in the plant’s environment (wrong temperature, wrong amount of sunlight)
* **[M]** The system will notify to user whether the water pump is empty
* **[M]** The system will support out-door mode by getting GPS signal of user.
* **[L]** The system will hydrate the plant by sensing dry condition.

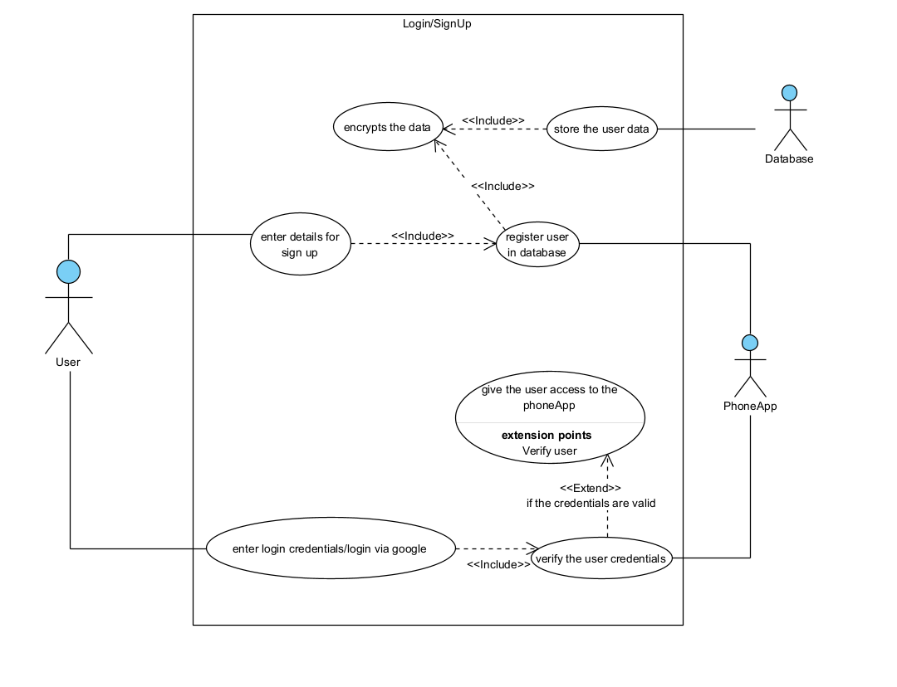
New functional requirements:

* **[H]** The amount of water used in watering should be customizable.

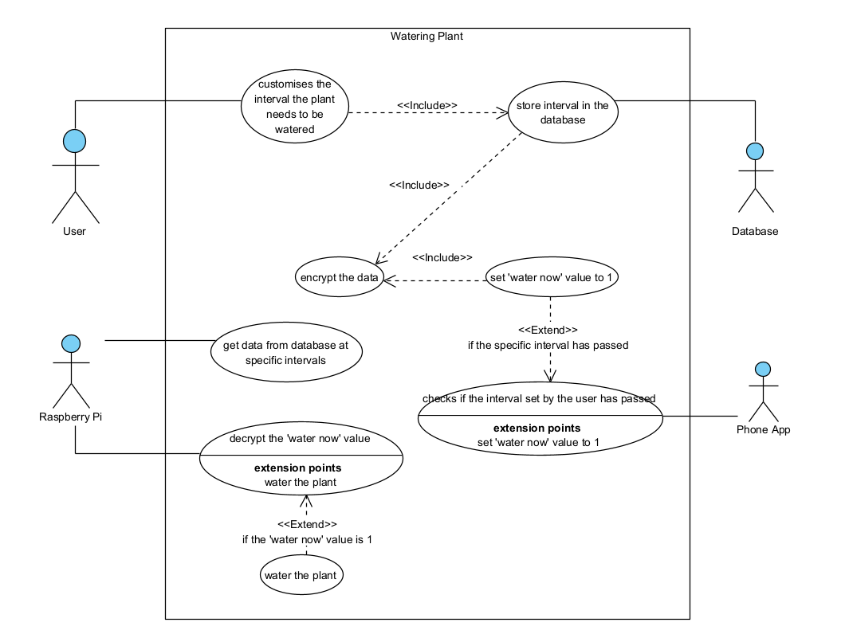
- II)” The non-functional requirements of a Raspberry Plant are:

* **[H]** The system should use user-friendly GUI for user’s comfortable use.
* **[H]** The system should not be loud than 50DB.
* **[H]** The system should have a secure database
* **[H]** The system should have secure data transfer
* **[H]** The success/fail rate of system will be less than 5 percent.
* **[M]** The application should allow more than 20 users simultaneously.
* **[M]** The application design will be fit with each user environment (mobile, pc, and tablet).
* **[M]** The GUI will support the night mode for comfortable user experience.
* **[M]** System should be flexible for add-ons.
* **[M]** The system can be easily implemented in updated environments.
* **[M]** The time to restore a system after a crash should not be more than 15 minutes
* **[M]** The app must be available for the user for more than 95% of the time

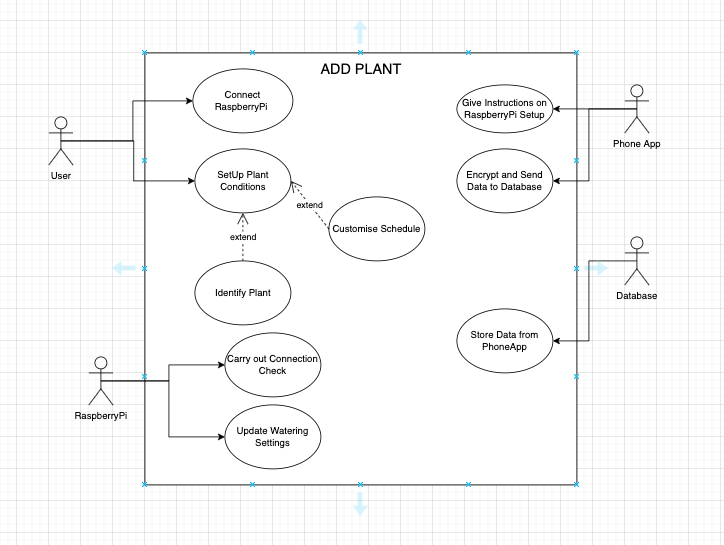
1. **Architectural Design**

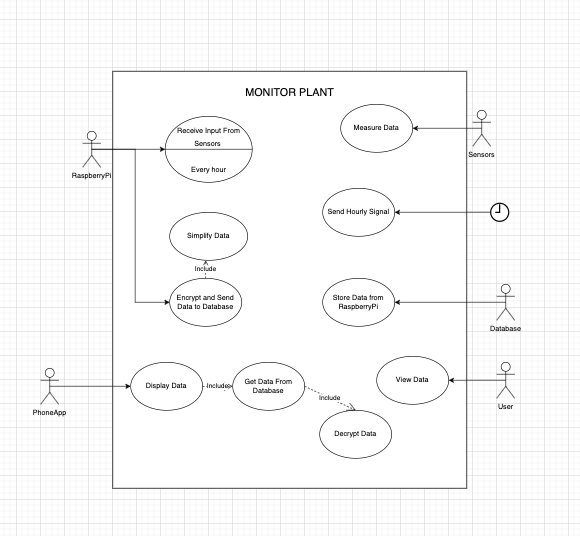
**Use case**

The use case diagram shown above depicts the processes that occurs when a user wants to log in or sign up. To sign up, the user enters their details and this is stored in the database. All data stored in the database are encrypted. When the user wants to log in, the user inputs their credentials and these credentials are then verified with the database values by the phone app.

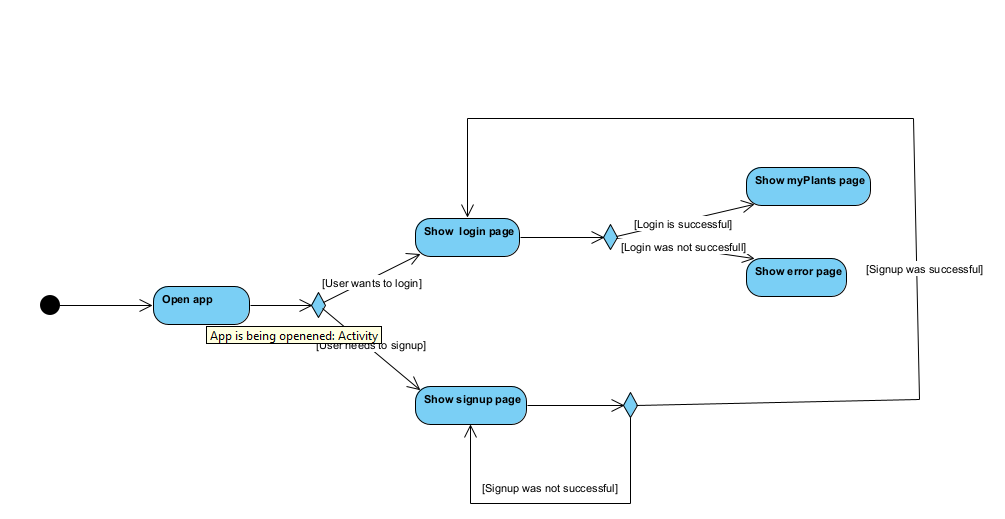


This use case diagram shows the processes the actors carry out when watering the plant. The user has the option to customize the intervals at which the plant needs to be watered. This data is stored in the database, also encrypted. The phone app gets this value from the database and monitors if the interval has passed. There is a ‘water now’ variable in the database, which denotes that the plant needs to be watered when the value is 1. When the interval set by the user has passed, the phone app sets the ‘water now’ value in the database to 1. The Raspberry Pi reads the ‘water now’ value in the database and decrypts it at specific intervals. When it reads the value 1, it waters the plant.

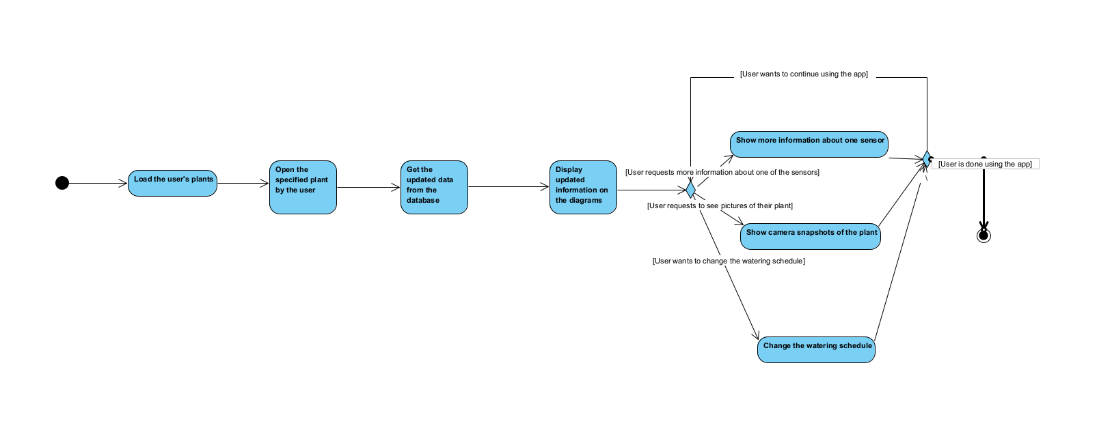


This use case diagram shows the processes the actors carry out when a new plant is added to the user’s account on the app. The phone app gives instructions to the user for setting up the Raspberry Pi, and gives the user the ability to Set Up the Plant Conditions, with optional choices of identifying the plant and customizing the watering schedule. A connection check is carried out by the Raspberry Pi, and the watering schedule information is encrypted by the app and sent to the Database, from which the Raspberry Pi checks the information and updates it’s settings.

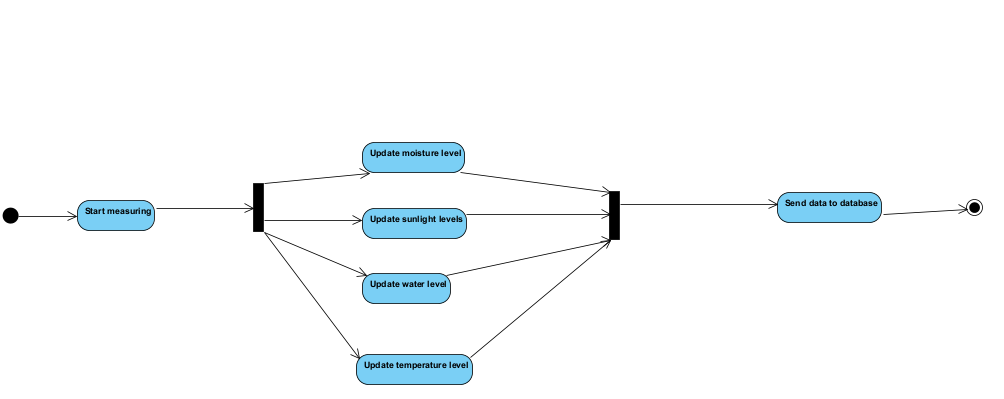
This use case diagram shows the processes the actors carry out in order to monitor the plant. The sensors measure data, which the Raspberry Pi receives in one hour intervals. It simplifies, encrypts and sends this data to the Database. The Database stores this data, which the Phone App accesses and decrypts in order to display in the app, so that the User can view it.

**Activity**

This activity diagram explains how to login and signup works for the user. Where the user can choose whether they want to sign up or login. When they choose to login and this turns out to be successful, they will be led to the plants page. Otherwise they will have to try again. The signup process is similar and requires the user to succesfully login after before they can enter the login page.

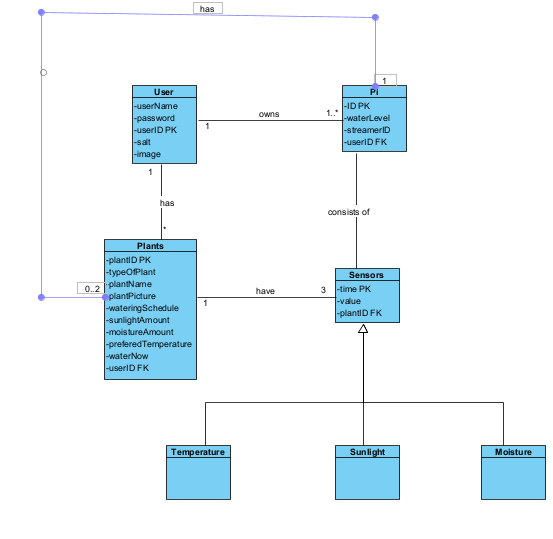


This diagram shows from the app's the perspective the actions that need to be done when the user successfully logs in, such as showing all the user's plants and giving them the chance to look into their plant's health and change the water scheduling if they prefer to.

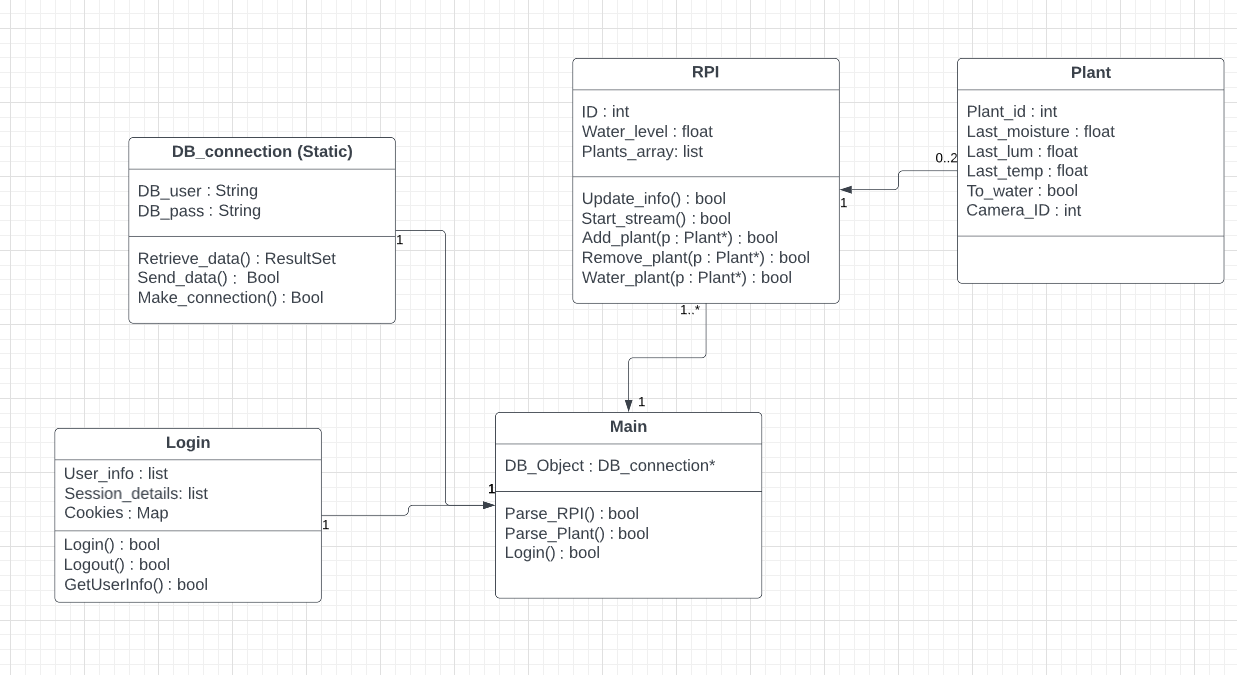


This diagram shows the activities from the point of view of the Pi. When it needs to start measuring, it will do four activities in parallel and then send the updated data to the database.

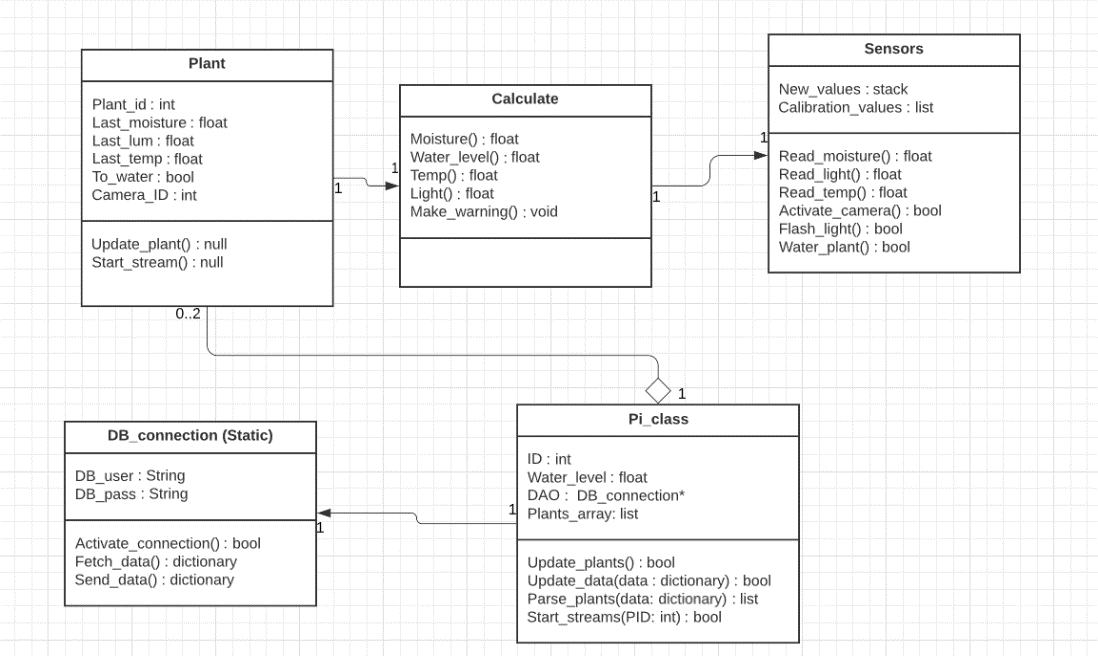
**Class diagram**



Our database will have 6 tables, which will have the above mentioned values. We have made sure that every class has a primary key and a foreign key that connects the table to a different table. On top of that, we have decided to make different tables for temperature, sunlight and moisture values such that it is easier to get specific information for each of these values.

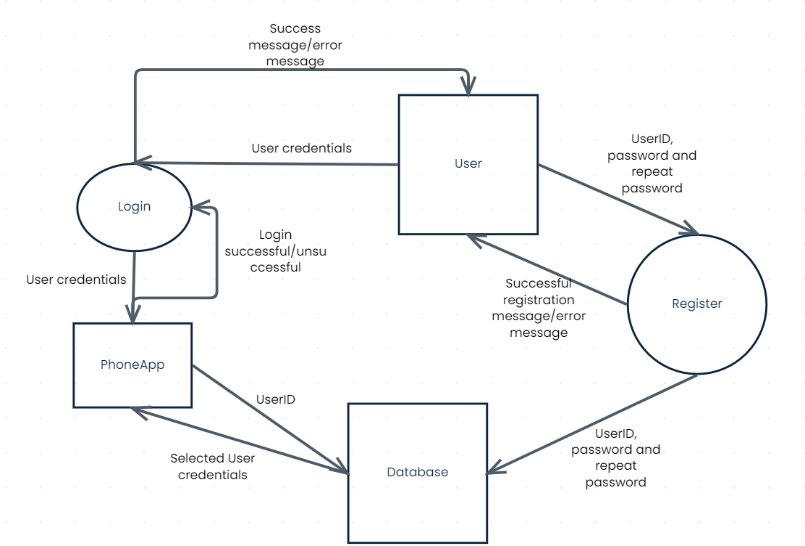


The application will not have many classes. The classes that we do have are mostly for the basic functionality that the app will serve, and the rest are to model the data that we already have on the database. This is because a large portion of the app is just showing data and editing it.



Since the PI in our project is essentially used as a data collection tool, we only need a few classes to model the data that is being collected in relation to how it will be used, and then a few classes to communicate with the database.

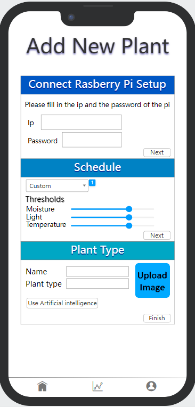
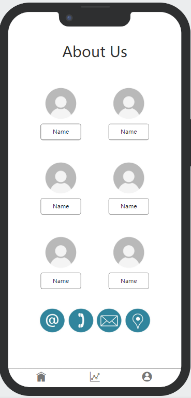
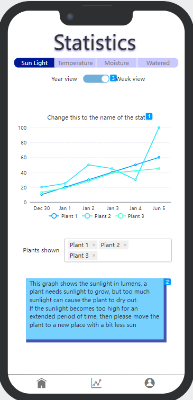
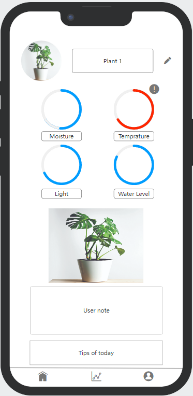
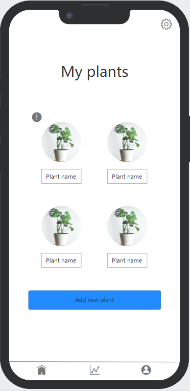
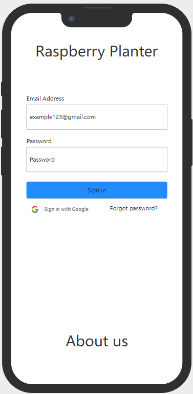
**Data flow**



This dataflow diagram depicts how data flows between the database, user and phone app during the login and sign up process. During login, the user enters their credentials to the phone app and these credentials are then verified with the data in database. The phone app sends the UserID to the database and the database returns the user credentials to the phone app so that the data can be verified. Once verified, the phone app sends a login successful/unsuccessful message. During sign up, the user enters the userID, password and repeated password on the phone and these are then sent to the database to store. If registered correctly, the user receives a successful registration message. Otherwise, they receive an error message.

1. **Product User Interface**

[*https://rp.mockplus.com/run/qo6PQXnEdy/Gk10dms8N9?cps=expand&rps=expand&nav=1&ha=0&la=0&fc=0&dt=iphoneX&out=0&rt=1*](https://rp.mockplus.com/run/qo6PQXnEdy/Gk10dms8N9?cps=expand&rps=expand&nav=1&ha=0&la=0&fc=0&dt=iphoneX&out=0&rt=1)



* + Start with Login page(optional google login, sign up button -> sign up page)
  + My plants (all of the plants that the user has). Add new plant button for adding plants.
  + My plant page(Camera, information about my plant: last water time, current light, current moisture, any warnings)
  + Stats pages to show the statistic information of the own plant
  + About us page, contact information about team mates.
  + Add new plants page, fill up the form and create a new plant page.
  + Each pages, the bottom buttons will bring users to home, statistic, and about us page.

1. **Prevention/Mitigation Criteria (Security Controls)**
   1. File structure
      * Database credentials will be stored in pickle files, and the code files will be password protected
   2. Input Sanitation

* In order to prevent any SQL injections, we will use prepared statement every time we need to access the database so that no input given by the user can cause harm to the database. The users will not have the privilege of editing data in the database.
  1. File Encryption (for passwords and other hidden items)
* Any credentials that are stored in the code will be encrypted in a file so that in case anyone has access to the code, they still won't be able to access the credentials.
  1. Password Hashing (salt + pepper)
* Any password that is stored in the database will be hashed using salt and pepper (generated using a secure random number generator) and an encoding algorithm (SHA512) so that no malicious users will be able to decode them.

For detailed explanations of how these security measures apply to our project, please refer to the SDC.

1. **The cost involved (if any):**

Apart from the costs of procuring the hardware to implement the security on (Raspberry Pi), there are no financial costs involved with implementing our security controls, as we are not buying any equipment or access to software. It will take us some time to implement the hashing and user sanitization on both sides of the application. However, since this is something that we are used to from the last module, it should not take that much time. Specifically, we predict a time cost of: half a day for the File Structure implementation, 1 day for the Input Sanitation, 1 day for the File Encryption and 2 days for the Password Hashing.

1. **Conclusion:**

A noteworthy design decisions of our application is that the phone app has a very user friendly interface where the user is alerted if any factors go above or below their optimal level. The user can also select one or more plants to see its graph of a certain factor (example: temperature, light intensity, moisture). l. The most challenging part of the design phase was the creation of class diagram and deciding how to water the plant within each interval set by the user as we were confused whether the time needs to be monitored on the raspberry pi or on the phone app. Challenges for the next phase would be getting the sensors to work properly and getting the communication between the raspberry pi, database and the phone app working.