CSC 431

Plant Power

System Architecture Specification (SAS)

Team 15

Jorge Jaime-Rivera	Hydroponics Engineer
Dylan Aron	Full-Stack Engineer
Dominick Bello	Full-Stack Engineer

Version History

Version	Date	Author(s)	Change Comments
1	3/25/21	Dylan,Jorge,Dom	First Draft
2	4/12/21	Dylan,Jorge,Dom	Fixed issues commented
3	5/4/21	Dylan, Jorge, Dom	Fixed issues commented by the grader

Table of Contents

Table of Figures

1. System Analysis

1.1 System Overview

This system is intended to be used for data visualization of plant conditions, which can then be used to predict optimal conditions. It contains the following five components: a data collector, data manager, data analyzer, validator, and plant webpage.

The first component is the data collector. The data collector is the Arduino board with the sensor shield attached. The data collector is connected to the Arduino create environment and logs initial data from the sensors through the provided platform.

The data manager takes the sensor data from the Arduino create environment to our database powered by SQL.

The data analyzer consists of python code that will generate graphs for data visualization and determine the optimal conditions for each plant.

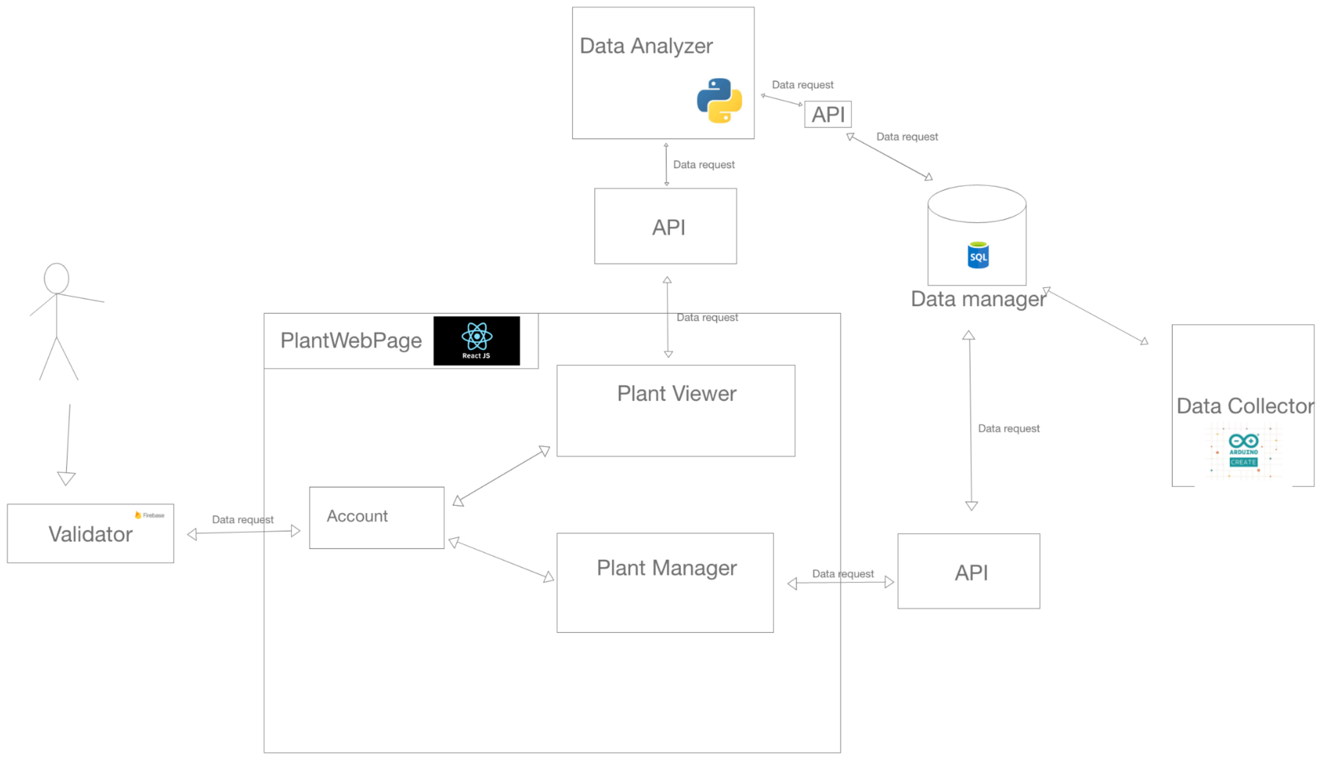
The validator uses Firebase to host our website server and manage account information, so that each user sees only the data corresponding to their own personal data collector.

The final component is the plant webpage. This is the user interface for the system, and contains a Plant Viewer section and Plant Manager section. The Plant Viewer section takes the data visualization from the data analyzer and displays it on the webpage for the user to view it, while the Plant Manager allows the user to add or delete plant profiles, which generates a new ID in the data manager, or deletes an existing entry. The plant webpage contains an account page that pulls the user's ID and information from the account manager.

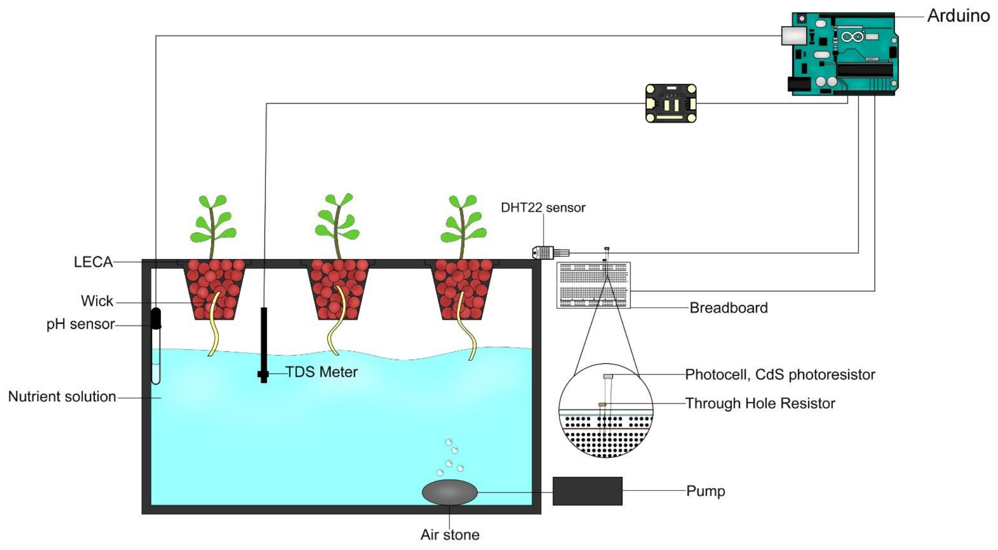
The following pipeline displays the interactions between the system when a user wants to see their data: (1) The user opens the webpage, (2) the webpage pulls the user's ID and information from the validator, (3) user selects “Plant Viewer”, (4) the data analyzer processes information pulled from the data manager, (5) data analyzer displays data visualization onto webpage.

The architecture used is the Event-driven style, where the system reacts to external events and then communicates using events. The first event is the user logging in. This event communicates with the validator component to verify the user's information. The following event is the user setting up their data collector. At this point the data collector takes data on the plant itself using attached sensors. All the events that follow from this feed into different components, such as processing the data through the data manager.

1.2 System Diagram



1.2.1 Growing Structure Hardware



1.3 Actor Identification

There is one main human actor which is the user. The user may be an at home grower, any person who wishes to make their at home gardening and production more efficient. At a larger scale, the user could be a hydroponic production company, which is a smart farm company using indoor agriculture to grow produce. Some examples of current hydroponic production companies include Aerofarms, Plenty farms, and Click and Grow. The user can initiate actions such as logging in, viewing plants, and editing existing plants. The main non-human actor is the Arduino microcontroller which controls the sensors and the plant's growing environment. The Arduino can initiate actions such as collecting data from the plant, and sending data to the database. Another non-human actor is Skyvia's SQL Web API, which will be able to initiate its main action to transfer data from the SQL server, our data magner, to the PlantWebPage component.

1.4 Design Rationale

1.4.1 Architectural Style

Our system uses the event-driven architecture style, as this style is used where the system reacts to external events and then communicates using events. The event driven style is especially useful when dealing with a high volume of data and using the Internet of Things (IoT), which our system does. The first event that begins to use the system is the user logging in. This event communicates with the validator component which employs Firebase to verify the user's information. The following event necessary to continue use is to set up the data collector and connect it to the user's account, which is done by the Plant Manager section of the web page component. The data collector will then begin to take data on the plant using the equipped sensors. Once these two events are completed, the system continues to communicate using events, such as processing the data from the data collector using the data manager, or displaying the data through user-friendly graphs and data visualization using the data analyzer.

1.4.2 Design Pattern(s)

Our design pattern to be used will be the Command design pattern. We chose this design pattern because it is a behavioral design pattern. Behavioral design patterns simplify communication between objects by standardizing messages and data. Our application utilizes a lot of data and messages between the user and the data, so this is why we need a behavioral design pattern.

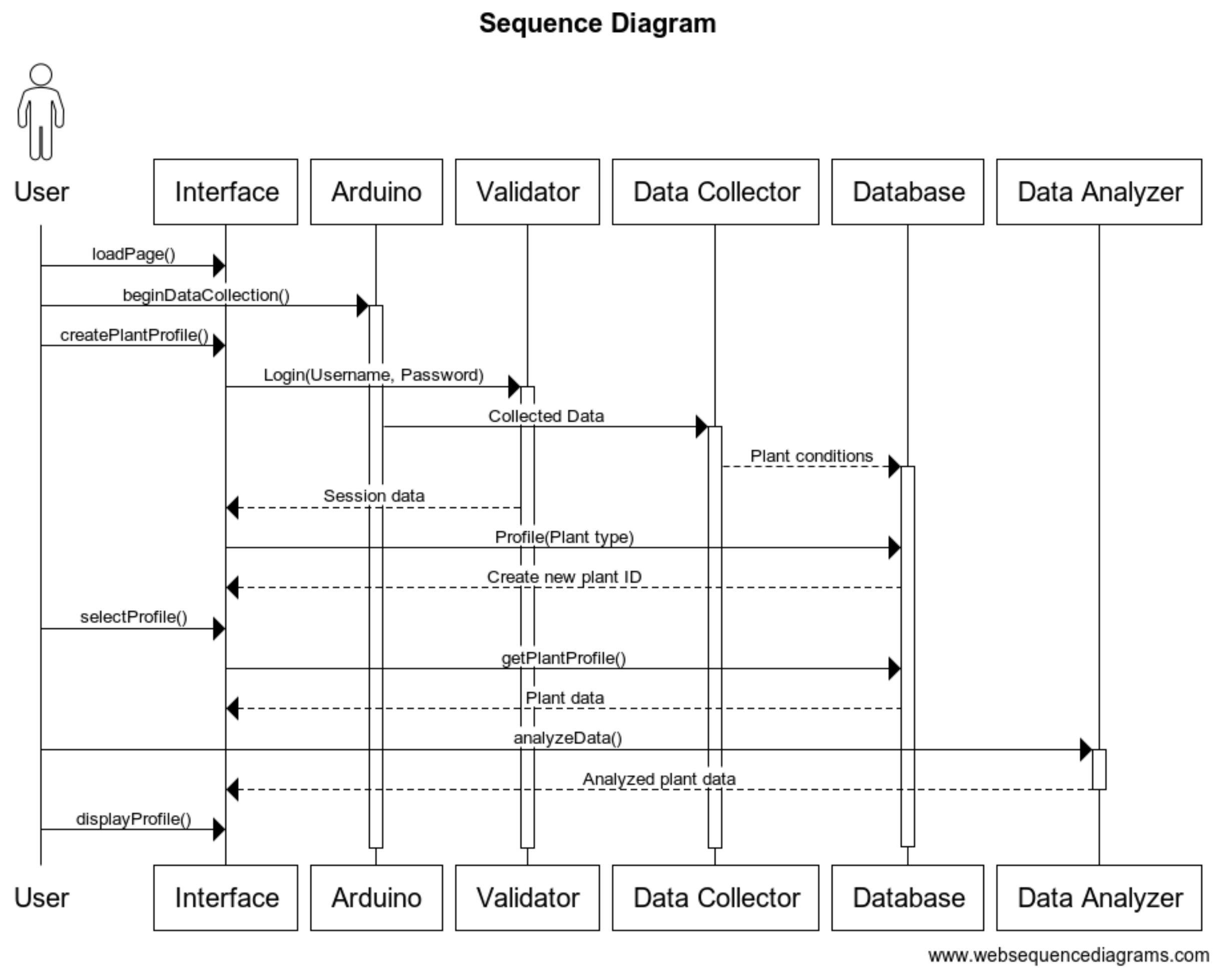
We specifically chose the Command pattern because it encapsulates command requests letting you parametrize the clients with different requests. Our application will have different user IDs and different sets of data, but they will be utilizing the same commands within our application, so the Command pattern makes sense for our plant automation application.

1.4.3 Framework

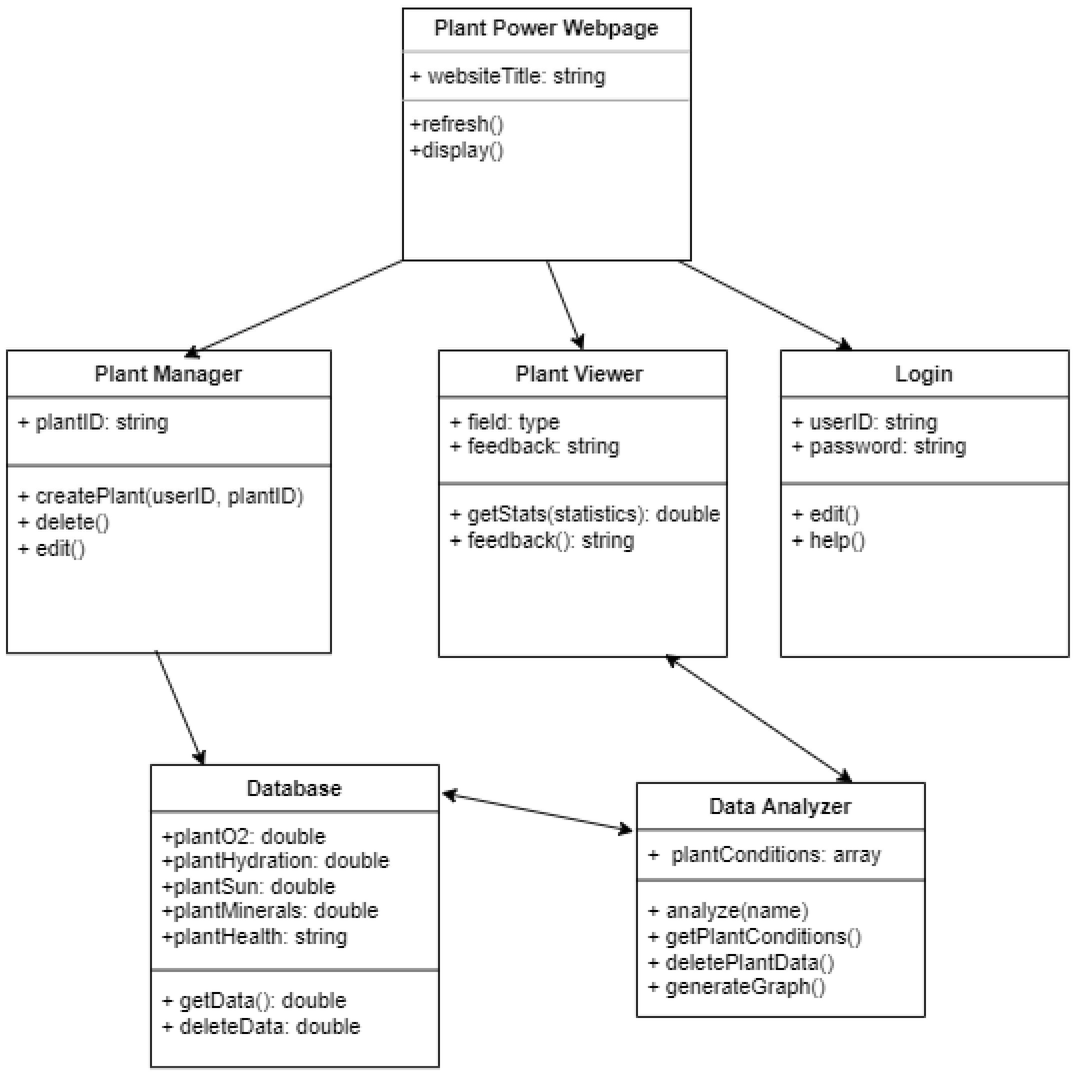
The framework we are using is React. We chose React because it dynamically presents different views to the user based on state information. It is highly scalable and portable. In addition, it lets us reuse components and cut down on the amount of code we have to write.

2. Functional Design

2.1 User Interaction



3. Structural Design


Plant Power Webpage
+ websiteTitle: string
+refresh()
+display()
Plant Manager
+ plantID: string
+ createPlant(userID, plantID)
+ delete()
+ edit()
Plant Viewer
+ field: type
+ feedback: string
+ getStats(statistics): double
+ feedback(): string
Login
+ userID: string
+ password: string
+ edit()
+ help()
Database
+plantO2: double
+plantHydration: double
+plantSun: double
+plantMinerals: double
+plantHealth: string
+ getData(): double
+ deleteData: double
Data Analyzer
+ plantConditions: array
+ analyze(name)
+ getPlantConditions()
+ deletePlantData()
+ generateGraph()