Pocket 'Ponics

Semi-Automated Microscale Greenhouse Leveraging Hydroponic Technology

Senior Design Interdisciplinary Project

Department of Electrical Engineering and Department of Computer Science

University of Central Florida

Elli Howard	Computer Science	ehoward@knights.ucf.edu
Rohan Patel	Computer Science	rohanpatel5753@knights.ucf.edu
Catherine Abbruzzese	Computer Science	abbruzzese@knights.ucf.edu
Alex Cusell	Computer Science	alexandracusell@knights.ucf.edu
Matthew Bonsignore	Computer Engineering	matthewB0217@knights.ucf.edu
Graham Hill	Electrical Engineering	grahamlhill@knights.ucf.edu

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ADA	American's with Disabilities Act
API	Application Program Interface
ARM	Advanced RISC Machine
AWS	Amazon Web Services
ECE	Electrical and Computer Engineering
FAO	Food and Agriculture Organization
GUI	Graphical User Interface
LAN	Local Area Network
LED	Light Emitting Diode
MCU	Microcontroller Unit
PCB	Printed Circuit Board
RDS	Relational Database Service
REST	Representational State Transfer
SBC	Single Board Computer
SOC	System on Chip
SPI	Serial Peripheral Interface
TDS	Total Dissolved Solids
UART	Universal Asynchronous Receiver/Transmitter
USPTO	US Patent and Trademark Office
WCAG	Web Content Accessibility Guidelines

1.0 - Executive Summary

Despite the rapid growth of agriculture in the past half a century, many people around the globe are still facing food shortages. The people often have difficulty finding enough food to meet their nutrient needs, and their diets often lack important nutrients. One reason for this issue is the growing food desserts that can be found in cities; places where there are few to no grocery stores, or where the cost of groceries (particularly fresh fruits and vegetables) is too high for the residents to afford. To solve this issue, many people have turned to various urban agriculture solutions, such as community or rooftop gardens, or hydroponics. Unfortunately, many of these solutions require significant time and knowledge investment, which poorer families can not afford.

Our solution is to provide a semi-automated hydroponics greenhouse that is monitored and adjusted autonomously, with minimal input needed from the user. With a design focused on low-cost, low-space needs, a Pocket 'Ponics greenhouse is set up to mimic a bookshelf, but with plants growing on each tier. An easy-to-use mobile app, compatible with both Apple and Android phones, allows non-technical users to easily register their greenhouses, set up the plants with step-by-step instructions, and monitor the status of their greenhouses in real-time.

The Pocket 'Ponics project consists of three main parts: the hydroponic greenhouse, the server system, and the mobile application. The hydroponic greenhouse includes a sensor grid, pumps, and a lighting system that will monitor the status of the growing plants and maintain an optimal growing environment. The server system receives data from the greenhouse, informs the greenhouse about what actions to take, and surfaces information to the app so that users can take actions. The mobile application displays the current status of registered greenhouses and alerts users whenever they need to take action.

The hydroponic greenhouse leverages low-cost components to ensure that the growing conditions for each plant are optimal. On each of the four growing tiers, the greenhouse has full-spectrum light emitting diodes (LEDs) that can be automatically cycled on and off, as well as a separate growing tray so that individual balances of nutrients can be maintained. Within each growing tray, a pH sensor and an electrical conductivity sensor work together to monitor the nutrient concentration, sending data back to the server. If the server informs the greenhouse that certain values are out of the desired range, the individual pumps within the greenhouse can adjust the water and nutrient levels. The greenhouse also has an additional shelf for seedling cultivation, which is unmonitored. The greenhouse's water tank and nutrient tank are monitored by water-level sensors and the greenhouse has an auxiliary battery backup to help maintain systems if the wall power goes out.

The server coordinates all the information between the greenhouses and the mobile app. When it receives electrical conductivity and pH information from the greenhouse, it checks the values against ideal ranges for the particular plants and notifies the greenhouse to shift the water or nutrient levels when necessary. When the greenhouse notifies it about the levels in the water and nutrient tanks, it saves the information and sends notifications to the mobile app when levels are too low and the user needs to refill. The server also keeps track of plant life cycles, notifying the mobile app when it is time to set up seedlings, move seedlings to a new level, or harvest produce. Similarly, it maintains all the historical data for a greenhouse, ensuring that users can always view the trends that their greenhouse follows over time.

The mobile app is the user's main point of interaction. It retrieves data from the server and surfaces it in an easily accessible, user-friendly manner. It provides a step-by-step walkthrough for setting up a new greenhouse, including animations that guide a user through assembly and cultivation. It displays a detailed view of the greenhouse conditions, including the health status of each tier in the greenhouse and what the current battery, nutrient, and water levels are. The mobile app also provides notifications to the user whenever actions such as refilling the tank, moving seedlings, or harvesting the plants need to be taken. These appear as push notifications, so the user can be alerted even when the app is closed. When clicked, these notifications direct the user to screens that help guide them through whichever actions are necessary.

Because of this unique, three-part integration, Pocket 'Ponics removes much of the tedious day-to-day maintenance required for hydroponic growing. It makes growing accessible to anyone, regardless of experience, and provides a low-cost, low-space alternative to traditional agriculture. The space optimizing design of the greenhouse ensures that food can be grown easily in urban environments, and the self-contained nature means that users will not need any external resources to monitor or grow the plants. (Aside from nutrient fluid, water, and seeds) Additionally, because Pocket 'Ponics prioritizes affordability and accessibility, without sacrificing quality or user experience, all user interfaces are tested for useability against web content accessibility guidelines (WCAG), and all equipment is chosen for its cost-to-quality ratio. Furthermore, all the code and design implementation for Pocket 'Ponics will be open-sourced under the MIT license, available to anyone who wants to contribute to it or set up their own implementation. This ensures that as many people as possible will have the ability to set up a Pocket 'Ponics system, further reducing the entry barrier into hydroponics.

2.0 - Project Narrative

When laying out the design for a project, it is important to consider the context that the project is found within. First of all, when designing a project, it is important to start with a problem, instead of a solution, so looking at the broader impacts that the project can have is an important step in determining the project's design. After the broader impacts are established, it is also important to look at the legal and ethical considerations that could arise when trying to solve the problem, so that a solution can be devised that is as ethical as possible. Only once these contexts have been established can a reasonable solution be devised to solve the established problem.

In order to establish this context, we researched the incidence of food insecurity around the world, with a particular focus on the urban areas of the United States of America. We also researched the global food production statistics, to figure out the cause of the food insecurity levels, and looked at what previous solutions had been devised to try and solve this issue. Furthermore, once we had an idea of what solution we would use, we looked at what potential ethical issues we might run into, focusing on the data collection of internet-connected home appliances.

2.1 - Broader Impacts

The World Factbook estimates that 30-60% of the Earth's population is employed in agriculture around the world[1]. Records show that many of those people spend more than 12 hours per day working on food production, from farmers that rise and set with the sun, to meat factory worker that take increasingly long shifts. In various agricultural industries, food production has increased by 20% or more in the last decade.

And yet, despite this massive investment into agriculture, the Food and Agriculture Organization of the UN (FAO) estimates that 800 million people (1 out of 9) suffer from regular food shortages and that over 2 billion people (1 out of 4) have micronutrient deficiencies.[2] Indeed, eradicating hunger was the first of the eight Millennium Development Goals[3] that the United Nations hoped to meet by 2015, and is the second of the seventeen current Sustainable Development Goals[4], which the UN aims to achieve by 2030. According to the USDA's Economic Research Service, approximately 11.1% of households across the US were food insecure in 2018, meaning that they did not have enough nutritious food for all household members at some point during the year.[5] Members of such households have a higher rate of health issues due to poor nutrition and have reduced lifespan expectancies.

One of the reasons for this dichotomy is the increasing migration of people to cities. As people become more distant from the source of their food, increased energy is needed to transport the food, causing the price to skyrocket. Furthermore, cities often contain food desserts, which are areas where food is scarce due to the absence of grocery stores or food prices that are outside of the resident's budget. The residents of such areas are more likely to be food insecure, leading to a decreased quality of life.

In particularly bad food desserts, long-time residents may be forced to move out due to their inability to meet the soaring food costs. Residents that cannot afford to move, are stuck between a rock and a hard place - they cannot leave, but staying negatively impacts their health, both long and short-term. This negative impact on their health causes additional healthcare bills that they have to pay, further reducing their ability to buy nutritious food, in a vicious cycle that reinforces itself to the detriment of society.

To solve this issue, many people have proposed that people living in urban areas should grow their own food. Rooftop gardens, balcony vegetables, and community plots have all become more popular in the past decade, but all of them have several drawbacks. Rooftop gardens require proper infrastructure, as well as proper insulation and protection from pollution. Balcony gardens have similar problems and are also constrained by the amount of sunlight that they receive. Community plots may be restricted to certain locations, and cost a lot in real estate and irrigation. And all of these solutions require significant time investments and knowledge devoted to their cultivation and harvesting, and many people cannot afford any of those necessary factors.

Our project will take a different approach than these more traditional gardens. Instead of adding more requirements to already busy lives, Pocket 'Ponics will automate the use of small-scale hydroponics. By leveraging low-cost materials, Pocket 'Ponics greenhouses will be cheaper for the end-user, reducing the cost burden of urban agriculture. (When Pocket 'Ponics is available for end-users to buy, they will be sold on a buy-one-give-one model; the cost that an individual user pays will purchase a second greenhouse that will be donated to lower-income families)

Similarly, the greenhouses will be easier to set up than a traditional garden, with a plug-and-play approach that has become so popular with IoT devices. The user-friendly app and the Pocket 'Ponics server will allow the greenhouses to automate many of the time-consuming tasks of hydroponics and provides notification and direction every step of the way. Basically, our project would be like the IKEA furniture of urban agriculture; it would be easy to set up and simple to run, requiring no specialized knowledge and containing all of the instructions needed. Pocket 'Ponics would be easily accessible to everyone, regardless of background.

2.2 - Legal, Ethical and Privacy Issues

As we designed our project, we considered the legal, ethical and privacy issues that might arise as we progress with our project. From an ethical standpoint, our app doesn't seem to perform any actions that may be considered unethical. Our project is a hydroponic greenhouse designed to provide users with the opportunity to grow their own food and it doesn't take any actions that may be considered unethical. Additionally, there are no ethics issues with providing a product and service that allows people to grow their own food.

From a legal standpoint, our team considered the intellectual property that was involved in our designing and building our project. A preliminary search of the US Patent and Trademark Office (USPTO) records did not find an existing patent that our project would be infringing upon. We decided that we would file a patent application for any intellectual property that may arise from our project towards the end of Senior Design II. Additionally, there are no other legal concerns surrounding our project, as our project doesn't take any actions that may be breaking federal or state law. We will be including a basic terms and services contract for new users to outline the appropriate use for our platform. Furthermore, we conducted a search of the USPTO for trademarks similar to or matching "Pocket 'Ponics" and found no results. We also searched USPTO copyright records for that same keyword phrase and found no results.

Privacy concerns regarding our project are very minimal. Our app will not be collecting any user information such as location or device data. Our app will operate using limited device permissions, accessing the camera only to scan QR codes. Our backend database will only store username and email values for each user, minimizing the amount of personal data stored in our database. Passwords for each user are stored in a hashed format. By only storing password hashes instead of the passwords in encrypted or plaintext format, if our database was breached by an unauthorized party, only password hashes could be stolen and cannot be used to log into our application. Additionally, if users reuse their passwords for other sites, they will not have to change their passwords for other websites because of our security breach. The sensor grid will be collecting sensor readings, and these readings will be stored in our database. However, the data collected by the sensor grid is not personal information and is limited to basic sensor readings regarding water level, temperature, pH and electrical conductivity found in the greenhouse.

2.3 - Our Solution

This project works to alleviate the drawbacks of traditional urban farming. To reduce the space needed for cultivation, Pocket ‘Ponics will use a tiered hydroponics system. An individual micro-greenhouse will contain five tiers, each with a different plant growing in a hydroponic medium. This will allow a variety of plants with distinct nutrients needs to be grown in a space about the size of a traditional bookshelf. It also removes the need for soil and sunlight, allowing the cultivation of food to be moved to more convenient locations.

Similarly, Pocket 'Ponics will need much less time invested in it than the large amounts of time that are typically required for traditional agriculture. By leveraging an advanced sensor grid to monitor water level, pH, and electrical conductance, Pocket 'Ponics will remove the burden of monitoring from the users. Furthermore, electronically-controlled lighting, as well as water and nutrient pumps will ensure that the day-to-day operation of a Pocket 'Ponics systems is primarily autonomous. Users will only need to do the initial seeding, move plants to their correct trays, and the final harvest of any food.

The largest benefit that Pocket ‘Ponics will provide over traditional agriculture is the lack of specialized knowledge that will be needed to operate it. The autonomously maintained hydroponics will require no knowledge of hydroponic systems; users will simply plug in the micro-greenhouse, register it to the app, set up the plants, fill the water and nutrient tanks, and harvest the eventual output. Similarly, the users would not need any knowledge of growing seasons or nutrient levels required for each plant; a user-friendly mobile application will prompt the users whenever they need to take action, and a sophisticated backend will provide information on each plant.

Beyond simply compensating for the flaws of traditional architecture, Pocket ‘Ponics will make the growing process more transparent to users. At any time, users will be able to use the mobile application to view the current status of the plant in the greenhouse, and adjust the settings outside of the default ranges to better fit their environment.

Overall, the goal is to enable people with no knowledge of hydroponics or agriculture to easily grow food within the small space of an apartment or a home. By bringing agriculture back into cities and food deserts, Pocket ‘Ponics could help people eat healthier, by bringing balanced nutrients and vegetables fresh into their homes. It could also help to alleviate hunger and food insecurity, without the requirement of massive restructuring efforts.

2.4 - Individual Motivations

Elli Howard

I designed this project to synthesize my two majors, Biology and Computer Science. As a Biology major, I have both seen and helped with many of the arduous tasks in the agriculture industry. I felt that some of these tasks, particularly those dealing with hydroponics, could be automated and computerized so they require minimal human intervention. Similarly, I felt that many of the knowledge barriers that were present in agriculture could be removed by automation and easy-to-follow instructions. To that end, I planned out a rough design for an automated greenhouse in my sophomore year, and have been refining the idea ever since, developing the skills I need to bring the idea to fruition and recruiting others with complementary skill sets. I hope that the team that I have assembled can help me to bring my idea into reality, and I hope that the Pocket 'Ponics project will go on to help people who are facing food insecurity.

Rohan Patel

My motivations for selecting this project were because of my interest in working on backend development and combining the concepts I have learned in my computer vision coursework with a real-world application in this project. I found it very interesting to be able to use computer vision principles to detect when plants in the greenhouse are ready for harvest. Additionally, I found it exciting to build a backend using the skills I learned during my summer internships.

Matthew Bonsignore

This project is the perfect balance of all my interests and hobbies. I am a current plant hobbyist with several different species that I am maintaining. I have always been into hydroponics and I have always wanted to get a chance to apply it to my current practices. I am also interested in active data collection, and storage maintenance. I am also motivated for this project because of the possible outcomes of this project on a global scale for helping people in need of homegrown produce. Another aspect of this project that motivates me is the skills that I can gain from this project and be able to apply them to my own personal projects.

Graham Hill

When this project was pitched to me the team stressed an importance on a stable and easily maintainable sensor grid array, and during my experiences with electrical engineering so far there hasn't been experience constructing components and circuits that actually serve a purpose and link to a database structure, so I didn't want to miss the chance to really test and put my electrical engineering skills to the test. I also have had an interest in biology so a chance to work with plants to tweak how they grow and their environments was too fun of an opportunity to pass up.

Catherine Abbruzzese

I really like the motivation behind the project - to be able to make it easy for people to grow their own food. Where I grew up, food wasn't always readily available and I think helping develop an idea like this is a step forward so that, in the future, everyone has the option to set up and run their own garden, even with little knowledge of plants or coding. I think this creates the best way for people to grow their own food, it's basically a tutorial that walks you through setting up the plants, all the way to harvesting them and I think that's a cool thing to be involved in. I've also been really interested in learning about databases. I will be in charge of creating and managing the database, so it gives me the perfect opportunity to begin building the skills needed to manage data systems

Alexandra Cusell

I personally chose to take part in this project because I believe in the usage and importance of it. Pocket 'Ponics can have such a profound impact on so many different people around the world. We have the ability to provide a safe, fresh and easily accessible food source to those who do not currently have that luxury. It's easy for me to get behind and contribute to an idea with such relevance and meaning. I hope that after the completion of this course, our group can continue to work on making this idea a reality and hopefully bring this product to market.

3.0 - Project Objectives

In designing this project, the main goal is to create a semi-automatic greenhouse that monitors the growing medium of a variety of plants, and allows a non-technical user to grow plants with minimal interactions. It will leverage a shelving system of static water culture beds to grow the plants in a small spatial footprint, and will function as a three part system: greenhouse, server, and mobile app. These three parts will network together over the internet to ensure a simple and streamlined user experience.

3.1 - Sensor Grid Objectives

The sensor specifications are set such that the sensors will reliably read and send data from the microcontroller (MCU) to the System On Chip (SOC). The specifications for the MCU and SOCs use must be clear to ensure that future designs for the system are preserving the integrity of the components of which the system relies on to function properly. Voltage and power needs for all components must be taken into consideration to ensure that all components can be connected and lines can be laid with the right AWG so nothing will burn or overheat. How often to poll the sensor was the next step to configure in tandem with the computer science component of our team, we all mutually agreed that the sensor array shall be polled at minimum of 1 data pull per hour. Lastly all components excluding wiring shall be contained within the confines of the greenhouses shelves to prevent damage to essential components.

Water pumps will be used to manage water and nutrient flow throughout the system. Some specifications that these water pumps must meet involve flow rate, working voltage range, load current, average power, maximum lift, and dimensions. The water pumps that will be used in this project from both the water and nutrient systems will have an average flow rate of 1.5-1.8 liters per minute depending on voltage provided to the pump, under full load the pump has a pump head of 5 meters and a suction of 1.5 meters. Minimum and maximum voltage is 6 to 12 volts respectively, load current during operation will not exceed 500mA, with a maximum power draw of 6 Watts under full load. Dimensions 95mm x 47mm x 36mm with a mount system allowing it to be easily placed and mounted securely.

The water level sensor (water float) is the safety system in place for the water management system of the greenhouse. Two of these sensors will be used with one in the inverted position to halt the pumping system from issuing more water or nutrients when the maximum fill sensor is sending a high output. Since these sensors are essentially a wire they have a load current limit of 1 amp and have a breakdown voltage of 220 volts. The outputs of these sensors shall connect to a

ground bus connected with resistors. Signal is then connected from there to the digital pins of the MCU which provides a high input impedance to limit the load current to prevent burning of the sensors, or getting false readings.

The pH sensor is one of two components used to determine the nutrient level of the nutrient infused water that is used to nourish the plants. The specifications are specific for this component such that the readings can be read reliably and be used for nutrient calculations for each tier of the greenhouse without risk of oversaturating the tier with nutrient solution.. The specifications that the pH sensor must meet involve the sensor's working current, detectable concentration range of pH, temperature detection range, dimensions, response time, setting time, and working power. The working current that the pH detector must have is 5 milliamps to 10 milliamps at a maximum input voltage of 5 volts, with a maximum power load of 500 millieWats when over volted. The response time is less than 5 seconds to receive a reading, and said reading should become stable in less than 60 seconds (Settling Time). The detectable concentration range for the pH sensor is between 0 pH to 14 pH. It requires a working temperature range of 0 degrees celsius to 80 degrees celsius to prevent misreading and miscalculations. The dimensions for this selected sensor is a length of 42 millimeters, a height of 20 millimeters and a width of 32 millimeters.

The LED lighting strips are used to supply the growing plants with sufficient light to thrive in. The specifications that are set on this component are used to ensure that the LEDs give off the correct type of light and that they can function reliably in the system. The specifications that the LED lights must meet involve input voltage, density of the LEDs, the red to blue ratio, wavelengths of the red and blue colors, and maximum width. The LEDs need to have an input voltage of 12 volts at 2 amps, with a power load of 24 watts per a strip. The density of the LEDs needs to be around 60 LEDs per meter to provide enough light for proper growth. The LEDs need to have a red to blue LED ratio of four red to one blue, with the respective wavelengths being 660 nanometers for the red LEDs, while the blue LEDs provide 455 nanometers.

The electrical conductivity sensor is the second component used to determine the nutrient level of the nutrient infused water that is used to nourish the plants. The specifications are specific for this component such that the readings can be read reliably and be used for nutrient calculations for each tier of the greenhouse without risk of oversaturating the tier with nutrient solution. The specifications that the electrical conductivity sensor must meet involve the sensor's input voltage working range, output voltage range, working current range, and the range of total dissolved solids that it can measure. The electrical conductivity sensor must have an input voltage range of 3.3 volts to 5.5 volts, with an output voltage range from 0 volts to 2.3 volts. The working current range for this sensor should be from 3 milliamps to 6 milliamps due to high input impedance of the circuit. The electrical conductivity sensor can read between total dissolved solids measuring r

from 0 parts per million to 1000 parts per million. This range also has an accuracy tolerance of about plus or minus ten parts per million.

All sensors and electrical connections need to be waterproofed to ensure the integrity of the system and the signals that need to be passed between the different components, as shown in Table 3.1.1.

Table 3.1.1: Wiring Specifications

The system wiring shall be waterproofed to an equivalent standard.	IP55

Microcontroller Unit (MCU) is the central component that is in charge of collecting sensor readings from all sensors in the greenhouse and governing the pumps for the nutrients and water along with the LED systems. This system needs to be able to control multiple relays that are integral for the operation of the system as a whole. MCU control specifications are shown in Table 3.1.2

Table 3.1.2: MCU Control Specifications

The MCU shall have a stable input voltage at all times	+12V
The MCU shall control all pump based systems for the use of water or nutrients.	
The MCU shall provide a digital high to relays to turn on or off either the pumps or the LEDs in the greenhouse.	+5V
The MCU shall be governed by the SOC system to administer changes to the pumps and LEDs	
The MCU shall receive and use analog readings from the pH and EC sensors hourly	

The power system is composed of the power supply, the 12V to 5V step down bucks, the power relays, and the power connection between the sensors and/or SOC/MCU. The power supply of the power system will be a stand alone system instead of a student designed . This system is the most curated due to the fact that this system is connected to all the components in the greenhouse. Requirements for the power system (Table 3.1.3) are stricter, needing set values for each section of inputs for the sensors and the array.

Table 3.1.3: Power Systems Specifications

The power supply shall receive an AC signal as an input.	120V AC at 60Hz
The power supply shall step down the voltage of the AC signal	12V AC at 60Hz
The power supply shall rectify the AC signal to DC at specified values	12V
The power supply shall provide a DC signal to the LEDs	12V
The power supply shall provide a DC signal to the pump system	12V at 500mA
The step down bucks shall provide a DC signal to the pH sensor	3V-5V at 5mA-10mA
The step down bucks shall provide a DC signal to the EC sensor	3.3V-5.5V at 3mA-6mA
The power supply shall provide a DC signal to the SOC through a 12 to 5 volt convertor	5V at 3A
The power supply shall provide a DC signal to the MCU	12V at 2A
The power supply shall maintain designated voltages with a specified variance	±5% specified
The step down bucks shall power the power relays	5V
The power relays shall trigger when a digital high is applied to their control pins	5V digital high

The SOC is the component that has connections to the API and all of the sensors within the array. The SOC shall receive all connections of the sensors from the MCU and will provide the readings to the API whenever the SOC pushes the information to the backend. The SOC shall also be able to receive all readings with no information being crossed or lost in the communication. The SOC shall

maintain a connection to the backend system, the SOC is rated to conform to IEEE rating of 802.11. The SOC will keep this connection and up time for 99% of the time. The SOC shall run a full Linux operating system to allow for easier data manipulation. The SOC power specifications are demonstrated in Table 3.1.4 and the SOC connection specifications are listed in Table 3.1.5.

Table 3.1.4: SOC Power Specifications

The SOC shall receive a DC signal from the power system through USB.	5V at 3A

Table 3.1.5: SOC Connection Specifications

Shall send water level, pH and conductivity readings for a single sensor on a specified tier
Shall send water level, pH and conductivity readings for all sensors on specified tier
Shall send water level, pH and conductivity readings for all sensors in the greenhouse
Shall be able to modify a single or all tiers water, light, and nutrient levels based on backend feedback on current levels.

Sensor Stretch Goals

Some of the goals would be to implement a camera, set up a back-up power system, and to add more sensors. Adding a camera to the system would allow users to have a broader control and better understanding of when plants are ready. By having a camera constantly monitoring the system, one can view the progress of the plants on all tiers without physically being at the system. A back-up power system would consist of a battery array with a battery controller for safety, the user would not have to worry about a power failure shutting down the system and having to reboot all the components. If implemented this system would allow the MCU and the SOC to remain up and running and would send a signal to the API to notify the user that there was a power failure. We could also add humidity and temperature sensors for the greenhouse environment and water temperature sensors. The first two would allow for a better understanding of the environment within the greenhouse so better tweaking maybe be done for optimal growing conditions. Water temperature sensors would be used to check if the temperature of the water is in an appropriate range for the plants to thrive and the other sensors to operate correctly.

3.2 - Backend Objectives

The backend shall be able to receive specific information on each greenhouse from the sensor grid. Additionally, the backend shall be able to receive water level, pH and conductivity readings for a single sensor on a specified tier. Furthermore, the backend shall be able to receive water level, pH and conductivity readings for all sensors on specified tier. Also, the backend shall be able to receive water level, pH and conductivity readings for all sensors in the greenhouse. Also, the backend shall be able to receive current power supply for the greenhouse. The backend shall be able to receive current backup battery level for the greenhouse.

The backend shall be able to receive specific information from the app. Furthermore, the backend shall be able to receive lighting adjustments for a specific level in the greenhouse. The backend shall be able to create a new greenhouse using the information provided by the user. The backend shall be able to modify existing greenhouse using the information provided by the user. Also, the backend shall be able to modify existing levels in the greenhouse using the information provided by the user. Also, the backend shall be able to delete existing greenhouse.

The backend should also retrieve/store specific information on each greenhouse in the database. The backend shall retrieve/store water level, pH and conductivity readings for a single sensor on the specified tier in the database. Also, the backend shall retrieve/store water level, pH and conductivity readings for all sensors on the specified tier in the database. Additionally, the backend shall retrieve/store water level, pH and conductivity readings for all sensors in the greenhouse in the database. The backend shall retrieve/store cycle time for a specified tier in the greenhouse in the database. Additionally, the backend shall retrieve/store the greenhouse's user-defined name in the database. Furthermore, the backend shall retrieve/store the greenhouse's backup battery's current power level in the database. The backend shall retrieve/store the greenhouse's current power source in the database. Also, the backend shall retrieve/store the greenhouse's owner in the database. The backend shall also retrieve/store the greenhouse water tank level in the database. The backend shall also retrieve/store greenhouse nutrient level in the database.

The backend shall send specific information on each greenhouse to the app. The backend shall send water level, pH and conductivity readings for a single sensor on a specified tier. The backend shall send water level, pH and conductivity readings for all sensors on specified tier. Also, the backend shall send water level, pH and conductivity readings for all sensors in the greenhouse. Additionally, the backend shall send light start time and cycle time for a specified tier level. Furthermore, The backend shall send current power supply for the

greenhouse. The backend shall also send the current backup battery level for the greenhouse.

The backend shall send a list of the user's greenhouses to the app. The backend shall notify the frontend when refilling water or nutrients is needed. Additionally, the backend shall notify the frontend when planting seedlings or moving to different tier is needed. Furthermore, the backend shall notify the frontend when harvesting plants is needed

The backend shall send commands to the greenhouse. The backend shall adjust light starting times for specific levels on user command. Additionally, the backend shall handle authentication for mobile app interactions. Also, the backend shall receive new user information and store it in the database. The backend shall also receive existing username/password and verify credentials with the database. The backend shall also issue a temporary token to the mobile app for future API requests. Also, the backend shall modify the user's password in the database. The backend shall reset the user's password and allow the user to change the password by providing the current and new passwords.

The backend shall handle authentication for sensor grid interactions. The backend shall link a new sensor grid to a specific user's greenhouse. Also, the backend shall store serial number and hash of the secret key in the database. The backend shall receive existing sensor grid credentials and verify against database values for all API requests. The backend shall also remove the sensor grid from the database if the greenhouse is deleted.

The backend shall retrieve the current plant ideal values from the database for the administrator's portal. The backend shall also allow the admin user to delete plants from the database. The backend shall also allow the admin user to create plants in the database. The backend shall also allow the admin user to modify existing plants in the database.

Backend Stretch Goals

The backend will use artificial intelligence to classify images of plants growing in the greenhouse, taken by the user. The backend will also monitor power source and backup battery. The backend will send a notification to the frontend when the power source is interrupted. Additionally, the backend will send a notification to the frontend when the backup battery level is low. The database will also store the current power source for each greenhouse. The database will also store the current backup battery level for each greenhouse.

3.3 - Frontend Objectives

The app shall allow users to set up a new account when presented with the login screen. The app shall also allow users to log in to an existing account, and the app shall save a user's login to allow for easy closing and reopening of the app. Furthermore, the app shall allow users to log in and out with ease, and the app shall allow users to reset their password whenever they want by clicking a reset password button in the user settings page.

When setting up a new greenhouse, the app shall provide step-by-step instructions for the setup and installation. Greenhouse setup shall use local networks for easy registration and connection. Greenhouse setup shall also allow users to select which plants will be grown on each tier of the greenhouse. Furthermore, Greenhouse setup shall provide instructions for filling the water and nutrient tanks.

Once a user has logged in, the app shall list all greenhouses associated with the logged-in user. The app shall provide specific information about each greenhouse, including information about the water and nutrient tank levels for each greenhouse and information about the battery reserves for each greenhouse. The app shall also provide historical data about tank levels and battery reserves for each greenhouse. When scrolling all the way to the bottom, the app shall allow the user to modify and remove greenhouses from their account.

When a user clicks on a specific tier in a greenhouse, the app shall display specific information for the selected tier, including the pH, electrical conductance, and the water level for each tier. Furthermore, the app should indicate if these values are within the desired range, and if they are being adjusted by the greenhouse. The app shall also display estimated growing times for each tier, and the app shall display estimated harvest dates for each tier. When plants are ready to harvest, the app shall display a Harvest button that shows the user on how to harvest the plant with step-by-step instructions.

The app shall send push notifications to alert the user to time-sensitive information about the database. For example, the app shall notify the user when water tank levels are low, when nutrient tank levels are low, when it is time to plant seedlings, when it is time to install seedlings into the proper tier, and when it is time to harvest the plants. Clicking on any of these notifications shall take the user to the appropriate screen within the app, so that additional navigation is not necessary to complete the actions.

In order to maximise useability, and meet WCAG and ADA standards, the app shall be accessible to people with disabilities. The app shall provide suitable color contrast for all forms of color-blindness, the app shall be navigable by users that rely on text-to-speech software, the app shall have navigation that is intuitive

for people with learning disabilities, and the app should not provide timelines that are inaccessible to people with anxiety. Wherever standards match up with web guidelines, the app shall follow web standards such as WCAG. If there are not available accessibility standards for a particular component, the app shall meet best practices recommendations to ensure that as many people as possible can use the app without needing additional accommodations.

Frontend Stretch Goals

The first stretch goal is that the app shall integrate with social media services such as Facebook and Twitter. The app shall allow users to easily link social media accounts, and it shall store linked social media accounts for future posts and updates. The app shall allow users to post greenhouse status to social media, and the app shall allow users to post-harvest dates to social media.

The second stretch goal is that the app shall allow users to handoff greenhouses to another user. The app shall allow for permanent greenhouse transfers, with a confirmation dialog, and the app shall also allow for temporary greenhouse transfers, with the ability to end the temporary transfer at any time.

For the third stretch goal, the app shall allow users to share greenhouses between two or more users. The app will designate a primary user for the greenhouse, with secondary users having limited permissions, and the app will allow the primary user to control the user permissions for the secondary users. Furthermore, the app will allow primary users to revoke secondary users at any time.

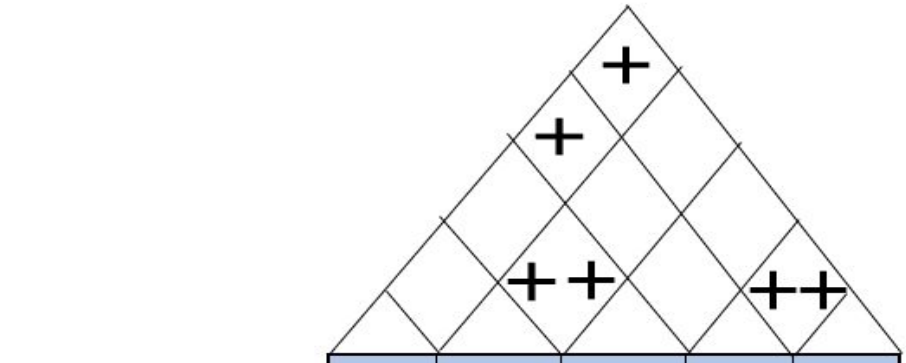
The fourth stretch goal is that the app shall provide easy links to re-order materials that may be used up. Reordering shall integrate with Amazon's Add To Cart feature, and shall allow the user to reorder seeds, nutrient fluid, and rockwool from Amazon. The reorder feature shall be accessible from the user settings menu.

The fifth and final stretch goal is that there will be an administrative portal in the Pocket Ponics website so that admins can edit, add, and delete plants that users can then add to new greenhouses. The website will only be accessible to users that have been flagged as admins, and will require a login to access. It will also be hidden form average users, so that only people who know the specific url for the admin login will be able to navigate to the admin login for the admin portal.

3.4 - House of Quality

The House of Quality is a product planning matrix that is used to relate the needs or requirements of the customer to the methods or ways a business will meet those needs or requirements. The house of quality can be broken up into two different sections. These two sections are the correlation matrix, and the body of the "house". The correlation matrix is the "roof" of the house of quality it helps to define if the engineering specifications defined are correlating with each other either optimizing or sub-optimize each of the specifications that are involved. A single + shows a correlation while the ++ shows a stronger correlation implying that the specifications will have a stronger effect on each other. The body of the house of quality is the main structure of the house of quality and is used to compare the product specifications and market specifications. For each of the specifications, there will be a + or - associated with the specification. The location that they meet in the grid shows how closely they are correlated. Double arrows represent high correlation and the direction determines if they are positively or negatively correlated. This can be shown in Figure 3.4.1 below.

Figure 3.4.1: Marketing and Engineering Requirements



		Efficency	Install Time	Dimensions	Quality	Cost
		+	-	-	+	-
User Friendly	+		↑	↑	↑	↓
Low Cost	-	↑↑	↓	↓	↓	↑↑
Ease of Installation	+		↑↑	↑	↑	↓
Reliability	+	↑			↑	↓
Low Maintenance	-	↓			↓	↑

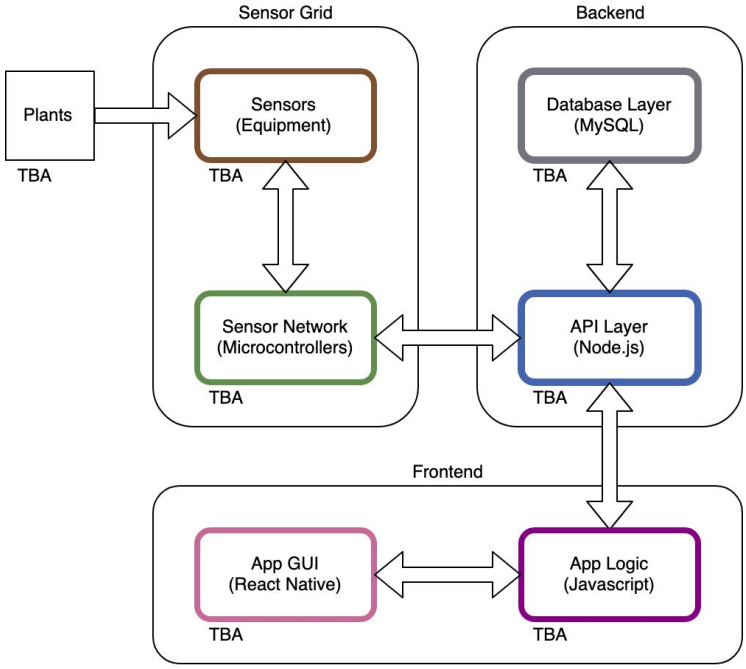
4.0 - Block Diagrams

The following block diagrams provide an overview of how data flows through the Pocket 'Ponics system. The overview diagram shows the system as a whole, and then the other diagrams provide a more detailed look at each of the constituent parts of the system. Although these diagrams provide a good look at how the system will function, they are only intended as a summary and a guiding framework; more advanced implementation details are provided later in the document.

4.1 - Overview Diagram

The Pocket 'Ponics system is primarily divided into three major sections, each of which has two component subsections (Figure 4.1.1). The first section, the sensor grid, is the electronics attached to the greenhouse, which provide the monitoring and modification of the hydroponics system. It is subdivided into the actual sensors and equipment needed, and the sensor network, which is the microcontrollers that coordinate the sensors. This section sends data up to the backend section, where the API subsection provides back and forth communication with the database. The API also serves as the connection to the frontend section, via the app logic, which then serves information to the GUI so that users can see it.

Figure 4.1.1: Project Overview



4.2 - Sensor Grid Block Diagrams

The sensor grid can be broken up into two different layers. These two layers are the Network layer and the Sensor layer. The Network layer comprises all the components that are related to the power system, System On Chip (SOC), microcontroller unit (MCU) / PCB, AC to DC power system (power supply, relays, bucks), and it also connects the backend API with the Sensor layer. This can be seen in Figure 4.2.1. The Sensor layer contains all of the components that are used to maintain and measure all parts of the system. These include The water management system, the electrical conductivity sensors, pH sensors, and the LED lights. The water management system contains the water level sensors, water tank reservoir and the water pumps. This can be seen in Figure 4.2.2. These Network Layer Overview and the Sensor Layer Overview are meant to provide a general sense of how each layer interacts with one another, and what is expected to be done in each of the respective layers.

Figure 4.2.1: Network Layer Overview

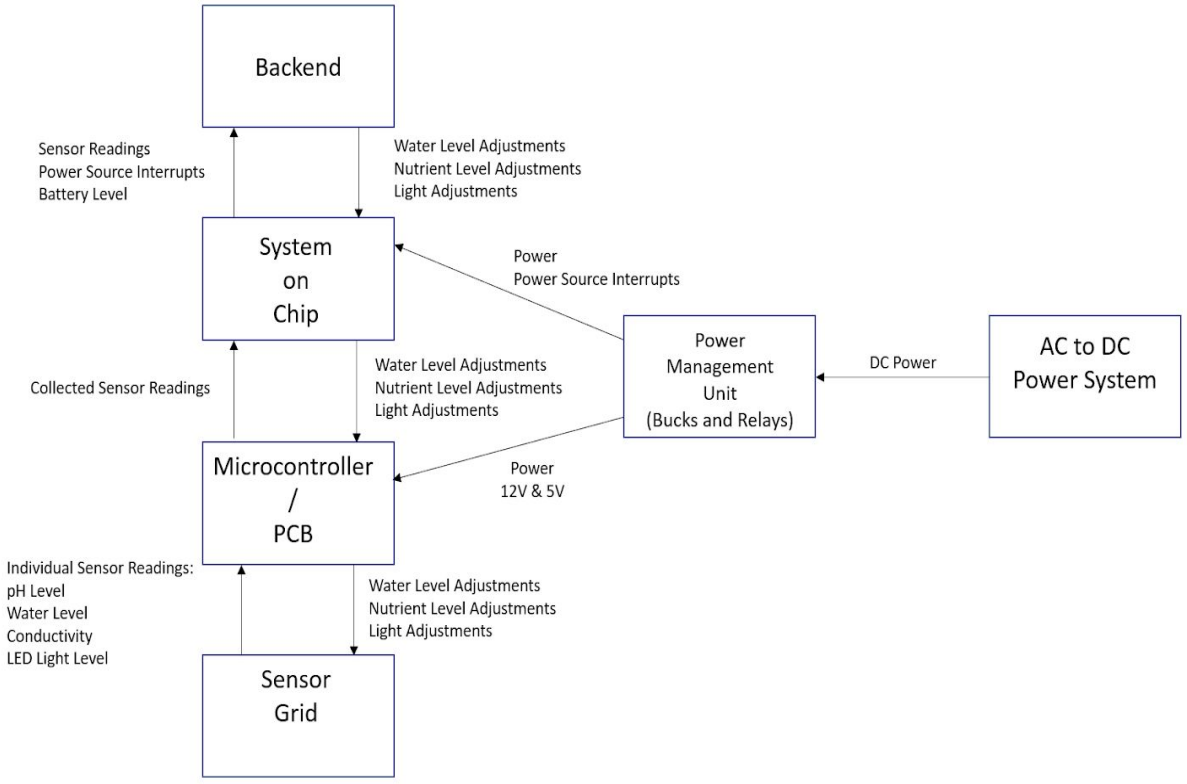
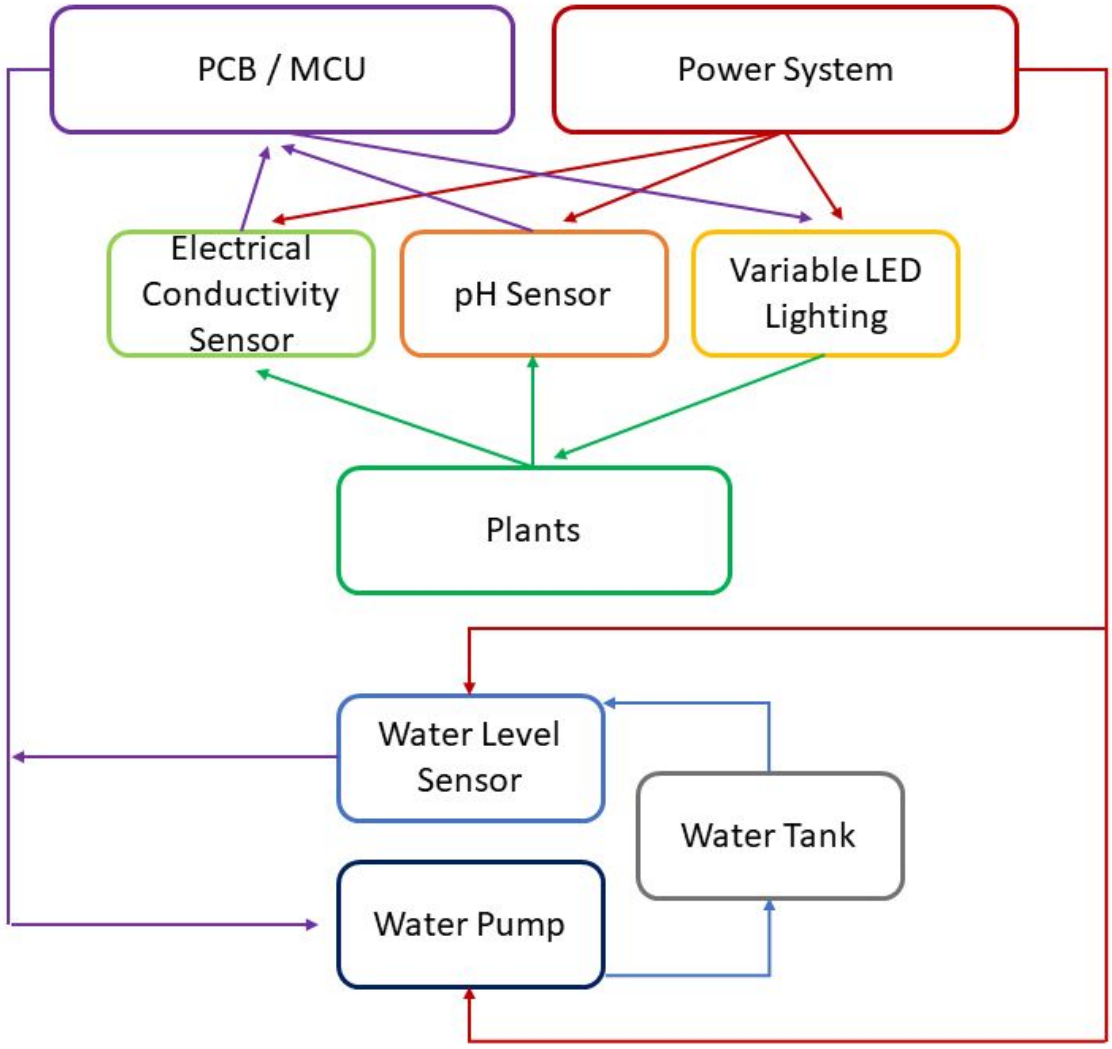


Figure 4.2.2: Sensor Layer Overview



4.3 - Backend Block Diagrams

The backend will be divided into two parts: the Node.js backend server (written using REST API protocols), and the MySQL database, as shown in Figure 4.3.1. The Node.js backend server will handle all incoming and outgoing communication between the mobile app and the sensor grid. The MySQL database will contain the data for each greenhouse, user, and sensor grid. The backend will provide and/or store data for each greenhouse and user in the MySQL database. Indirectly, the sensor grid will store sensor readings in the MySQL database through the use of an API endpoint. The mobile app will retrieve data from the database through a separate group of API endpoints (Figure 4.3.2).

Figure 4.3.1: Backend Interactions Overview

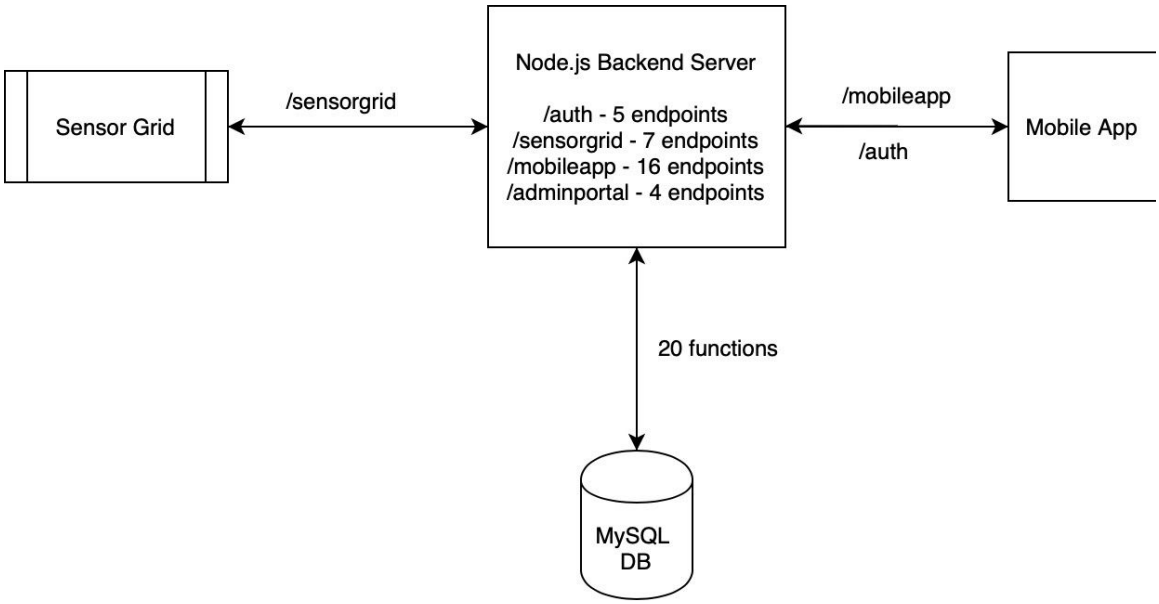
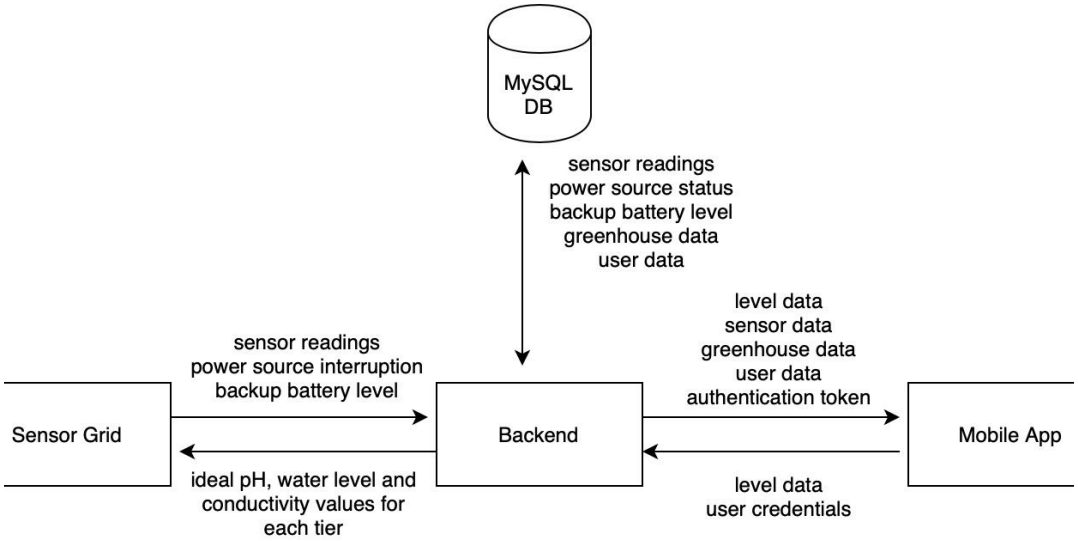


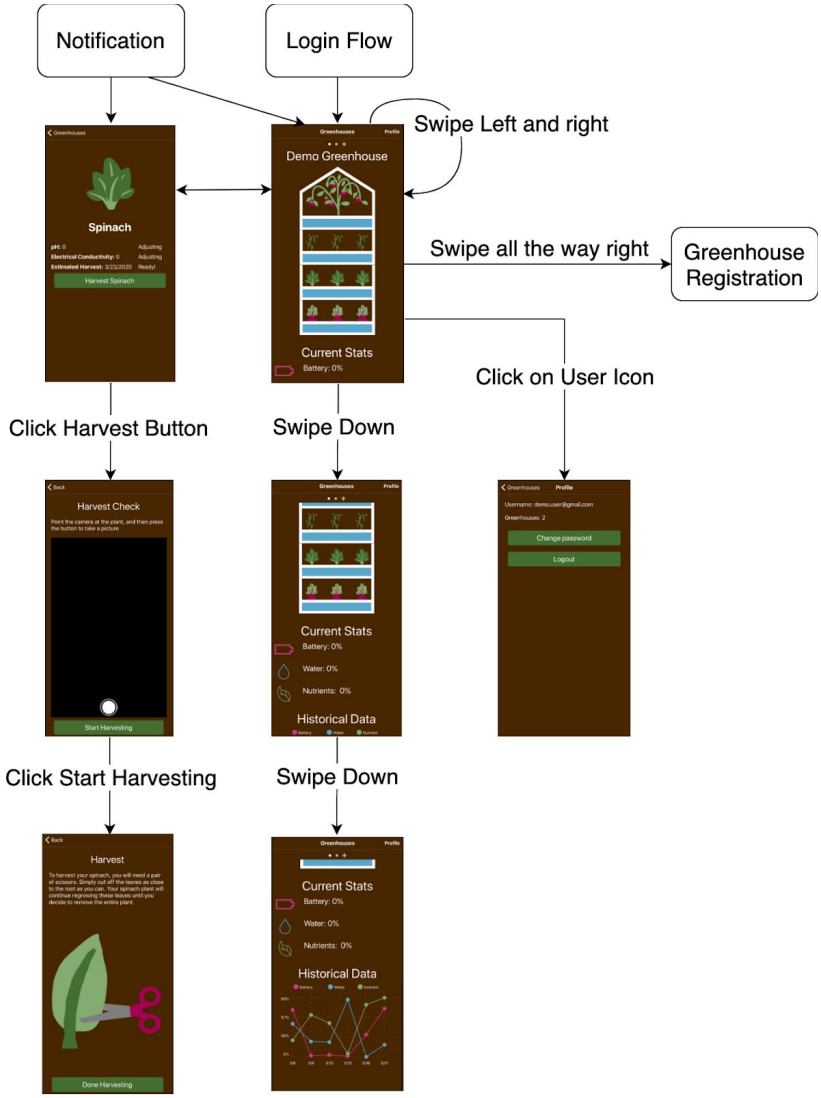
Figure 4.3.2: Backend API Overview



4.4 - Frontend Block Diagrams

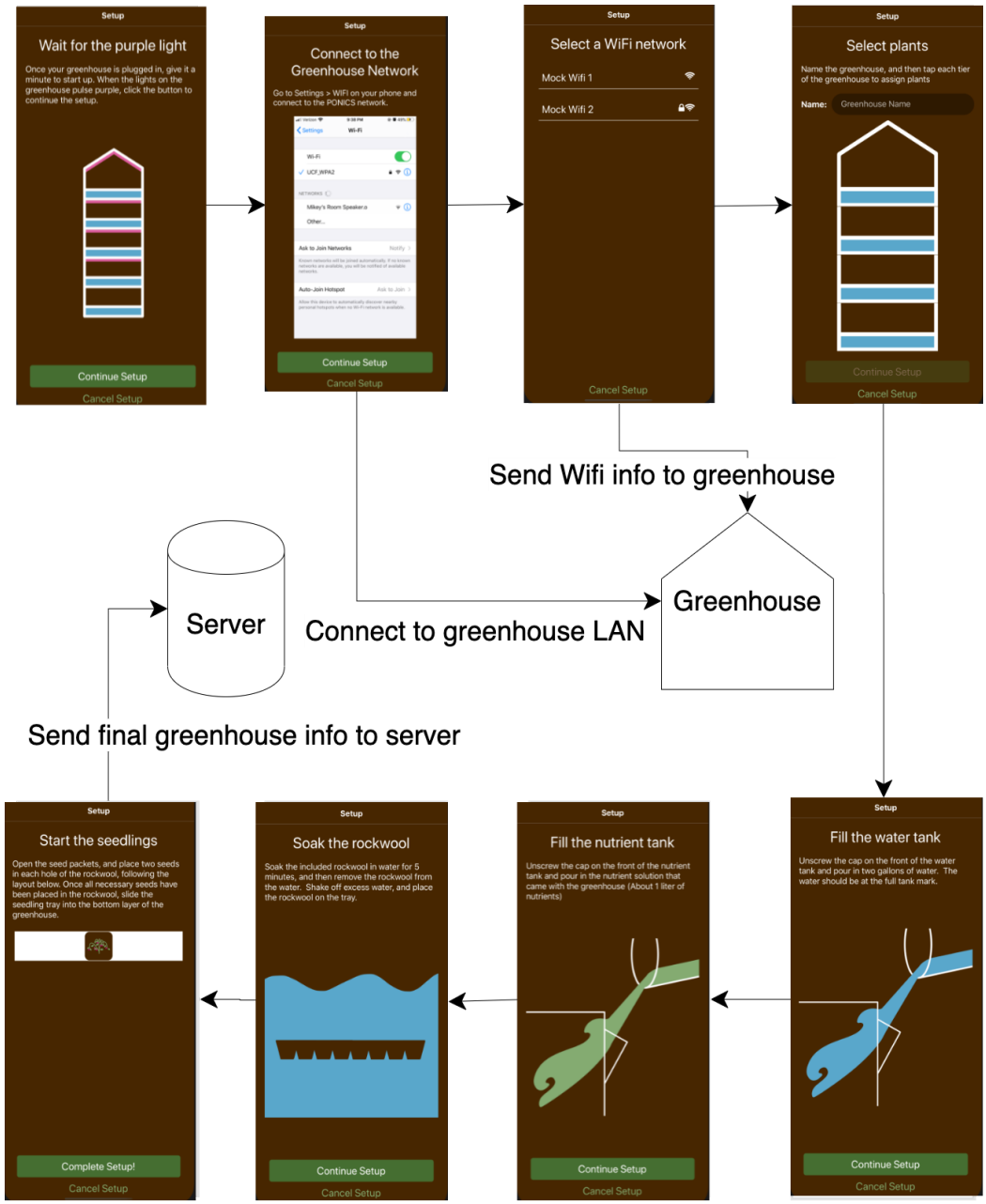
The frontend mobile app can be broken down into two major flows: the main navigation flow, and the greenhouse registration flow. The main navigation flow (Figure 4.4.1) is the primary way that users will navigate around the app. If the user enters via the login flow or via notification, they will be able to swipe left and right to switch between greenhouses. They will also be able to scroll down to view greenhouse data and historical data, or click on a particular tier of the greenhouse to view detailed information about that tier. Clicking on the harvest button will walk the user through harvesting the plant, while clicking on the user button will take the user to user settings. Finally, swiping all the way to the right will allow users to register a new greenhouse, starting the greenhouse registration flow.

Figure 4.4.1: Main Navigation Flow



The greenhouse registration flow shown in Figure 4.4.2 is in charge of the initial coordination between the greenhouse and the server. Once setup starts, scanning the greenhouse's QR code connects the app to the greenhouse's Local Area Network (LAN). the user then selects a Wifi, and the information is sent to the greenhouse. Then the user is instructed to fill the reservoir tanks and set up tiers, including instructions on how to set up seedlings. Finally, the app sends all of the greenhouse data to the server, initializing the new greenhouse.

Figure 4.4.2: Greenhouse Registration Block Diagram



5.0 - Project Brainstorming

When starting this project, each person in the group was asked to come up with ten to fifteen ideas of how to solve the problem laid out in the initial problem statement. Once these ideas were generated, we sat down as a team and selected the best solutions, refining it until we had a handful of solutions from each section that could be compared and analyzed. These solutions were then further investigated by the team member in charge of each subsection, to ensure that we arrived at the best possible design for this project.

5.1 - Sensor Brainstorming

The sensor brainstorming focused on the component parts that would be needed to monitor and maintain the hydroponics systems in the greenhouse. Because the hydroponic system is going to be a static water-feature, most of our sensor brainstorming revolves around the parts needed to monitor and maintain pH, electrical conductivity, and water level, as well as the necessary light levels for each plant. We also needed a way to coordinate the pumps and the sensors, so we looked at various options for a microcontroller unit (MCU) to control the various components needed on each tier of the greenhouse.

We also spent some time brainstorming the design of the greenhouse itself, and established that it would be a five shelf design, with a shallow water and nutrient tray on each of the top four shelves, the water reservoir, nutrient reservoir, and electronics storage on the bottom shelf. The greenhouse itself will be based on the simple micro greenhouse shelving units available at stores like Lowes and Home Depot, for easy indoor gardening. These designs often include wheels on the bottom of the greenhouse, to make transporting and installing the greenhouse easier.

Possibilities Considered

When considering all the different possibilities in regards to the composition of the sensor layer of our project. We decided to brainstorm about every aspect of the layer to see what all the options were out there for us to consider. The main component that we focused on was the MCU that would manage the inputs and outputs of all the sensors. We also need to figure out whether or not we would need an MCU and SOC for this project. Furthermore, we also had to decide what types of sensors would be most beneficial for this project without overextending ourselves.

In regards to the MCU, we had thought of three different candidates. The first was an ARM (Advanced RISC Machine) MCU, a Texas Instruments MCU, or going for a more curated system that an arduino can provide. In conjunction with deciding which controller to use for this project, it was also important to determine what kind of communication methods that would need to be used to transfer data between the MCU and the SBC if it is decided that two systems are needed.

For our purposes in regards to the functionality of our hydroponics project, we found that having sensors for measuring water level, water conductivity, and pH would be necessary to make sure that all aspects of the hydroponic system are running effectively and efficiently. Along with the decisions of sensors we had to decide how we would implement their use for flow control of the entire water management system.

The next aspect of the sensor layer that needed to be determined was the composition of the main power distribution. The options that we considered were a rechargeable lithium-ion based battery pack paired with a battery controller, a student designed AC to DC power system, or a prebuilt power supply that would convert 120V AC to 12V DC. Another possibility we considered was having a few AA batteries as a backup power system that could run the basic aspects of the main MCU and sensors in case of a power failure. We also thought that having a way to check on current battery levels would be a good idea, such that users could easily maintain the system.

Lastly, we decided that having the ability to change the intensity and also the composition of the light coming from the LED lighting strips in each of the levels of the hydroponic system could be useful if light intensity was needed on a plant to plant basis. But a simple static light level LED strip is another possible consideration.

Design Comparison

All of the MCUs that we considered during the brainstorming session have aspects that make one better than another. ARM devices are considered to be more professional and are designed to maximize the balance between high precision and low power consumption. Of all of the MCUs that we considered, TI is the most common and also respected among professionals. On top of that, it is much more affordable and we as a team have more experience using TI MCUs than the others. We also considered using a SOC (System on Chip), instead of using, or in conjunction with an MCU. Specifically, we were looking at either an Arduino or a Raspberry Pi. The strengths of these are that they are very easy to come by, very user-friendly, and relatively affordable. The weakness of using an Arduino or RaspberryPi as opposed to a TI or ARM MCU is that you trade a more

respected industry standard for a more client based product that is less widely used in large engineering firms.

Determining the specific sensors and devices is a balance between the amount of range per sensor, ease of use, and cost. For the water pump, having an average flow rate of around 100 Liters per hour along with a decent efficiency was important . We also needed to have an electrical conductivity sensor that would be able to measure how conductive the water in the tiers of the system is, and shows the total amount of dissolved solids in the tiers water. Finally, a pH sensor is vital to our project so that we can reliably measure the pH of each of the tiers. The pH determines the amount of hydrogen ions dissolved into the water, which is a component of the nutrition of the water for the plants. We also wanted to minimize the cost associated with all of these sensors and pumps, so we shopped around at a variety of different locations, including Amazon, Adafruit, DFRobot, and various component shops.

Selected Design

For the sensor grid, we decided to use both the Arduino MCU (ATMega2560) and a SOC device (Raspberry pi 3B). The combination of the different strengths of both systems in the Arduino and SOC devices will allow our system to be more flexible and adaptable. The Arduino device has the power and computational efficiency advantage over a SOC device. However, an SOC device is much better suited to communicate with an API effectively such that we can send and receive data through the SOC between the MCU and an API, using the SOC as a middle man for communication. For the MCU we have decided to use a atMega2560 chip with a modified arduino based PCB. As for the SOC, we are going to use a Raspberry Pi to perform the communication between the atmega2560 and the API.

For the sensors and components of the project, we decided to use the electrical conductivity sensor, pH sensor, water pump. As for the water pump, we selected the Gikfun DC 12V Mountable Water Pump. We decided on this pump because of its very high efficiency to cost ratio, and because of its ease of use. It is important that the pump has a large flow rate because this will increase the efficiency of pumping the nutrient-rich water to the plants regardless of how high the tier is that needs the nutrients. For both the pH and EC/TDS sensors we decided to go with DFrobot brand sensors that are specifically designed to run with the arduino brand systems due to the including analog based pins that provide voltage readings. These sensors have an output voltage that can be converted into a pH reading with code that is provided with the sensors.

5.2 - Network Brainstorming

The network brainstorming focused on the parts needed to coordinate the data received from the sensors with the API layer. As such, it was important to select components that would be able to manage an internet connection, as well as components that could handle a large number of inputs. We also needed something that was fairly easy to program, since the network was already going to require a lot of time to design and build, and we wanted the learning curve to be as simple as possible.

We also needed to consider how the network devices would be attached to the greenhouse, and how they would be protected from liquids. Since the greenhouse is designed to handle hydroponics, it is absolutely vital that the electronic components be protected from stray water or nutrient solutions, since even a little bit of water could destroy the components due to the high levels of electrical conductivity when the system is live. We also needed to consider how to mount the network system to the greenhouse, which would be further complicated by needing to shield the plants from excess heat that the network system might put off.

Possibilities Considered

When designing and considering the network behind the sensor grid it is vital that a stable and safe system was used for the connections to the sensors and to distribute power to the array of sensors and computers/controllers on the grid.

When choosing how to protect the wiring of the system it came down to three options. We could seal it with liquid silicon, cover it with electrical tape, or protect it using heat shrink each having varying degrees of protection and downsides.

A PCB (printed circuit board) throughput for connection from the MCU to the sensors will allow for the sensor array to connect and meet at one board before being processed by the MCU and sent to the SOC. The team wanted the connections and interactions between the different logical hardware pieces to be as laid out and clean as possible, using the PCB in between the SOC and sensor grid will allow for easy combination of sensor data to be sent to the back end system.

The use of a power supply based system to provide power to all the components of the system is essential for the continuous function of the greenhouse. The team came down to two options of providing power to the system using a wall socket and converting the 120V AC signal to different DC signals to the components. The other option is to use a battery-based power source composed

of rechargeable lithium-ion batteries, with their signal being transformed up or down depending on the requirements for the component being powered.

The power supply is going to be the highest heat producing component in the system since it will be stepping down the 120V ac to 12V dc. Because of this if a student designed power supply is implemented then a heat sink based system is going to have to be implemented, if not then the components could easily overheat and the system would go offline. The heat sink would be a simple metal plate that would mount to the linear voltage controllers and transformers to alleviate energy dispersion from the system. If we opt for a prebuilt power supply most times they will be enclosed within a metal enclosure that will function as the heat sink.

For mounting the system components to the greenhouse there are a number of different methods that would be possible, some of which would be using a large metal plate or using fiberglass as mounting plates for the electronics and for the systems running the greenhouse. If using the metal plate it could double as a heat sink for the other components in the system but extra protection of the components will be needed since some of the pcbs of the smaller components have throughput holes that could easily cause shorts in the system if they were to touch the metal mounting plate. On the other hand using a fiberglass plate would alleviate the chance of components shorting and destroying the PCBs on the system.

We also needed to consider where we were mounting the system. Our first choice for mounting the system would be to mount the components onto the bottom shelf of the greenhouse. Making is easy to reach and work on during construction and use. The second option would be to place all the components along the top tier of the system which would keep them safe from water drips but would make it more difficult to reach them post construction and during operation.

Design Comparison

Of these three coating/protection options we looked at which were going to have the best protection while keeping signal integrity, and also keeping the cost of protective supplies lower. Liquid Silicon coating is versatile and easy to apply to lengths of wiring connection and one dried has little chance of coming loose and allowing water to penetrate to the wire. It's easier to apply multiple layers of the coating to the connection sites than it would be with different protection types. Electrical tape is cheap and easy to use to protect the connection points. You have to wrap the connections multiple times tightly making sure to cover any exposed connection points. Buying large amounts of heat-shrink shielding can be more expensive, and if not properly heated can leave gaps in the protective

coating allowing particulates and/or water to intrude onto the wire. But when applied correctly it is clean and protects the wire and its signal just as proficient as any other alternative.

We also needed to consider which of the two power systems would provide better quality of service and stability for the entirety of the system. Using a prebuilt power supply to convert the 120V AC to DC will give the entire device enough power to operate all the available components in the system. The drawback is that it wouldn't be built by hand so we wouldn't have that experience. A lithium ion-based battery pack system has the flexibility of the device being able to be placed anywhere with no limitations of a needed wall outlet to operate, it also would be able to be recharged and reused over time. The downside of this system would be the fact that the device is reliant on a battery-based system along with the fact that a battery monitoring system is also needed to monitor battery life which adds another level of complexity needed to monitor the system. The last option would be to design and build a power supply on our own, this can come with risks of its own being the fact that our sensors need a very precise voltage to operate properly and give accurate measurements for the system to decode, without mentioning that if designed poorly or inaccurately we could destroy all of the components of our system leaving us having to respin the design and repurchasing components. The only positive side is we would have experience of designing our own power systems.

Along the lines of us trying to decide which material to mount the components to we could choose a metal plate. A metal plate would be stable with little flex that would allow us to easily mount the components with little fear of the mounting board breaking. The downside of this mounting system is the fact that many of our components are bare PCBs that have throughput holes because of this the boards could short on the metal plate and be damaged or could damage the other components on the board. This could be diverted by applying a coating on the base of these components to protect their boards, but this would take extra time and effort that could be diverted if a different mounting board is used. The second option would be to use a fiberglass board for mounting the components. This mounting board would allow us to not have to worry about the components shorting on the mounting board which alleviates a large issue that could arise with other boards. The downside is that this board type is more flimsy but if we further mount this board to the greenhouse it could alleviate this issue.

Selected Design

For wire protection with exposure to water and other elements the decided design was to implement liquid silicon wire coating for waterproofing the sensor connections into the water tanks. This protection type is easy to apply and can be patched or covered if there are exposed spots in the coating without needing to remove the rest of the coating. For wire based connections we decided to use heat shrink and electrical tape depending on the situation, with heat shrink on

wire to wire connections and electrical tape on the larger connections where heat shrink wouldn't be able to cover the entire connection.

A PCB will be implemented into the system but mainly used for base connections from the sensor and MCU, this will allow for easy clean connections that won't hang off of the MCU. The design will possibly also implement the function of monitoring signal integrity coming from the sensors to ensure the right voltages and digital signals are at their proper strength.

System power is going to implement both options using a prefabbed system which would alleviate the pressure of the student designed system failing or over amping/volting other components. The main power supply will be a wall socket AC to DC converter that will rectify the signal into DC and then be split to each of the tiers and components. The battery array which is a stretch goal will be in case of power failure and outage that will provide power to key components i.e. the SOC and/or the MCU to send out a warning that power has faulted to the system and needs user administration.

For mounting the system, we decided to mount it on the bottom shelf. This will keep the system easy to access in case of system issues or system faults. On the other hand we will have to waterproof entirely all tiers to ensure that there are no leaks to maintain system integrity, if there is a chance of a leak it will need to be addressed immediately.

5.3 - API Brainstorming

As part of the design process for the backend, the team discussed the different options for various parts of the backend. First, several possibilities were researched and considered. After finding several working possibilities, the different options were evaluated for their strengths and weaknesses and chosen based on their ability to fit into our project while meeting all our backend's needs.

We also took into consideration the scalability for each option. Although Pocket 'Ponics is currently a very small system, if it grows in the future, scalability will be important to maintaining it. If we consider scale needs now, we can prevent the necessity of a system redesign later, and ensure that Pocket 'Ponics will continue to work smoothly, regardless of scale.

Possibilities Considered

When deciding on a package manager, the team considered two choices: npm and yarn. To choose a package manager, we considered a few parameters: the number of useful packages available, the past experience with the package manager, and the compatibility with Node.js.

When considering the backend server hosting service, three options were considered: Amazon Web Services, Heroku and Pivotal Cloud Foundry. When choosing a server hosting service, we considered the number of resources given under the free tier, the past experience the team has with the service and the advice given by faculty.

When organizing the backend REST API endpoints, the team looked at organizing the endpoints based on the action of the endpoint. For example, handling sensor data would be grouped together, regardless of whether the mobile app or the sensor grid were interacting with that endpoint. Additionally, the team also looked at grouping the endpoints based on the part of the project that the endpoints would be used by. For example, the sensor grid endpoints would be grouped together, and the mobile app endpoints would be grouped together. However, the authentication endpoints would be grouped separately.

The team was contemplating on using two types of databases: relational and non-relational. When we thought about the sensor data the sensor grid would be posting to the backend and the other data that would be stored in the database, we found relationships between data points started to form. We also considered the flexibility needs of our database. When choosing a service to host our database on, we looked at Amazon Aurora or Amazon Relational Database Service for MySQL. We compared the costs and scalability for choosing a database service.

For authentication, the team looked at using Google API for providing the login with Google option to users or managing the authentication ourselves. Using the Google API would require users to create a Google account if they didn't have one. Managing authentication ourselves would also require more implementation on our part.

For the flow of data between the backend and the sensor grid, we examined two approaches. The first approach dictated the sensor grid make POST requests to the backend for providing sensor data. The second approach dictated the backend poll the sensor grids for readings at a set time interval.

Design Comparison

The first thing that we considered was which of the two package managers, has the most packages and of those packages, how many would be useful for our project. npm has over 1 million packages, with 30 billion packages downloaded every month and 10 million developers actively using npm as a package manager. Yarn uses npm as a registry, but also has a performance advantage over npm, and stores the exact versions of dependencies in a yarn.lock file

We also considered which of the two package managers team members are the most familiar with. The team members responsible for building the backend have several years of experience using npm. Additionally, the team members used npm for building a REST API which means the experience will be very similar to building the API for this project. Another team member does have experience using yarn, but they are primarily focused on building the frontend so their experience will not be useful when building the backend using the package managers being considered.

Of the three hosting services, we looked at which one has the most resources allocated under the free tier, since affordability is an important constraint of our project. Amazon Web Services provides 750 hours per month of EC2 computing resources and 750 hours per month of RDS storage resources for 12 months. Heroku provides 512MB of RAM with 1 web / 1 worker and 10,000 rows of database storage. Pivotal Cloud Foundry provides 87$ of one-time free tier credit with an estimated monthly cost of $21.60 for 512 MB RAM and 2 app instances. The database service provides 5MB and 4 connections for free.

When looking at the three hosting services, we also took into account which hosting service team members are most familiar with. Amazon Web Services would be a great learning experience for the team because of its popularity in the industry and the lack of experience the team members have with the service, but it does build on a framework that the team members are familiar with, providing a good starting point for the team. One team member has previous experience with Heroku but they are primarily focused on building the mobile app. The team

members assigned to backend have one year of past experience using Pivotal Cloud Foundry. However, other team members responsible for the backend don't have experience with Pivotal Cloud Foundry.

We also considered which of the three hosting services were recommended by faculty. When speaking to Dr. Richie (one of the ECE senior design professors) about our project, he recommended using Amazon Web Services (AWS) because of its popularity in the industry. Because of its popularity in the industry, having a project that used AWS could help team members when searching for a job after graduation. The computer science teaching assistants also recommended Amazon Web Services, as they are easy to scale and can be spun up with minimal effort.

When looking at how to organize our API endpoints,we considered whether it made more logical sense to organize endpoints based on their actions or based on who was using the endpoints. If the endpoints for managing sensor data were grouped together, it might be easier to find all the endpoints available for performing a specific action. If the endpoints were organized based on who was primarily using them, it would be easier to find all the endpoints a specific part of the project could use. For example, one group of endpoints would be all the possible endpoints the sensor grid had available to it.

We also considered if it would it make more sense to have authentication endpoints in both groups of endpoints or have one group of endpoints just for authentication. Authentication endpoints are going to be very few, and it might be easier to store them in a separate group so that it is easier to find them and manage them. Authentication endpoints are mostly for the mobile app, but they should be separated from the mobile app endpoints for readability.

Furthermore, we looked at whether each sensor type should have its own endpoint, or if there should be one endpoint for all sensors to post their readings. Making separate endpoints would allow different sensors to use different endpoints and distinguish the differences between the readings. Since the readings are very similar, having a separate endpoint for each sensor type would be redundant and one endpoint could accommodate all readings.

For data storage, we considered whether we should store our data in a relational database or a non-relational database. There began to appear clear relationships between data points and using a relational database would make retrieval of data easier. A non-relational database would be more flexible to changes in the data stored in the database and the structure of the database.

We also looked at whether we should host our database using Amazon Aurora or if we should use Amazon Relational Database Service (RDS) for MySQL. Amazon Aurora provides us with more scalability options, while Amazon RDS is limited in scalability. Amazon RDS is more cost-effective than Aurora.

For authentication, we looked at whether or not we should require users to create a Google account to use our application. Users on Android devices already have a Google account so they wouldn't be inconvenienced by this requirement. Users on iOS devices may not have a Google account so they would be required to create a Google account to use our product.

In the absence of Google account authentication, we had to consider if managing authentication ourselves would become overcomplicated and more work than needed. It would require more implementation on our parts, but we would avoid the inconvenience that some users may face when using the product if they don't have a Google account. Users would not be required to create a Google account for the sole purpose of using our product

Finally, we looked at how we would receive data from the greenhouse. We considered a POST data approach versus a polling approach, and the pros and cons of each approach. The first approach would allow the backend to avoid keeping track of the IP addresses and ports for each sensor grid for all users. Under the second approach, the sensor grid would have to be able to receive requests from the backend, which may be more costly. For the second approach, if the sensor grid was temporarily unavailable, it may be difficult for the backend to get readings from the sensor grid. More logic would be required to retry reaching that sensor grid at a later time

Selected Design

For the backend server, our team decided to use Node.js because of their past experience with the platform. Because the team has previously successfully built a REST API using Node.js they decided to continue to use Node.js for this project. Additionally, since the frontend will be written in React Native/Javascript, utilizing Node.js reduces the number of languages used in the project and promotes internal consistency.

When deciding on a package manager, the team considered two choices: npm and yarn. Because the team had more experience using npm and was more familiar with what npm had to offer in terms of packages, we decided to choose npm over yarn. Another team member had previous experience with yarn but was not one of the team members working on the backend, so we decided to choose npm.

When considering the backend server hosting, the three options were Amazon Web Services, Heroku and Pivotal Cloud Foundry. Team members had experience using all three options, but Dr. Richie advised us to use Amazon Web Services because it would give us experience using a product that was

commonly used in the industry. Because of his advice, we decided to use Amazon Web Services for our backend server hosting.

For grouping the endpoints in the backend REST API, we considered separating the endpoints based on the actions those endpoints were performing or separating them based on the part of the project that would be interacting with them. We decided to organize them based on the part of the project that was using those endpoints; separating the endpoints for the mobile app from the sensor grid's endpoints. Additionally, there would be another group of endpoints for authentication endpoints and a group of endpoints for the administrator's portal.

For choosing a backend, we considered Amazon Relational Database Service and Amazon Aurora. We decided to use a relational database because it would make retrieval easier, and the relationships between data points could be accommodated better. While we would have to sacrifice flexibility, we believe that there would very rarely be any changes in the way we store our data or the data we store in the database, so lack of flexibility is not a concern. After selecting Amazon Web Services for hosting the backend server, choosing AWS for storing the database was instinctive. However, we still had to choose whether to use Amazon Aurora or RDS MySQL. We chose MySQL because the costs are lower.

We considered using the Google API to manage authentication or to implement authentication ourselves. We decided to implement authentication ourselves, to avoid inconveniencing the users who don't already have a Google account. This allows all users to easily sign up for our product and reduces the barriers for entry for certain users who may not use the Google platform.

When deciding on the flow of data between the backend and the sensor grid, we considered two approaches. We decided the sensor grid should make POST requests to the backend because it would be easier for the sensor grid to make POST requests to the backend than the backend keeping track of the active sensor grids and polling them. However, the backend will make POST requests to the sensor grid to toggle the light sources on and off.

The backend will query the database to retrieve sensor readings or other miscellaneous data stored in the database. The backend will execute queries to insert data from the sensor grid into the database. The mobile app will make GET requests to the backend to retrieve sensor readings. We also considered a polling approach for the mobile app but found it would be more effective for the mobile app to request data as needed. The mobile app will make POST requests to the backend to send user preference data, commands to greenhouse, configuration changes.

5.4 - Database Brainstorming

The entity-relationship models for the database were designed with the users table in mind. Their information had to be the first in the logic of the database in order to store all the other incoming information under the correct user. How the information was connected was another problem. Which way should the data be stored? How should the information in the sensors be organized? Then, it was a matter of trying out the different relationships that connect each table in the most efficient way possible.

Possibilities Considered

The first consideration (Figure 5.4.1) was which entities tables to include. Should sensor data be on its own table or should it be combined with greenhouse data? What information from the greenhouse should we keep? Do we save it in another table? How long should the data be kept? Different models were created testing out how the schema would look if the sensor readings for the plants were all stored in the same table or if it was better to separate individual greenhouses. Most models keep the data separate since it makes it easier to retrieve specific information and it organizes it by what information you might be looking for; for example, looking to retrieve the sensor readings of the third tier in the second greenhouse. Another aspect to consider is where to place seedling data, since that information differs from the other plant needs. Since the seedlings are very simple to take care of, it was decided that not many attributes were going to be needed for them and thus they were placed under the greenhouse table.

The other consideration was figuring out which is the best way to relate the tables (Figure 5.4.2). The user data needs to be connected to the sensor readings; the sensor readings will be connected to the individual greenhouses that they're in. If the sensor readings are connected to an individual greenhouse then it makes sense to connect the user to the greenhouse first and then stem off from the greenhouse to individual tiers. The ideal conditions for each plant to grow must be able to retrieve data that includes the plant's current conditions and the identifier to where that plant is located. This means that the plant ideal table has to be connected to the sensor grid (current plant conditions) and the greenhouse table, since, combined, both of these contain the data that will be needed to examine the plant's conditions and adjust accordingly. Users might benefit from being able to see water or nutrient usage. Or maybe that can just be kept track from the greenhouse tanks. History was added as the idea to display this information later was being considered. If historical data is a new entity then attaching it to either the sensor readings (plant conditions) or greenhouse tables (the tanks) is the way to design it, since the information can be accessed as it comes in and stored on a clock.

Figure 5.4.1: Database Brainstorming Diagrams: Entity Tables

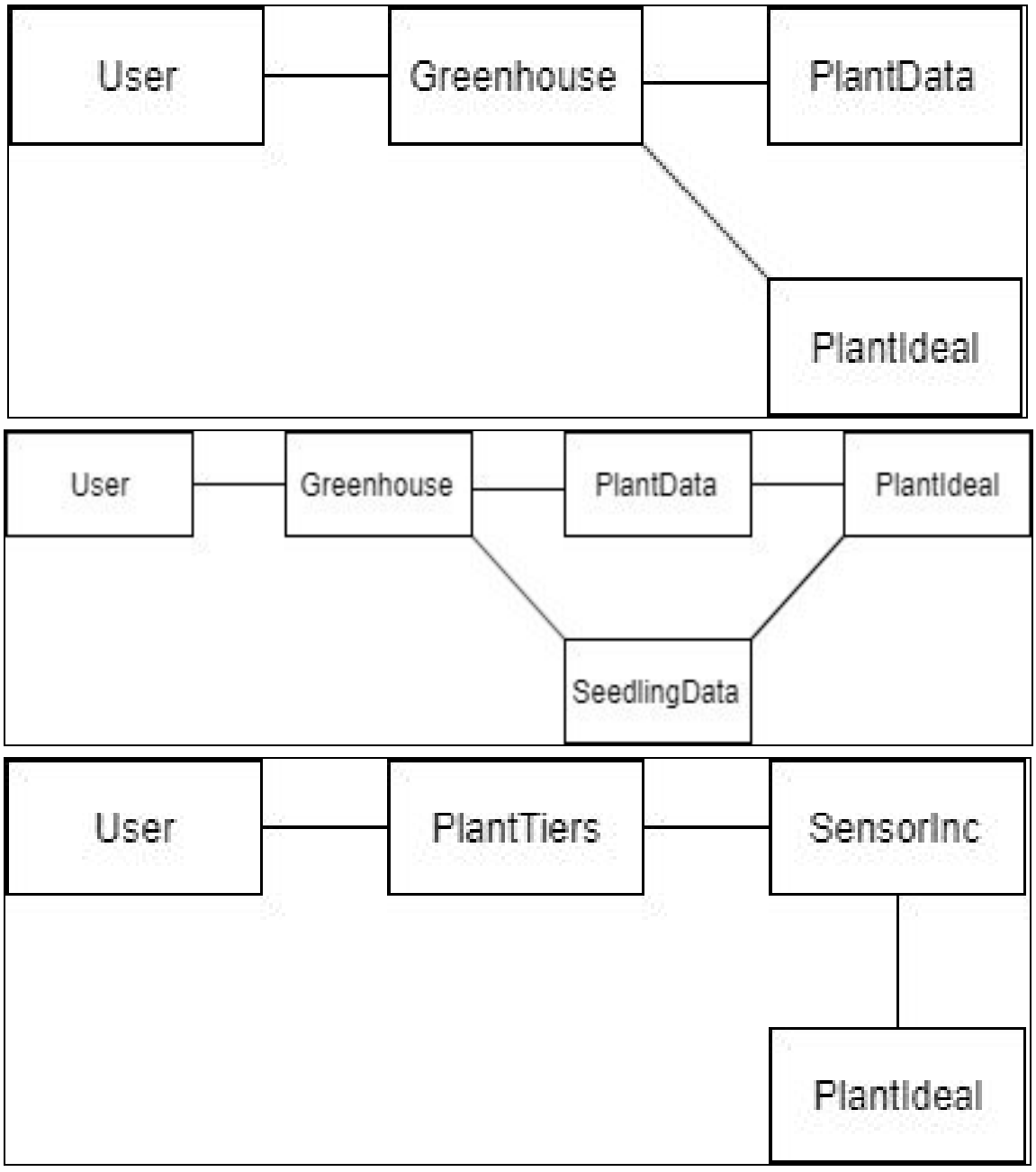
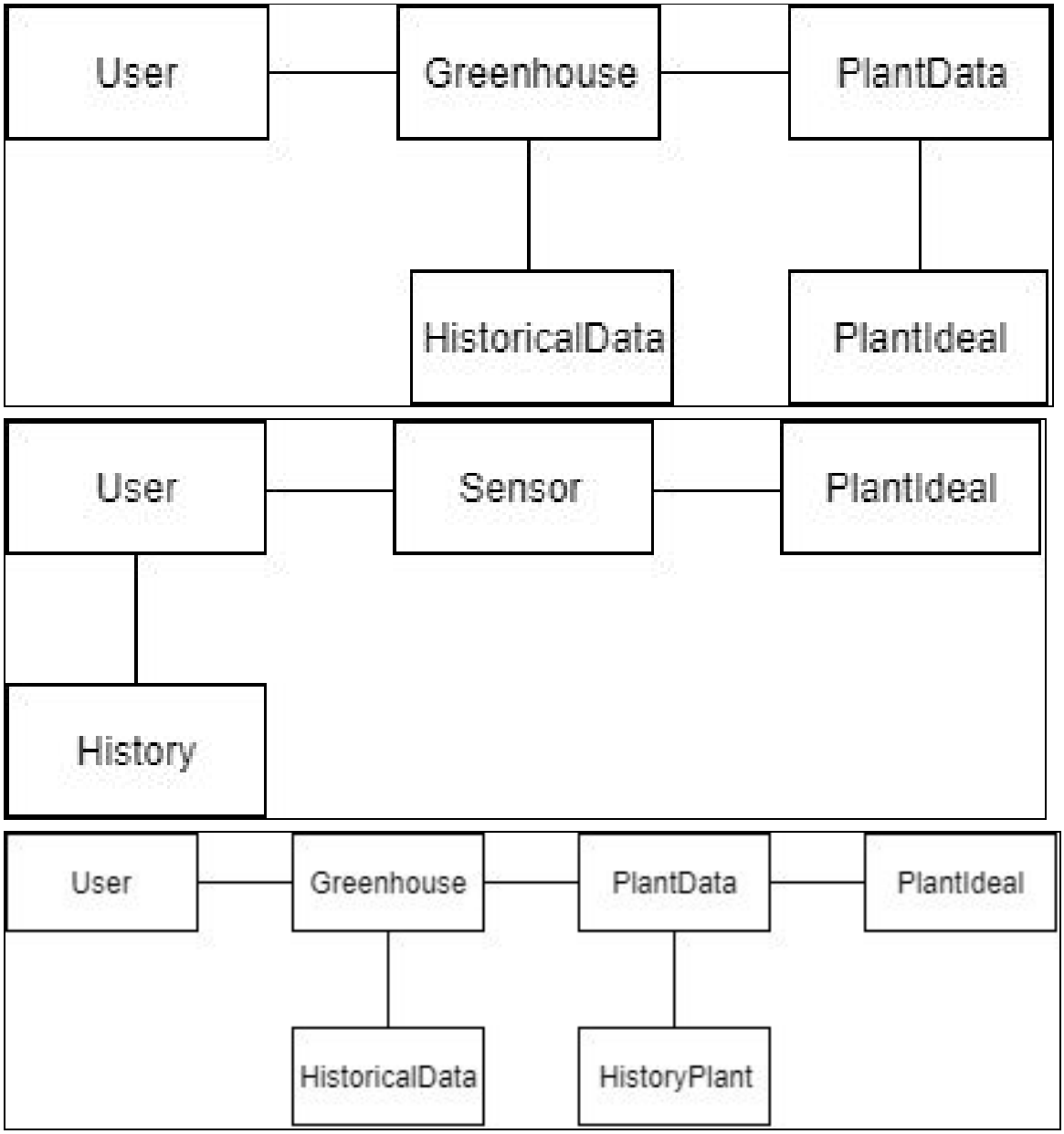


Figure 5.4.2: Database Brainstorming Diagrams: Relating Entity Tables



Design Comparison

First we considered whether or not the greenhouse should be treated as its own entity, or if the greenhouse data could be incorporated under another entity, such as users. One table for all sensor data in all greenhouses under the same user, this includes all the current plant conditions, tanks and light usage. Greenhouses can have their own table, storing their unique ID, user ID, and sensor ID, along with a tank and maybe seedling information. We also looked at whether or not seedlings should get their own table, or if they should be a part of the greenhouse table. Puting seedling data in the same table as all the other plant information is one option, but the problem is that seedlings don't have the same attributes as the other plants since they have fewer needs. Grouping seedlings in the greenhouse table because of those different needs may solve this issue.

We also looked at which table needed to be connected to the ideal plant growing conditions. Connecting to the greenhouse table can allow for ideal plant conditions to be retrieved without having the appropriate plant already. This may be useful if users want to meet the requirements for taking care of different plants. Connecting the ideal plant conditions to the plant data since they share similar attributes and the current plant conditions needs to be compared to the ideal conditions. Furthermore, connecting the ideal table to the tier table allows for a simple, many-to-one relationship of plants.

In addition, we looked at whether historical data should be stored by greenhouse or by sensor. Relating history with the greenhouse table allows for retrieval of the overall history of a greenhouse as well as plant data. Relating it with plant data allows for the retrieval of only the plant information over time. Since plants are likely to change fairly frequently, it makes more sense to associate the historical data with the greenhouse, so that users can track usage trends over time.

Finally, we looked at what attributes each table would need. For users, the list of attributes was fairly small: the chosen username, the password, hashed before it's stored and each user should have a unique id. The greenhouses need attributes that were a little more complicated: a greenhouse identifier, water and nutrient levels for the tanks, the current power source for the greenhouse and when the seedlings were planted. The tiers would mostly need information about the sensors that they contain, including plant id, the current growth stage, a column for each condition, and the number of plants in a tier, to adjust calculations accordingly. The plant ideal table would need information about the conditions necessary for plants: the minimal pH, maximal pH, minimum ec, maximum ec, and the amount of light that the plants would need.

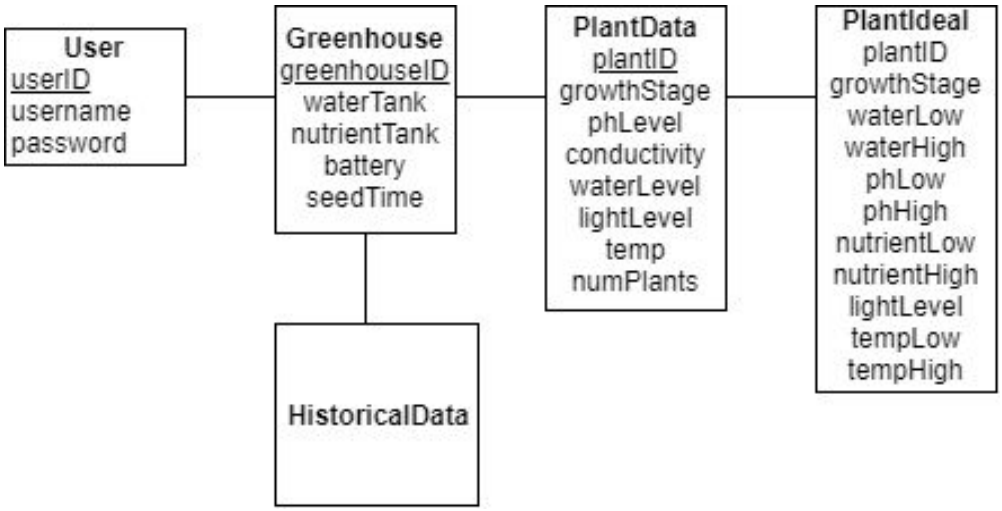
Selected Design

Figure 5.4.3 shows the selected database design. User information is stored first to be able to place incoming data appropriately. The information is then connect to individual greenhouses. Each user will have their own attributes that need to be kept track of, such as the water and nutrient levels, as well as storing seedling information since they will only need to be watered at this stage. Every user will have at least one greenhouse they connect to. From there, the greenhouse table will have two connections. It will feed it's current conditions to another table, historical data, which will then store it for future use. Additionally, the greenhouse will also have ties to the sensor readings. The link between these tables will be the greenhouse primary key, greenhouse id.

The sensor readings will be stored under the plant data table. This will store the current conditions of every plant in the greenhouse, storing it by tier. Every plant will be uniquely identified by an id and this allows for the plant ideal table to be used to check when conditions are not being met. Sensor readings, as mentioned above, will be connected to the greenhouse table in addition to being connected to the plant ideal table, which allows the intervals that are stored there to be used so that when a reading falls above or below the stored interval, the system can fix itself.

The plant ideal table will be specific plants where we manually store the ideal conditions for the plants of our choosing. The tables will hold low and high values for every condition that the plants have, which includes water, nutrients, pH levels and more. The low and high values allow the backend to determine the adjustments needed to be made to fix the out of bounds condition.

Figure 5.4.3: Selected Database Design



5.5 - App Logic Brainstorming

When brainstorming the app logic, the main consideration was to develop a listing of what screens would be needed for the app, and how those pages were related to each other. We focused primarily on intuitive navigation, to make sure that users saw the information that they needed with as little navigation as possible, while screens that were less used would be deprioritized.

Possibilities Considered

The first iteration included only a login screen, settings, a list of greenhouses, and a detail view for individual greenhouses (Figure 5.5.1, left side). Then the consideration of individual tiers came up, and ideas were generated where tier information could be reached directly from the greenhouse list, or only from the greenhouse detail page (see Figure 5.5.1, right side).

We also considered the necessity for a screen to register new greenhouses. The initial consideration had the greenhouse registration connected to the greenhouse list as a method of adding a new entry to the list (Figure 5.5.2). We also considered making the greenhouse registration available through the individual greenhouse screen (Figure 5.5.3, top). This makes the most sense when considering the individual greenhouse as the main entry point of the app, or when assuming that most users would only have a single greenhouse registered. (Figure 5.5.3, bottom)

Once we had established the necessity of a registration screen, we also had to look at what the main entry point would be for the app. If most users would only have a single greenhouse, it made more sense to present their greenhouse as the main landing page, with the list presented as a nested subview. (Figure 5.5.4) If, however, most users would be likely to own more than one greenhouse, it made more sense to present the greenhouse list as the main entry point, and the individual greenhouse view as a subview of the main list. Furthermore, the selected entry would likely inform whether or not users chose to have multiple greenhouses, in addition to being easier to use one way versus the other.

Another important consideration was whether or not tiers should be their own independent view, and how users should be able to navigate to those tier views. On one hand, it makes sense that users should be able to access the tiers directly from the individual greenhouse view, since tiers are a subunit of the greenhouse. On the other hand, it might be convenient for the users to be able to access the individual tiers directly from the greenhouse list, in order to expedite navigation.

The final consideration was the possible addition of a historical data record that maintained a log of all recordings. The options were considered for separate historical data records for both greenhouses and tiers; the greenhouse historical data would allow users to track the water and nutrient usage, as well as the battery power levels over time. The tier historical data would allow users to track the EC and pH fluctuations on a tier, as well as the light level patterns. Two different designs were considered for the independent historical data: one with the individual greenhouse as the main entry point (Figure 5.5.5, top) and one with the greenhouse list as the main entry point (Figure 5.5.5, bottom). We also considered designs that included only a single historical data screen, that would either include all the data from the different tiers, or else exclude the tier data all together. (Figure 5.5.6)

Figure 5.5.1: Possible App Logic Diagrams: First and Second Iterations

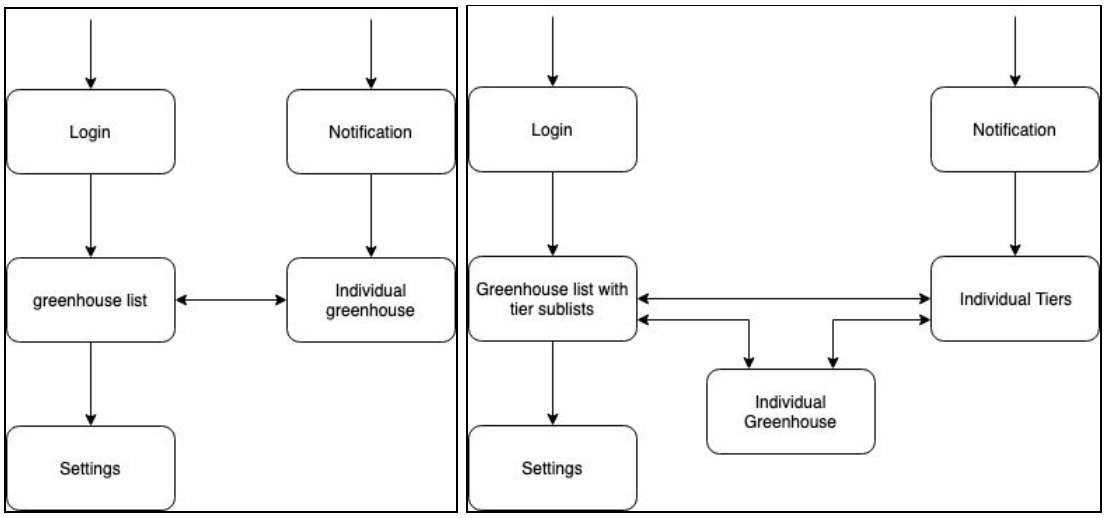


Figure 5.5.2: Possible App Logic Diagrams: With Greenhouse Registration

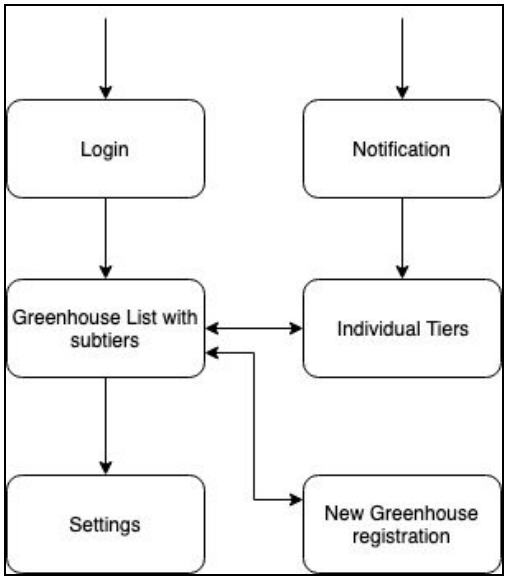
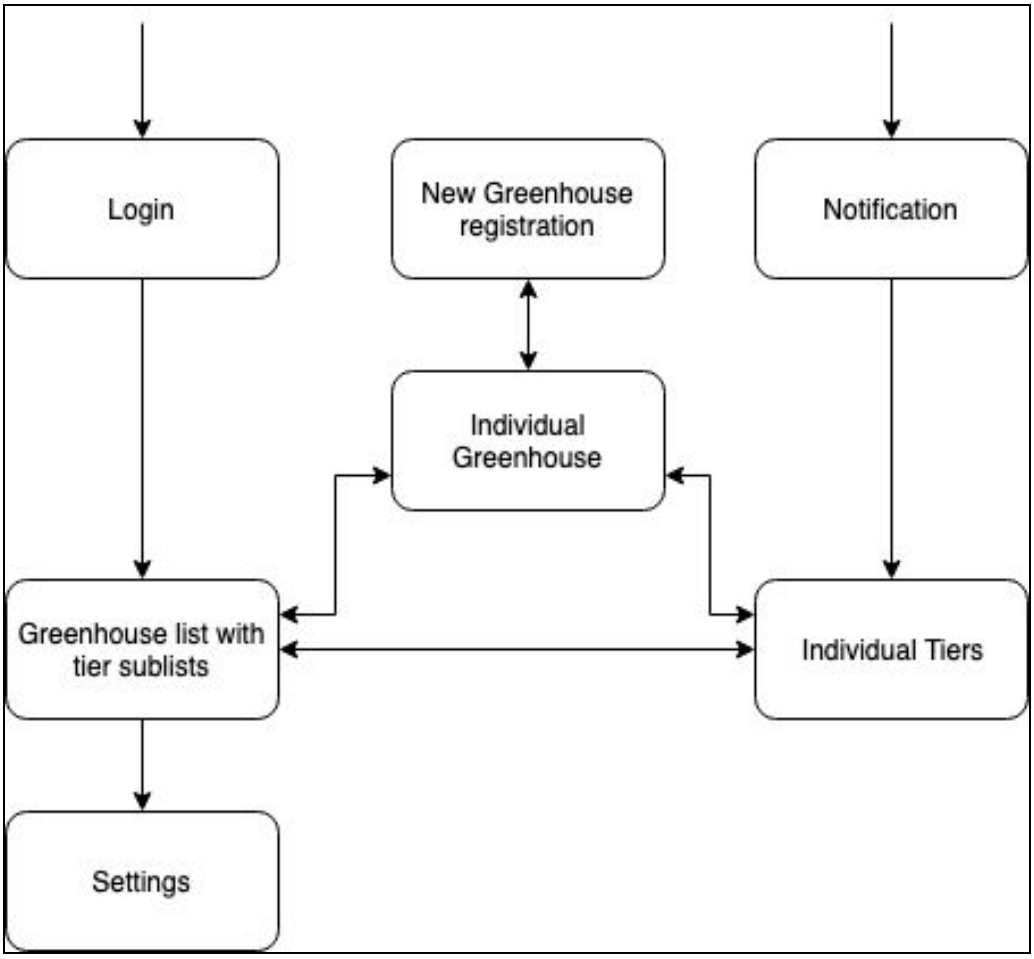


Figure 5.5.3: Possible App Logic Diagrams: Greenhouse Registration Options



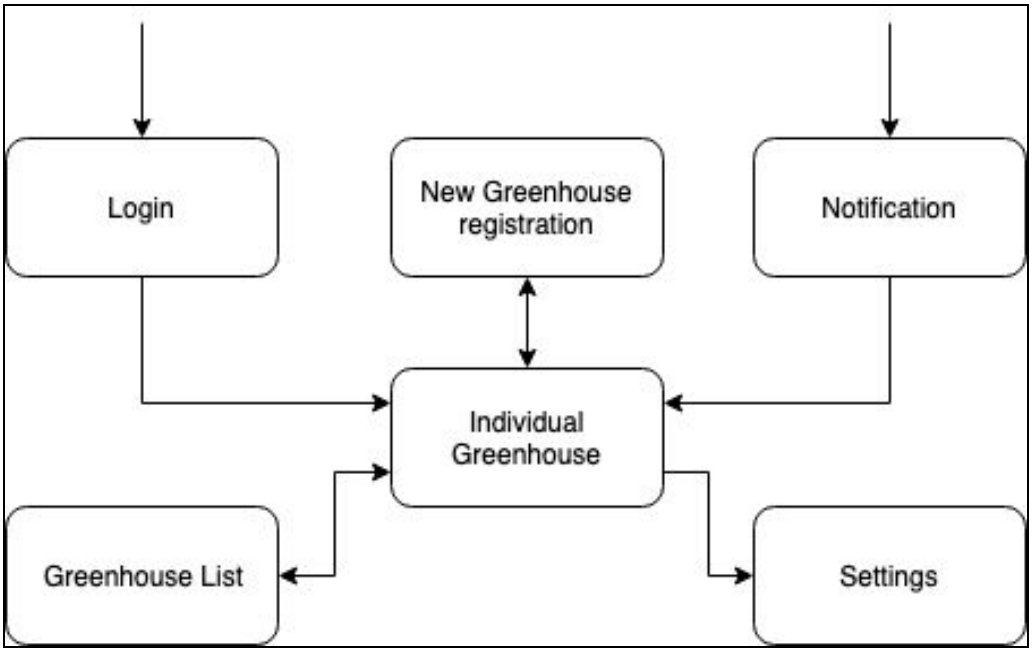
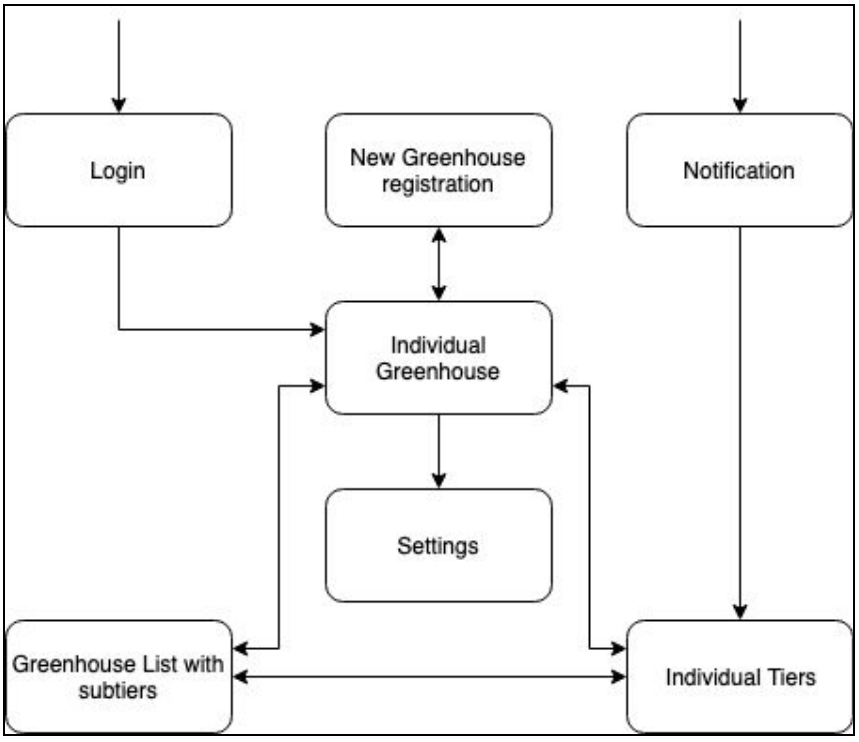


Figure 5.5.4: Possible App Logic Diagrams: Individual Greenhouses as Entries



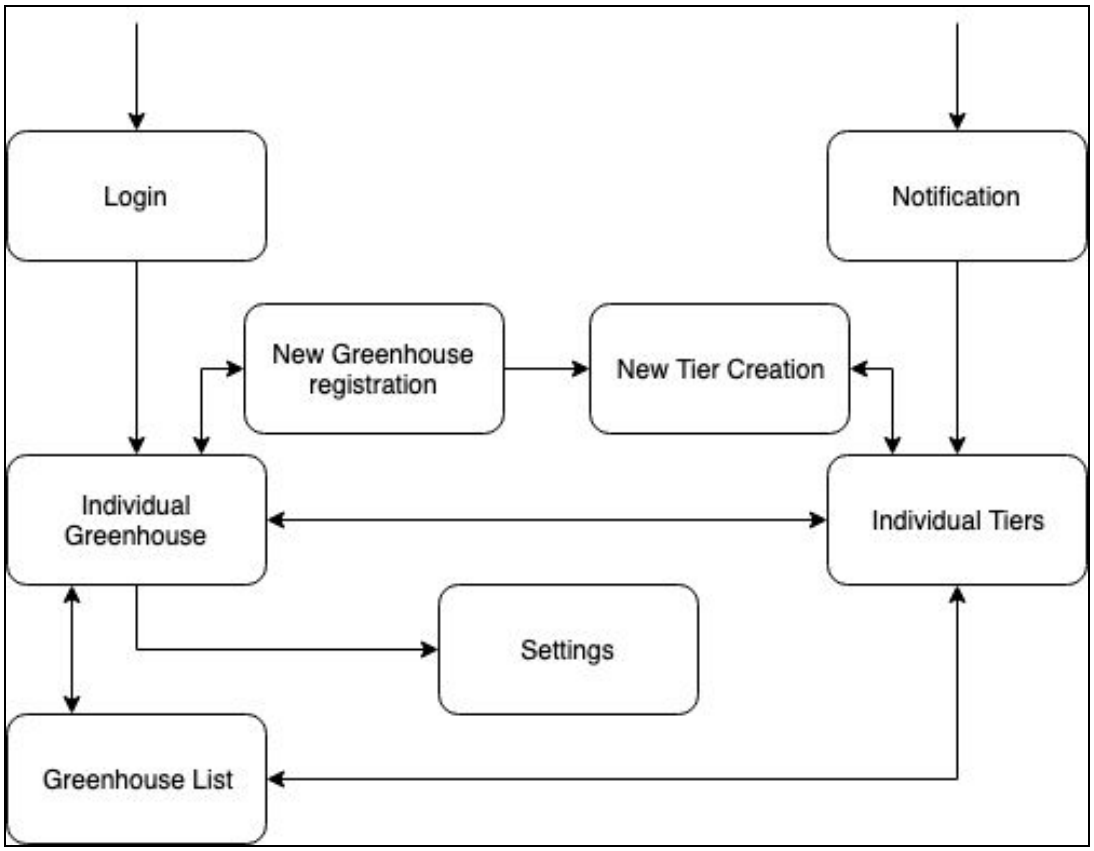
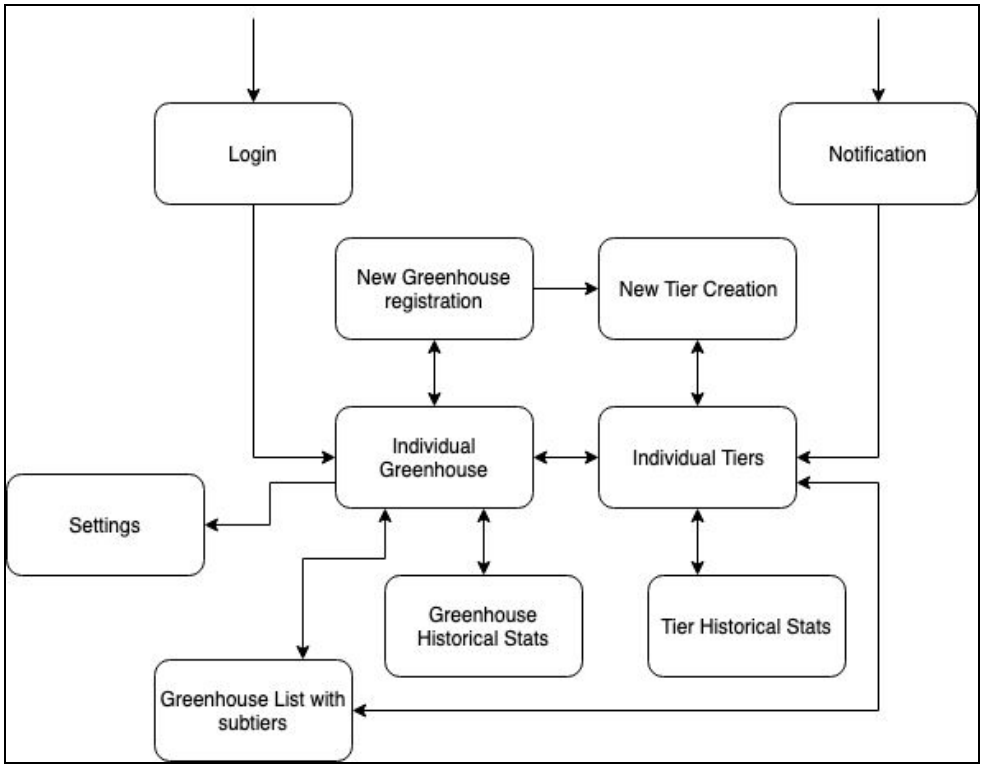


Figure 5.5.5: Possible App Logic Diagrams: Independent Historical Data Options



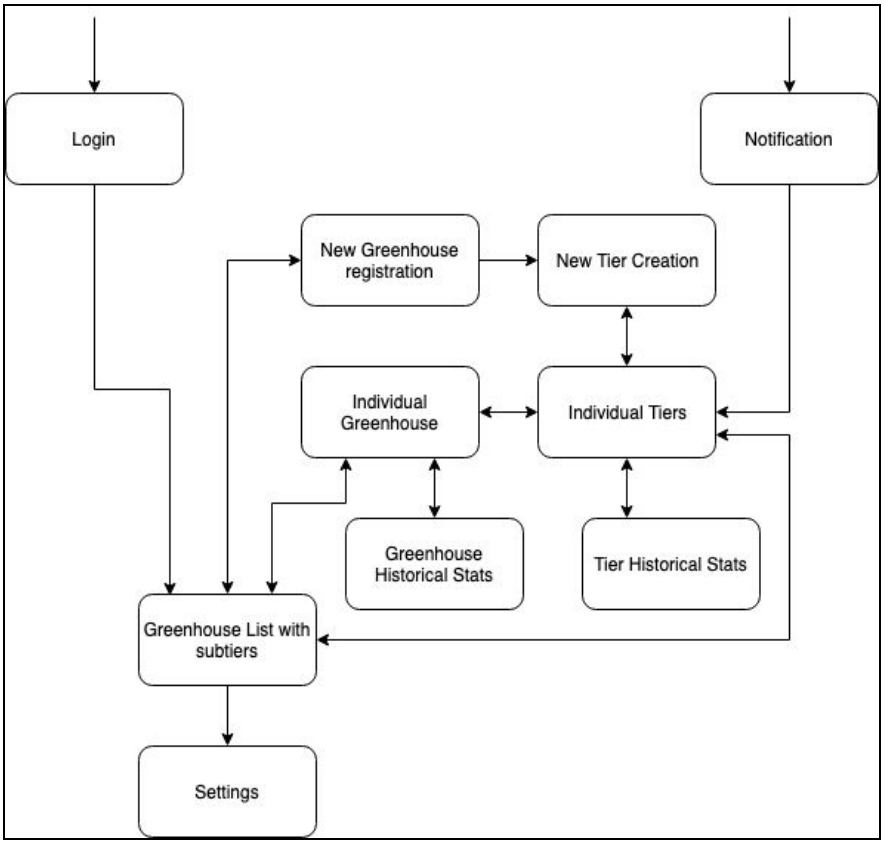
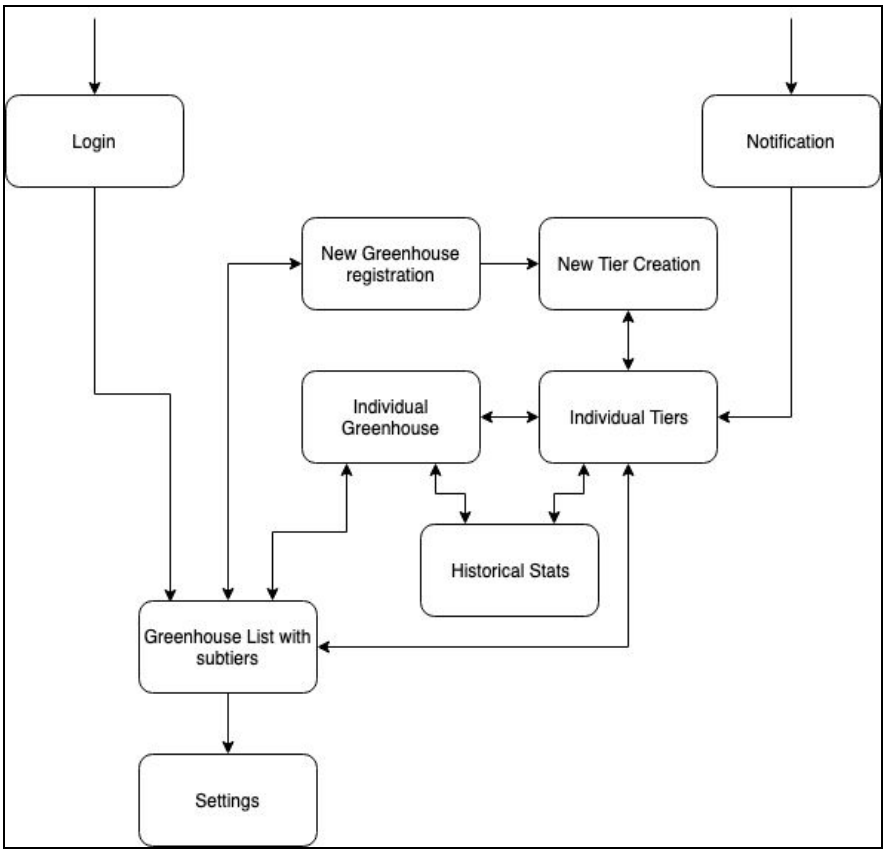
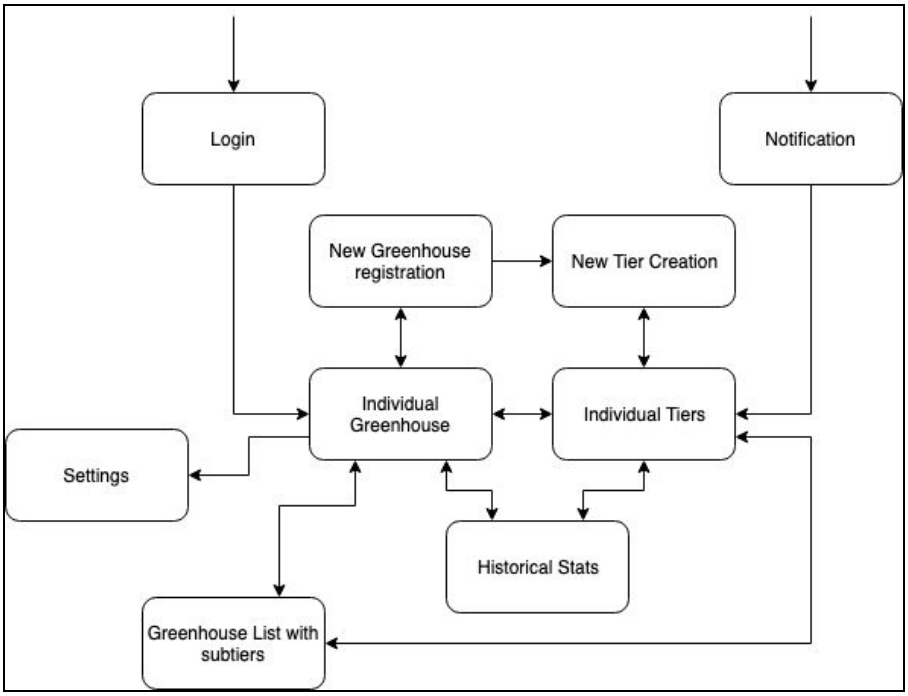


Figure 5.5.6: Possible App Logic Diagrams: Greenhouse Only Historical Stats





Design Comparison

The first major consideration was whether or not the greenhouse list should have subheadings for individual tiers, or if individual tiers should only be available from a specific greenhouse screen. Allowing subheadings for individual tiers allows users to quickly and easily access the data for a specific tier. Allowing subheadings for individual tiers locks the greenhouse list reduces the creative flexibility of the list, as any display would need to be able to show subheadings. Only displaying greenhouses in the greenhouse list makes the hierarchy very clear; greenhouse tiers only exist as a subset of a greenhouse, and should not be viewed in isolation. Only displaying greenhouses in the greenhouse list opens up the design flexibility for that display. Only displaying greenhouses in the greenhouse list ensures that users have to navigate through the greenhouse to get to tiers, and thus will always be informed about any issues that impact the whole greenhouse, such as low water-levels.

The second design consideration is whether greenhouse registration should be connected to the greenhouse list or to individual greenhouses. Connecting greenhouse registration to the list of greenhouses makes adding a new greenhouse logically equivalent to adding a new entry to the list. Connecting greenhouse registration to the individual greenhouse page makes adding a greenhouse logically equivalent to duplicating and modifying an existing greenhouse.

We also looked at whether or not individual greenhouses should have a chart of historical data, or if historical data was necessary at all for greenhouses. Having a chart of historical data makes it easier for users to identify trends in their greenhouses, such as times when the greenhouse may use more water or times when heat and light conditions may be less than ideal. Storing historical data can rapidly balloon the amount of storage space needed for each greenhouse, as all records will have to be stored individually.

In addition to historical data for greenhouses, we also looked at whether or not individual tiers or plants should have a chart of historical data, or if historical data was necessary at all for tiers or plants. Having historical data for individual tiers can help track trends, but the data may not be relevant beyond the lifespan of a plant. Storing historical data for tiers may balloon even faster than historical data for greenhouses because there are four tiers in each greenhouse.

We also considered if the main screen should be the greenhouse list or the screen for an individual greenhouse. If most users will only have one greenhouse, it makes sense to display that greenhouse as the main screen and create the list of greenhouses as a screen for atypical users. If most users will have more than one greenhouse, it makes sense to display the list of greenhouses as the main screen for easy navigation.

Furthermore, we looked at whether scrolling in between cards should be set to snap to center in a paginated manner, or be allowed to freely scroll like a list view. If the cards snap to the center, it will be easy for the user to determine exactly which bit they are looking at, but it may decrease the users ability to see all of the greenhouse data together. Also, it will be difficult for users whose screen size does not match the cards to see all of the information. If cards scroll freely, they will be easier to see and associate with one another, but they may result in users overshooting the information that they wanted to look at.

Selected Design

After careful consideration, the finalized version of the app logic was selected to maximize the ease of use for users, as previously shown in Figure 4.4.1 and Figure 4.4.2. The list of greenhouses was visualized as a series of greenhouse cards that the user can swipe left or right to exchange for one another. Swiping to the right on the last greenhouse in the list will open up the greenhouse registration screen, allowing users to add a new card to the list.

Within each card, clicking on an individual tier will take the user to the tier detail screen, while scrolling down will present the user with the greenhouse stats and the historical data. No historical data will be presented for individual tiers, as data would quickly lose its relevance when plants were cycled in and out.

When opening the app for the first time, users will be prompted to log in or create a new account. New users will be taken directly to the greenhouse registration screen while returning users will be presented with the first greenhouse card in their list of greenhouses. Notifications that bring the user in from outside the app's context will lead the user either to the greenhouse card that is related to the notification or to an individual tier's detail view.

Users will also be able to access their user information and the settings for the app from a simple menu button in the greenhouse card view. This button will be available above every greenhouse card that the user swipes through. Cards themselves will scroll freely within a greenhouse, but greenhouses will incrementally snap to center when scrolled side to side.

The final part of the design is the login flow. We decided that users should always be directed through login when they reopen the app, so that we can be sure that they always have up-to-date authentication credentials. For the most part, this will simply look like a loading screen, with auto sign in happening in the background using the users established and saved credentials. Even when clicking through a notification, users will be directed to an auto-signin that uses their saved password, and then redirected to the specific page that the notification was about when the sign in succeeds.

5.6 - GUI Brainstorming

The GUI brainstorming primarily focused on user accessibility. We wanted to design a user interface that made intuitive sense, so that the users would be able to navigate the app without needing a tutorial. We also wanted to make all of the app icons and images look like simplified versions of the plants that they are representing, so that our app has a unique and coherent visual presence.

Possibilities Considered

While designing this application, there were many considerations we discussed and worked through as a team. Almost every aspect of our user interface has evolved throughout the iteration process. Originally, we planned to implement the typical menu option used on most applications, which is a small icon of three horizontal lines in the top left corner. Then, once the user selects this icon, a menu bar would pop out of the left-hand side of the screen and the right-hand side of the screen would be darkened. After careful consideration, we decided to implement a card view style application. This made the need for a menu icon entirely obsolete and gave our UI a cleaner look. We decided to go this route because we felt it would give our application character and provide a more user-friendly experience.

It's always been important to us that we provide the user the ability to add several greenhouses to their application. Without this feature, users with more than one greenhouse would have to logout and login to a different account each time they wanted to switch between greenhouse. This is especially important for less developed countries, where smartphones are less common. With just one smartphone, produce for an entire village can be easily managed. For this feature, we decided to adopt the iPhone scheme of having a dot at the bottom of the screen for each greenhouse and then all the way to the right of the dots, having an addition symbol. This addition symbol signifies the creation of a new greenhouse. Users can swipe right/left or select the dots to flip through the card view of each of their greenhouses. Once they've reached the end of their preexisting greenhouses, if they swipe right once more or select the addition symbol, they will see the setup screen for a new greenhouse.

We had a bit of trouble deciding how exactly we wanted to tackle the issue of registering a new greenhouse, because it's something none of us have dealt with previously. After tossing around a few ideas, we decided to utilize QR codes, at the recommendation of our project manager. We have created a seamless series of steps that users must complete in order to add a greenhouse to their app. Once the process is complete, the greenhouse is fully functional and ready to grow!

Design Comparison

Most of our design comparison was done by iterative design drawings. We first sketched up some basic designs on paper, and then moved to Adobe XD as a prototyping software. As the designed were drawn and redrawn, they evolved from basic concepts (Figure 5.6.1) into more refined UI (Figure 5.6.2 and Figure 5.6.3).

Figure 5.6.1: GUI First Iteration Brainstorming Diagram

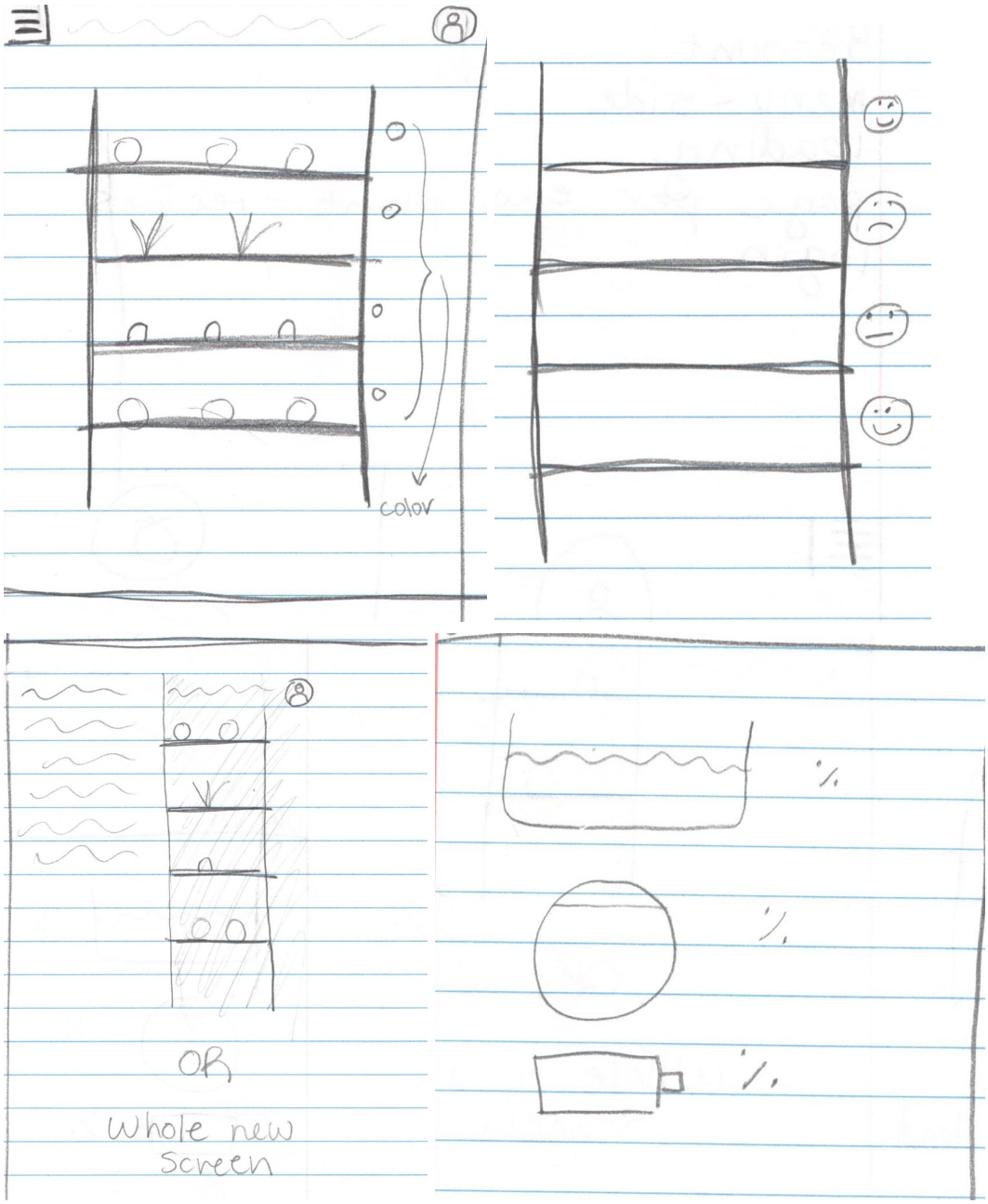


Figure 5.6.2: GUI Second Iteration Brainstorming Diagram

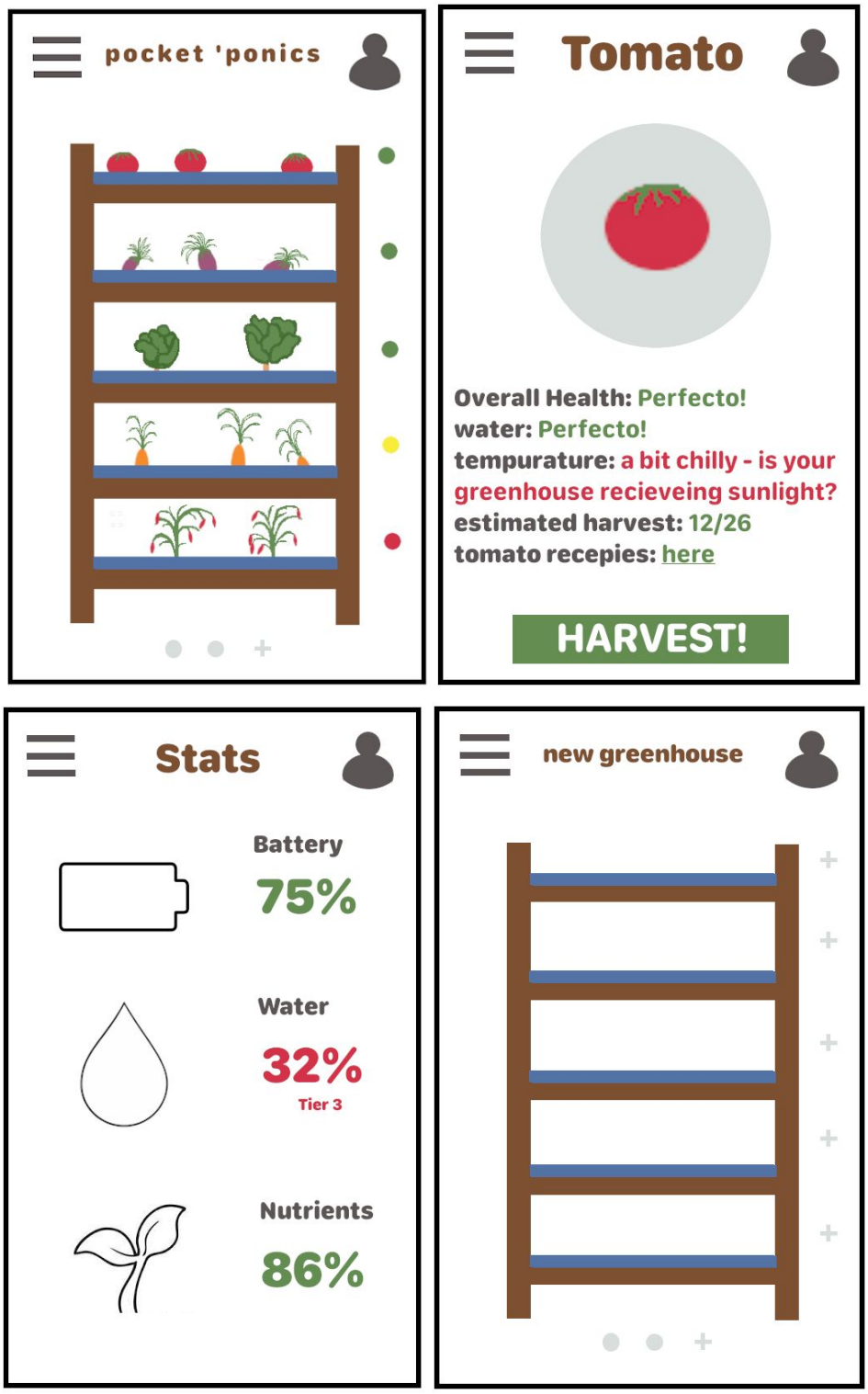
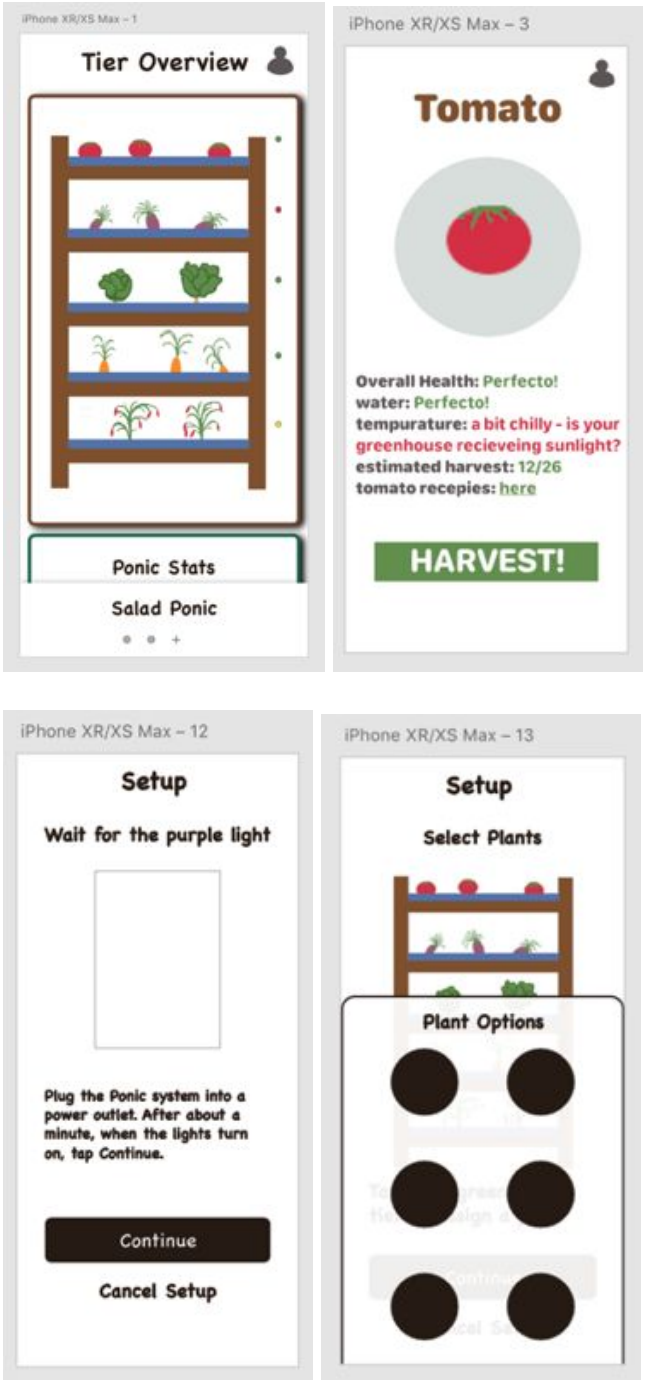


Figure 5.6.3: GUI Third Iteration Brainstorming Diagram



Selected Design

In the end, our selected design shown in Figure 5.6.4 used cartoony plants and simplified icons, along with a four part color scheme, to make sure that our app was easy to view and navigate. We also made sure that all the buttons were easy to find, and that actions taken made logical sense, and were intuitive.

Figure 5.6.4: Selected GUI Design

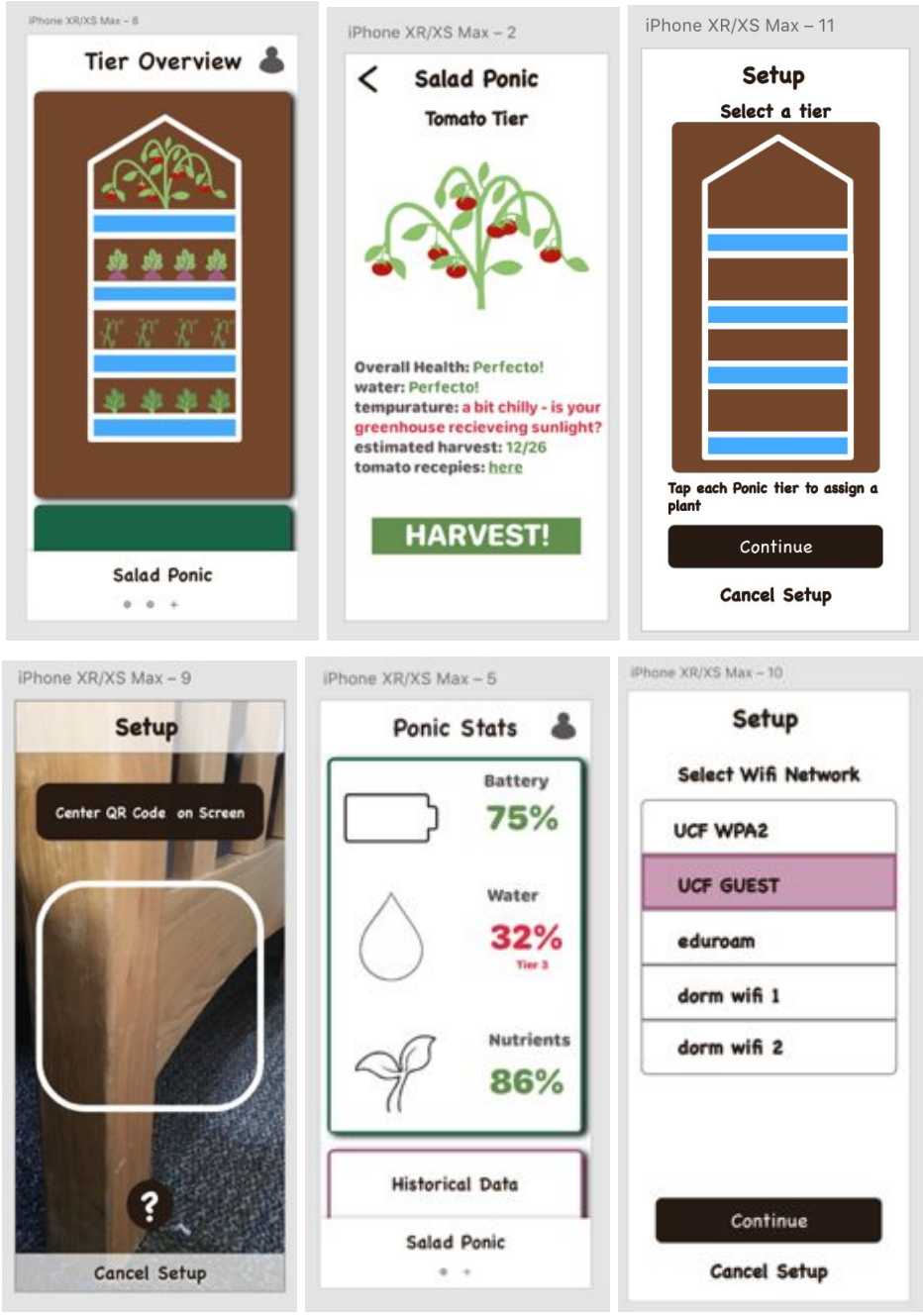
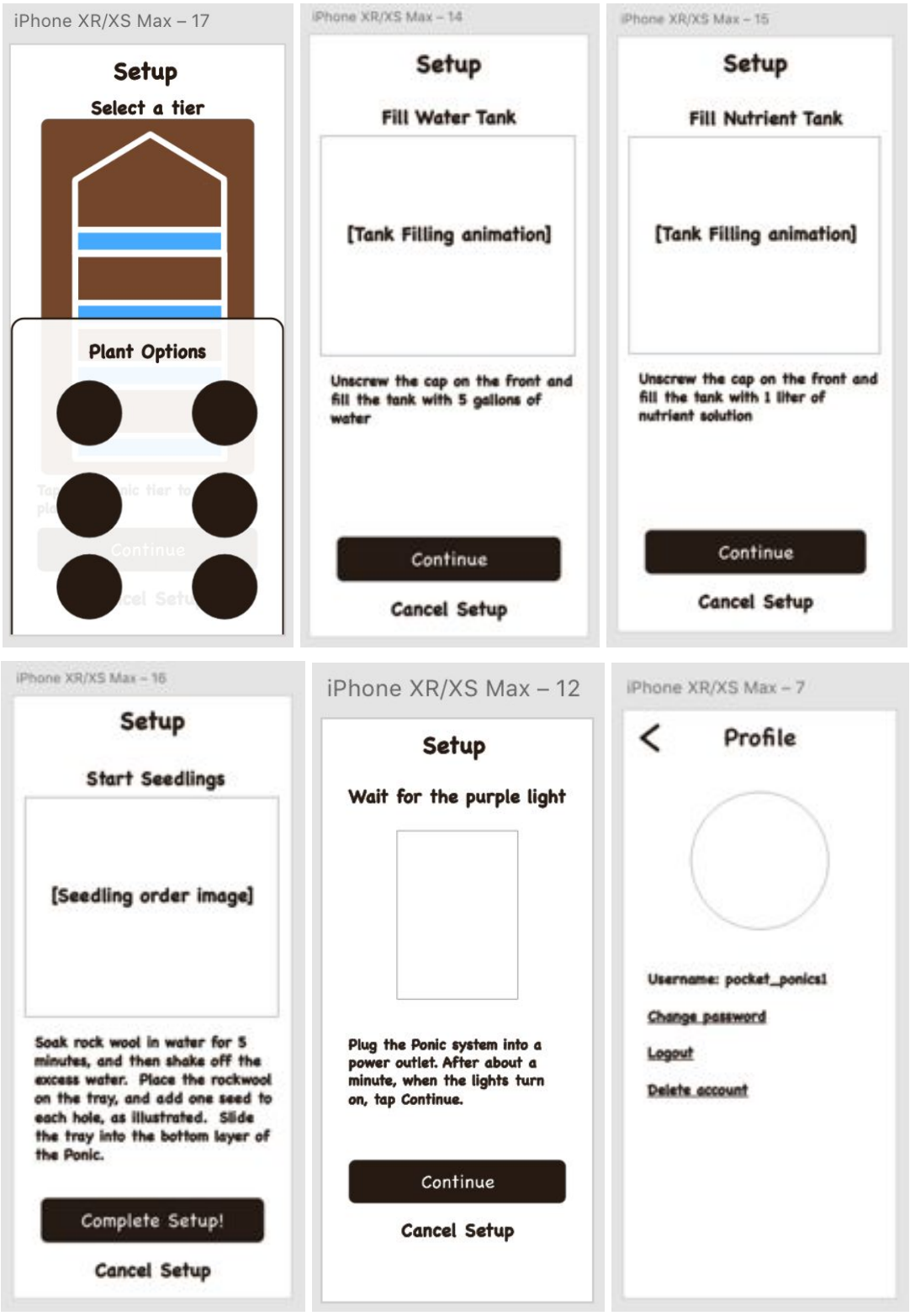


Figure 5.6.4: Selected GUI Design (Continued)



6.0 - Administrative Documentation

With a large project like Pocket 'Ponics, it is important to maintain concise administrative documentation about the project, including the project's budget, proposed timeline, and personnel. Establishing this documentation can affect design considerations of the project, so we sat down at the start of the semester and worked out project costs, internal deadlines, and a solid demarcation of which team members were in charge of various aspects of the project.

6.1 - Budget Proposal

As a student-sponsored project, Pocket 'Ponics has no financial backers or startup capital. As such, all expenses incurred will be divided evenly amongst the six students involved. All prices shown in Table 6.1.1 are estimated based on the current market and are subject to fluctuate before the project reaches completion.

Table 6.1.1: Cost Breakdown

Budget for Pocket 'Ponics				
Description	**Vendor**	**Price per Unit**	**Amount**	**Estimated Price**
Arduino	arduino.cc	$40.00	1	$40.00
pH Sensor Kit	amazon.com	$19.65	5	$98.25
Pump system	amazon.com	$11.98	2	$23.96
Water Tank	amazon.com	$19.33	2	$38.66
EC/TDS Sensor		$12.90	5	$64.50
Water Level Sensor	amazon.com	$2.13	8	$17.04
Lights	amazon.com	$13.99	1	$13.99
Liquid Electrical Tape	amazon.com	$6.98	1	$6.98
Step up Transformer		$10.10	1	$10.10
Step Down Transformer		$10.10	1	$10.10
PCB Fabrication		$50.00	1	$50.00
Construction Materials		$100	1	$100.00
Miscellaneous Electronics		$50.00	1	$50.00
Heroku Account	heroku.com	$0.00	1	$0.00
Hydroponic Plant Nutrient	amazon.com	$25.00	1	$25.00
Total Amount				**$548.58**

6.2 - Initial Project Milestones

Project milestones are shown in Table 6.2.1 (Senior Design 1) and Table 6.2.2 (Senior Design 2) as well as in Figure 6.2.1 (Gantt Chart).

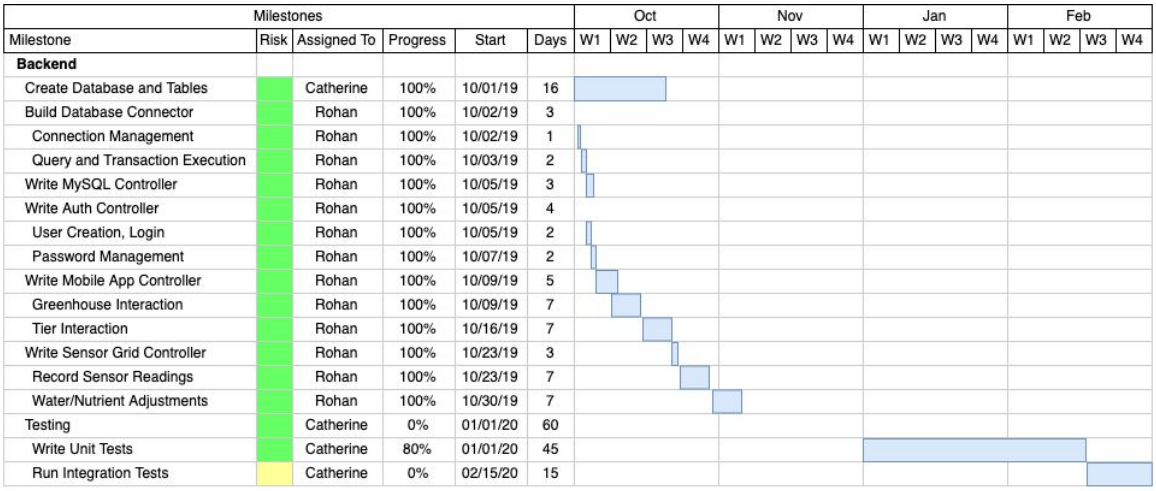
Table 6.2.1: Senior Design 1 Milestones

#	Task	Start	End	Status	Responsible
1	Project Selection & Role Assignment	08/25/19	09/10/19	Completed	All Members
	Senior Design Documents				
2	Initial Design Document	09/11/19	09/20/19	Completed	All Members
3	First Draft	09/22/19	10/31/19	Completed	All Members
4	Final Document	11/01/19	12/01/19	Completed	All Members
5	Group Meeting with Professors	09/25/19	09/25/19	Completed	All Members
	Research and Design				
6	Sensor Grid Configuration & Layout	09/21/19	10/20/19	Completed	Matthew, Graham
	Component Selection	09/21/19	10/05/19	Completed	Graham
	Interaction with Backend	10/06/19	10/20/19	Completed	Matthew
7	Backend API Endpoints and Database Schema	09/21/19	10/20/19	Completed	Rohan, Catherine
	API Endpoints	09/21/19	10/05/19	Completed	Rohan
	Database Design	10/06/19	10/20/19	Completed	Catherine
8	Frontend App Design	09/21/19	10/20/19	Completed	Elli, Alexandra
	User Interactions Diagram	09/21/19	10/05/19	Completed	Elli
	GUI Final Design	10/06/19	10/20/19	Completed	Elli, Alexandra
9	Final Block Design	10/20/19	10/25/19	Completed	All Members

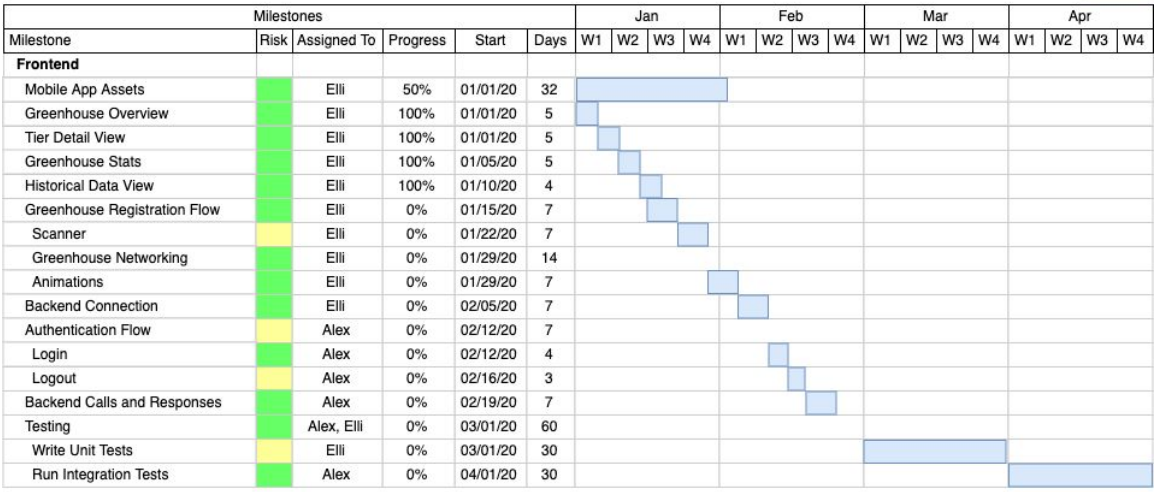
Table 6.2.2: Senior Design 2 Milestones

#	Task	Start	End	Status	Responsible
	Backend				
1	Create Database and Tables	10/01/19	10/01/19	Complete	Catherine
2	Build Database Connector	10/02/19	10/04/19	Complete	Rohan
3	Write MySQL Controller	10/05/19	11/06/19	Complete	Rohan
4	Write Auth Controller	10/05/19	10/08/19	Complete	Rohan
5	Write Mobile App Controller	10/09/19	10/22/19	Complete	Rohan
6	Write Sensor Grid Controller	10/23/19	11/06/19	Complete	Rohan
7	Testing	01/01/20	03/01/20	Not Started	Catherine
	Frontend				
8	Greenhouse Registration Flow	01/01/20	01/14/20	Not Started	Elli
9	Backend Connection	01/15/20	01/21/20	Not Started	Elli
10	Authentication Flow	01/22/20	01/28/20	Not Started	Elli
11	Backend Calls and Responses	01/29/20	02/11/20	Not Started	Elli
12	User View	02/12/20	02/18/20	Not Started	Alex
13	Historical Data View	02/19/20	02/25/20	Not Started	Elli
14	Testing	03/01/20	04/30/20	Not Started	Alex
	Sensor Grid				
15	MCU - Sensor/Pump Driver	01/01/20	01/04/20	Not Started	Matthew
16	MCU - SOC Driver	01/05/20	01/09/20	Not Started	Matthew
17	SOC - API Software	01/10/20	01/16/20	Not Started	Graham
18	Connect Components to MCU	01/17/20	01/23/20	Not Started	Graham
19	Connect SOC to MCU	01/24/20	01/27/20	Not Started	Matthew
20	Build Prototypes	01/28/20	02/26/20	Not Started	Matthew
21	Testing	02/27/20	03/28/20	Not Started	Graham
22	Peer Presentation	TBA	TBA	Not Started	All Members
23	Final Report	TBA	TBA	Not Started	All Members
24	Final Presentation	TBA	TBA	Not Started	All Members

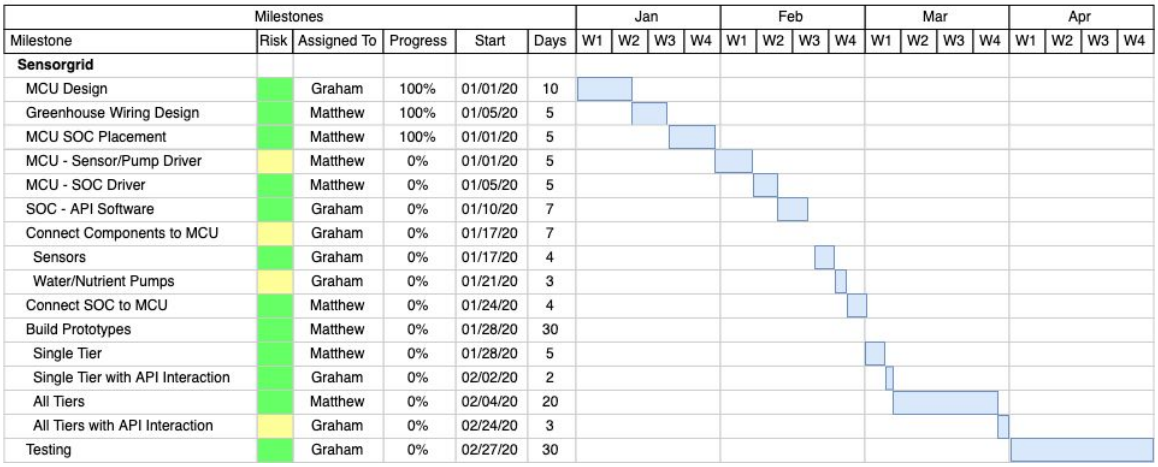
Figure 6.2.1: Gantt Chart for Each Section



Milestones						Oct				Nov				Jan				Feb			
Milestone	Risk	Assigned To	Progress	Start	Days	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
Backend																					
Create Database and Tables		Catherine	100%	10/01/19	16																
Build Database Connector		Rohan	100%	10/02/19	3																
Connection Management		Rohan	100%	10/02/19	1																
Query and Transaction Execution		Rohan	100%	10/03/19	2																
Write MySQL Controller		Rohan	100%	10/05/19	3																
Write Auth Controller		Rohan	100%	10/05/19	4																
User Creation, Login		Rohan	100%	10/05/19	2																
Password Management		Rohan	100%	10/07/19	2																
Write Mobile App Controller		Rohan	100%	10/09/19	5																
Greenhouse Interaction		Rohan	100%	10/09/19	7																
Tier Interaction		Rohan	100%	10/16/19	7																
Write Sensor Grid Controller		Rohan	100%	10/23/19	3																
Record Sensor Readings		Rohan	100%	10/23/19	7																
Water/Nutrient Adjustments		Rohan	100%	10/30/19	7																
Testing		Catherine	0%	01/01/20	60																
Write Unit Tests		Catherine	80%	01/01/20	45																
Run Integration Tests		Catherine	0%	02/15/20	15																



Milestones						Jan				Feb				Mar				Apr			
Milestone	Risk	Assigned To	Progress	Start	Days	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
Frontend																					
Mobile App Assets		Elli	50%	01/01/20	32																
Greenhouse Overview		Elli	100%	01/01/20	5																
Tier Detail View		Elli	100%	01/01/20	5																
Greenhouse Stats		Elli	100%	01/05/20	5																
Historical Data View		Elli	100%	01/10/20	4																
Greenhouse Registration Flow		Elli	0%	01/15/20	7																
Scanner		Elli	0%	01/22/20	7																
Greenhouse Networking		Elli	0%	01/29/20	14																
Animations		Elli	0%	01/29/20	7																
Backend Connection		Elli	0%	02/05/20	7																
Authentication Flow		Alex	0%	02/12/20	7																
Login		Alex	0%	02/12/20	4																
Logout		Alex	0%	02/16/20	3																
Backend Calls and Responses		Alex	0%	02/19/20	7																
Testing		Alex, Elli	0%	03/01/20	60																
Write Unit Tests		Elli	0%	03/01/20	30																
Run Integration Tests		Alex	0%	04/01/20	30																



Milestones						Jan				Feb				Mar				Apr			
Milestone	Risk	Assigned To	Progress	Start	Days	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
Sensorgrid																					
MCU Design		Graham	100%	01/01/20	10																
Greenhouse Wiring Design		Matthew	100%	01/05/20	5																
MCU SOC Placement		Matthew	100%	01/01/20	5																
MCU - Sensor/Pump Driver		Matthew	0%	01/01/20	5																
MCU - SOC Driver		Matthew	0%	01/05/20	5																
SOC - API Software		Graham	0%	01/10/20	7																
Connect Components to MCU		Graham	0%	01/17/20	7																
Sensors		Graham	0%	01/17/20	4																
Water/Nutrient Pumps		Graham	0%	01/21/20	3																
Connect SOC to MCU		Matthew	0%	01/24/20	4																
Build Prototypes		Matthew	0%	01/28/20	30																
Single Tier		Matthew	0%	01/28/20	5																
Single Tier with API Interaction		Graham	0%	02/02/20	2																
All Tiers		Matthew	0%	02/04/20	20																
All Tiers with API Interaction		Graham	0%	02/24/20	3																
Testing		Graham	0%	02/27/20	30																

6.3 - Personnel

Elli Howard

Computer Science and Biology

Elli Howard is the inventor of Pocket 'Ponics, and is the student sponsor for the project. They are also the project manager, and are leveraging leadership skills learned in 13 years of Girl Scouting to make sure that the group runs smoothly. Beyond leadership and sponsorship duties, Elli is in charge of the app logic on the front end. Their years of experience with Javascript and React have helped provide them with all the skills that they need to build the background logic for a React Native application. In addition to their coding abilities, Elli has a significant amount of experience with agriculture and hydroponics due to their biology degree.

Alexandra Cusell

Computer Science

Alexandra (Alex) Cusell is the Graphical User Interface designer for Pocket 'Ponics. Alexandra is responsible for most of the user interactions and layout on the front end of our React Native application. She has prior experience working with and designing Apple applications, making her a great asset to our team. She works closely with Elli Howard to ensure a seamless application design at all stages of development. Alexandra has also developed many of her own websites in the past, and she helps out whenever needed on that aspect of our project as well.

Alexandra brings a unique skillset to our team, as she has a strong business background. This quality helps us consider user perspectives and the needs of our customers when designing the application screens. Alexandra's business sense has also benefited our team in many other ways, such as taking long-term business decisions into consideration and encouraging us to apply for the Apple Female Entrepreneur Camp with Pocket 'Ponics. Alexandra filed incorporation paperwork for Pocket 'Ponics almost immediately after our team was created, once again showing her dedication to the long-term business health of our newly created venture.

Rohan Patel

Computer Science

Rohan Patel is the API stack developer for Pocket 'Ponics. He is responsible for making all of the API endpoints that the frontend app and the greenhouse will use. His past experience with API design, due to previous internships, gives him

the skills necessary to make the API robust and comprehensive. During his past internships, he was responsible for developing endpoints for a REST API that were used by other developers working on the frontend, giving him the experience needed to build this project.

Catherine Abbruzzese

Computer Science

Catherine Abbruzzese is the database designer for Pocket 'Ponics. She is responsible for designing all the tables and relationships, and ensuring that the database is properly optimized. She is also in charge of developing the unit tests for the API and the database. To develop the database her past experience with database systems, including her current databases class, ensures that she has the necessary knowledge to fully optimize the database system for the project. In addition, her current job developing unit tests provide her with the skills needed to write successful tests in order to ascertain 100% code coverage.

Graham Hill

Electrical Engineering

Graham Hill is the greenhouse network designer for Pocket 'Ponics. He is responsible for networking all of the greenhouse information together, and for communications between the greenhouse and the backend, as well as LAN connections between a new greenhouse and the front end app, and the electrical system to provide power to the hydroponics system. His past internship experiences provide him with the knowledge necessary to design robust and redundant network systems.

Matthew Bonsignore

Computer Engineering

Matthew Bonsignore is the sensor grid designer for Pocket 'Ponics. He is responsible for building and integrating the network of sensors, pumps, and lights that are needed for the greenhouse to grow plants. His past hobbyist experience in hydroponics, as well as classes on circuit engineering, ensure that he has the necessary knowledge to build a well-integrated sensor grid that can support hydroponic growing. Previous experience in MCUs and other SOC type of devices will prove to be useful for the work that will be done in the sensor grid. His past experience in research and prototyping his own personal projects will assist in the during the prototyping and testing phase of the system. His previous experience in construction will prove to be very important when building the Pocket 'Ponics system.

7.0 - Sensor Grid Details

The sensor grid is designed to monitor the hydroponics system of the greenhouse and control the water, nutrient, and light levels of each tier. To do this, it leverages a combination of various sensor types, as well as a main control unit and a network-enabled onboard computer. All of these electronics will be tightly integrated with the actual greenhouse, designed to work seamlessly with minimal input from the user. Furthermore, the components were all selected to minimize cost while still maintaining the reliability of the system, to ensure that the greenhouse can be produced with maximum affordability.

7.1 - Sensor Components

The sensor components of the sensor grid are a key aspect of how the entire system runs as a whole. As mentioned before, each tier of the system will consist of the same type of components. Every tier will have an electrical conductivity sensor, a pH sensor, and a pair of water level sensors. The sensor components can be broken down into two subsystems, one that is semi-automated, and one that is completely automated within the physical system. The semi-automated part of the physical system will contain the electrical conductivity sensor and the pH sensor (Figure 7.1.1), while the automated part of the system will contain the water level sensor (Figure 7.1.2) and the pump system.

In the semi-automated system, the electrical conductivity sensor and the pH sensor will take in the information and have to check with an external source, the server backend, to make sure that the measurements being read are in the desired range - not too high and not too low. If the server notifies a greenhouse that its reported levels are too low or too high, the greenhouse can use the pump system to adjust the levels so that they are within the desired range.

In the second system, full automation ensures that no external information is needed. The water level sensors will monitor the amount of water in each tier, and if levels are below or above the designated threshold value, then an adjustment will be made. These designated threshold values are determined by whether the roots of the plants will be able to easily reach the nutrient infused water, but to also make sure that the level does not become too high, such that it does not overflow. The pumps will increase the amount of water that is delivered to a tier that drops below the threshold, refilling the tier and ensuring that the plants have an adequate amount of water. These levels, both the low water level mark and the recommended water level mark values, are determined when the water level sensors are installed.

An important aspect of the sensor grid is the frequency in which the measurements are taken. If measurements are taken at too low of a frequency, then there is a chance of not being able to respond to sudden changes in the composition of the environment could negatively affect the growth and overall efficiency of the system. On the other hand, if the measurements are taken too regularly, then this would affect the overall power consumption of the system. This could make the system much less efficient overall.

Figure 7.1.1: EC and pH Sensor Layout

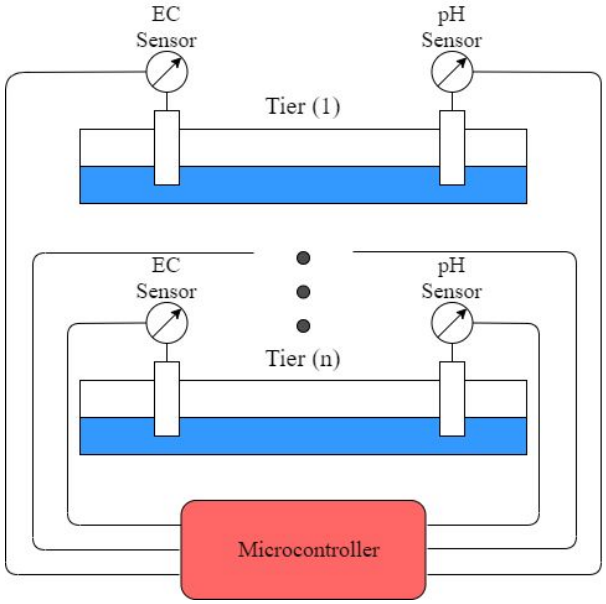
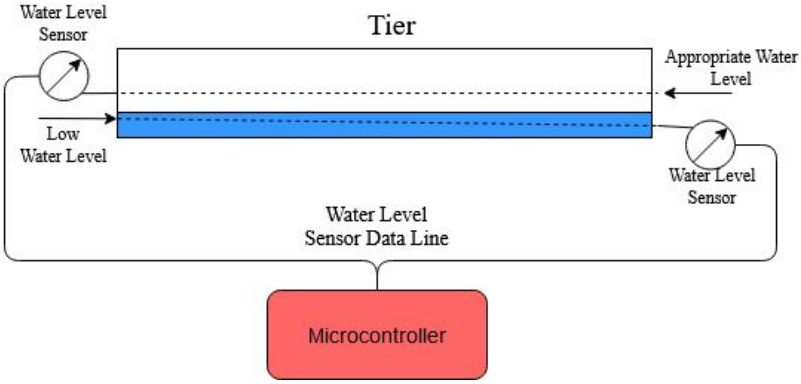


Figure 7.1.2: Water Level Sensor for a Single Tier



Electrical Conductivity Sensor

The electrical conductivity sensor is a very important sensor that is used in tandem with the pH sensor to provide an accurate representation of the nutrient level of the tier it is measuring. As mentioned before, the electrical conductivity sensor is used to take an accurate reading of the nutrient-infused water, that the

plants take in as one of their main resources. The way that the sensor measures is what is known as TDS, or Total Dissolved Solids, which is measured in ppm, or parts per million. A normal range for nutrient-rich water to be at would be around 800 parts per million to 1000 parts per million. This value can vary depending on which type of plant is using the nutrients. Some plants prefer a lower ppm value, while others prefer a higher one.

Electrical conductivity is considered an efficient way of determining how nutrient-rich a solution is. This is because, when mineral-based fertilizers are dissolved in water, the mineral will break down into a cation and an anion. The more minerals, or fertilizer that is dissolved into the solution, the higher the ionic concentration. This ionic concentration can then be measured by an electronic conductivity sensor to get an accurate measurement of how nutrient-rich a solution is.

As shown in Figure 7.1.1, each tier of the system will have its own electrical conductivity sensor that is connected to the Microcontroller Unit (MCU). The MCU sends this information to the System on Chip which will then send this information to the API and check to make sure that the measurements are in the desired range. If this range is not acceptable, the SOC will communicate back to the MCU. The MCU will then activate either the water pump, if the concentration is too high, or it will activate the nutrient pump, if the concentration is too low.

The orientation and position of the electrical conductivity sensor is also important to the functionality of the system. If the electrical conductivity sensor is oriented in such a way that the probe would be outside the water nutrient solution if the level dropped too low, this would pose an issue. If this occurred, the SOC would read an inaccurate measurement and could possibly perform the wrong action, and in turn harm the system. It is also important that the position that the electrical conductivity sensor is in does not harm the electrical components of the system. This can be prevented by applying an appropriate amount of silicone around the entry point, where the sensor will be positioned. If this area is not waterproofed, condensation could form on the electrical conductivity sensor, collect, and possibly ruin the electrical components to which it is connected.

pH Sensor

The pH sensor is also a very important sensor that, as stated above, is used in tandem with the electrical conductivity sensor to provide an accurate representation of the nutrient level of the tier that it is measuring. Similarly to the electrical conductivity sensor, the pH sensor takes a measurement of the nutrient-infused water. The amount of pH concentration of a solution is very important to the nutrient intake of plants. This is because the amount of nutrients that become available, and the amount that gets absorbed by a plant is inversely

proportional to the concentration of pH in the solution. This means that the lower the pH concentration is, the more a plant can absorb nutrients. It is recommended for a hydroponic system to have a pH concentration of around 5.8 - 6.6 pH.

As shown in Figure 7.1.1, each tier of the system will have its own pH sensor that is connected to the MCU. The MCU sends this information to the System on Chip which will then send this information to the API and check to make sure that the measurements are in the desired range. If this range is not acceptable, the SOC will communicate back to the MCU. The MCU will then activate either the water pump, if the concentration is too high, or it will activate the nutrient pump, if the concentration is too low

The orientation and position of the pH sensor is also important to the functionality of the system. If the pH sensor is oriented in such a way that the probe would be outside the water nutrient solution if the level dropped too low, this would pose an issue. If this occurred, the MCU would read an inaccurate measurement and could possibly perform the wrong action, and in turn harm the system. It is also important that the position that the pH sensor is in does not harm the electrical components of the system. This can be prevented by applying an appropriate amount of silicone around the entry point, where the sensor will be positioned. If this area is not waterproofed, condensation could form on the pH sensor, collect, and possibly ruin the electrical components that it is connected to.

Water Level Sensor

The water level sensor is also a major component in maintaining a balanced nutrient-rich hydroponic system. As stated above, each tier will have its own water level sensor pair, with also a water level sensor pair inside the reservoir tank. This is important because every plant differs in the amount of nutrients and water that it needs to thrive. The water level sensors are used to make sure that water in their designated tier does not get too low such that the plants would be starved of the nutrient-rich water that it needs. But also, that the water level of their tier does not risk the possibility of getting too high, such that it would overflow and harm the entire system.

A pair of water level sensors in the reservoir tank is also integral to the system so that the user will know when the mass storage of water will need to be replenished. The reason there is a pair of water level sensors for every tier is because the sensors are only activated when the water level reaches a specified level. This functionality is accomplished when the bobber of the water level sensor rises up to a specific height relative to the sensor. When the water level sensor reaches this height, a signal will be sent. This will be used to notify the system that the water has reached a specific level. By having two for each tier,

then the system will be able to tell when the water level is above the low threshold, and below the high threshold. This design choice was made because the purpose of the water level sensor was simple enough and would not be necessary to use a higher quality water level sensor. Because of this using a pair of cheaper, but still accurate, water level sensors is the better option in this case.

The orientation that the water level sensors will be placed in, will include one of the pair of the water level sensors at the specified height for the low water mark such that it will activate only when the water becomes too low. In order for this to occur, the sensor will need to be inverted such that the signal will activate when the “bobber” of the sensor reaches the specified height.

Inversely, the water level sensor used to monitor the higher water level mark will be placed right side up, such that the normal functionality of the sensor will occur. This means that when the “bobber” rises to the appropriate height level a signal will be sent. By orienting the sensors this way a signal will only be sent when the water level exceeds either the lower water boundary or the higher water boundary. This will make the system more efficient overall because having several sensors sending out a high signal for long periods of time can risk threatening the longevity of the system. When the water level in the water reservoir tank drops below the designated threshold, the SOC will receive this information from the MCU, and then will send this signal to the API that will then inform the user to refill the water tank.

7.2 - LED Lights

The LED lights that are also contained in each tier, will supply the other resource that the plants will need to thrive. Similar to the nutrient-infused water, all plants have a preferred amount of light that they need to thrive. Some plants require large amounts of light to efficiently photosynthesize, while some prefer less. By having a variable LED strip, the system can maintain the preferred amount of light that every plant needs.

Another useful quality that the LED lights provide, is the ability to change the composition of the light it creates. Depending on what stage of growth the plant is in, it will want a different amount of composition of red light to blue light. In the early growth stages, most plants prefer a higher concentration of blue light. While during maturity or the flowering stage of growth, most plants prefer a higher concentration of red light. The LED light accomplishes this by changing the composition of the wavelengths that it produces.

As shown in Figure 7.2.1 and Figure 7.2.2, the LED light strip encompasses the top rim of every tier. The LED light strip is positioned here to maximize light intake of the plants, and to maximize the amount of room for the plants to grow.

These are important factors for plant growth for a number of reasons. First, if the LED lighting strip was positioned lower on the tiers, the plants would grow towards the light. This would make the plants stunted and inhibit the growth. Secondly, if the LED lighting strip was too low, the plants of that tier could be at risk of being burnt by the close proximity and long duration of the light. Figure 7.2.3 shows how the LED light strips will be connected to the MCU. The SOC will receive data from the API layer for light strength and light composition. It will get these values from data stored for the plants from each specified level. The data will then be sent to the MCU, which will then send the light strength data over the light strength data line. It will also send the light composition data over the light composition data line. These data lines will then update the LED light strips to make sure that the plants get the optimal light that it needs to grow.

Figure 7.2.1: Side View of LED Light Strip

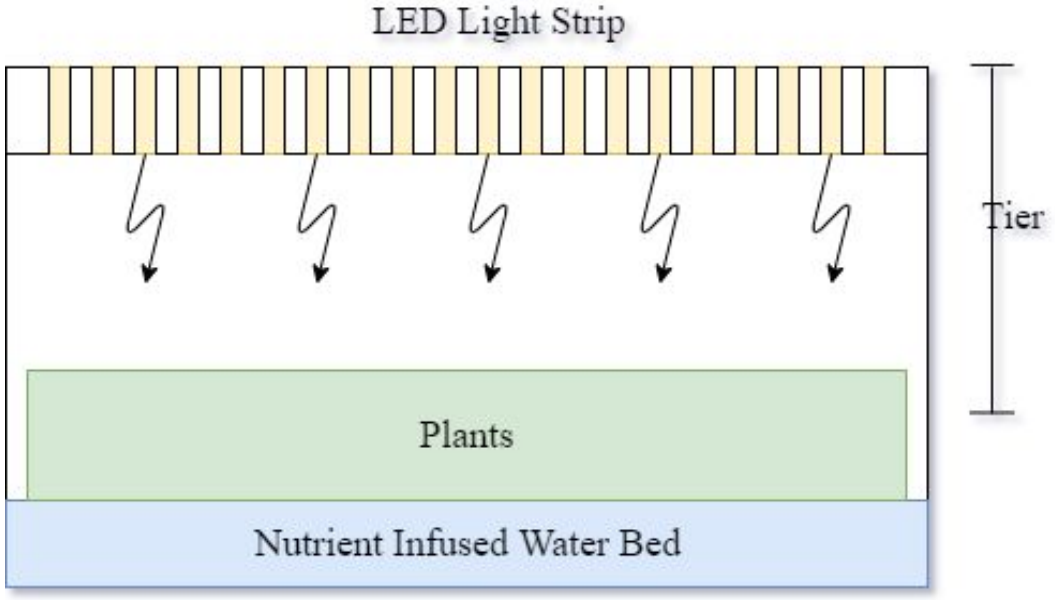


Figure 7.2.2: Top-Down View of LED Light Strips

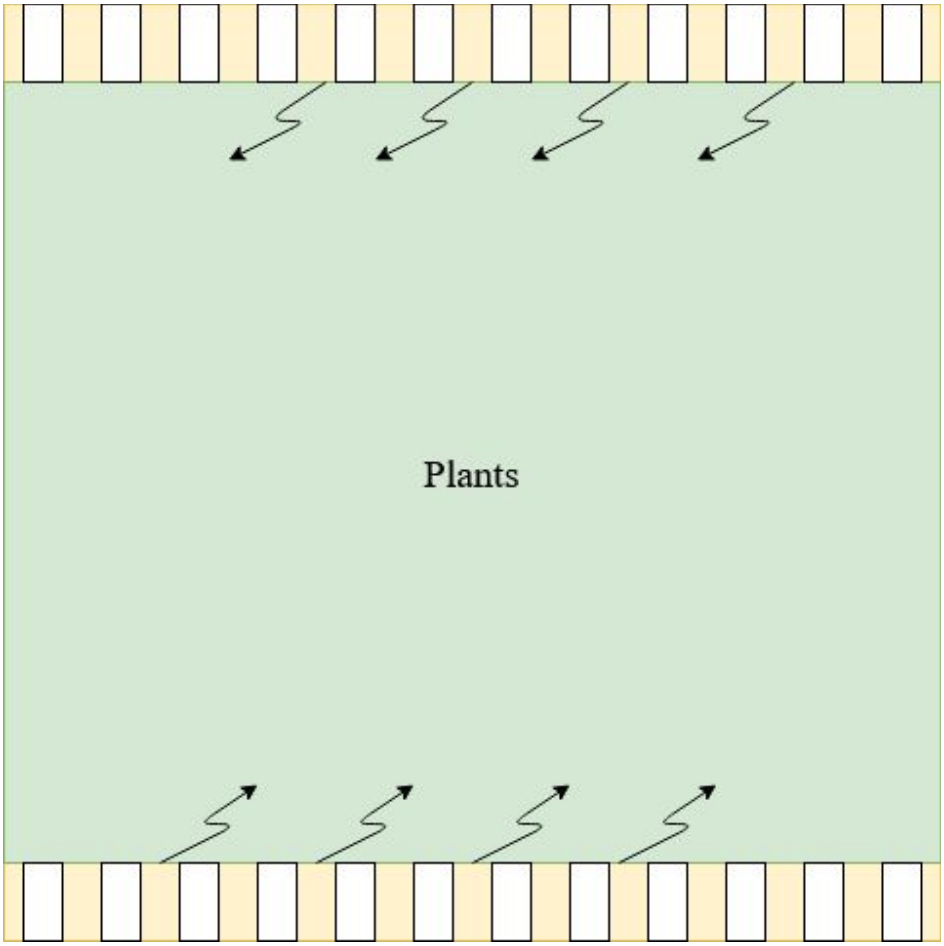
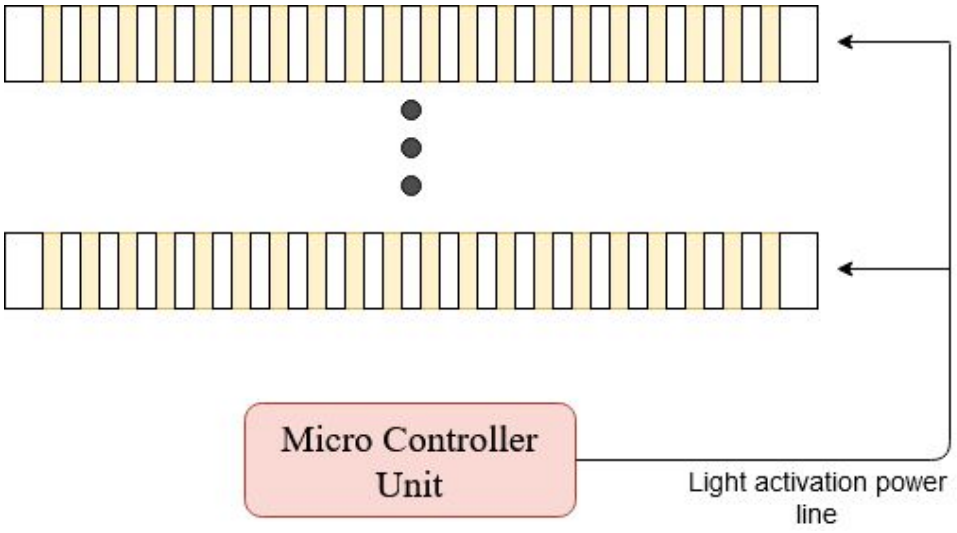


Figure 7.2.3: LED Light Data Lines to MCU



7.3 - Sensor Grid Connections

It was determined that SOC together would handle the main function of the system and would handle the readings from the MCU and administer adjustments to each of the tiers of the greenhouses through the MCU and the API. The MCU will be in charge of controlling the water management system and for intaking the sensor readings from each of the tiers. This was the case because the total amount of components would be too great for just the SOC to read from. Because of this, the SOC, along with handling the communication with the API will also handle the main code of the system for administering overall control of the system. The connections that the MCU will oversee will be all of the water pumps, and nutrient pumps, and LED lighting strips for every level of the Pocket Ponics system, along with all the inputs of the sensors.

MCU Connections

As stated above, the connections that the MCU will maintain are all the pumps and LED lights for every tier of the system. These will include all the water pumps and nutrient pumps for every tier. It was determined that the MCU would handle these components, instead of the sensors for two reasons. First, the SOC does not have enough GPIO pins to handle all of the components (sensors, pumps, and LED lights). Secondly, it was determined that the MCU will handle all of the components from the water management system and the sensor array. This will alleviate the pressure on the SOC to run the main code to oversee the system and allow the system to run more smoothly with it being easier to debug issues since all connections will be localized on one board.

SOC Connections

As stated previously the SOC will handle the connections out of the system to the API and database and will have direct connection to the MCU to govern the control of the system. To connect to the MCU we will use a Serial Connection of USB A from the SOC to a USB B on the MCU. Connections to the API and database will be over wifi connecting to a preset access point that will be administered through the app. Connections to and from the SOC will be kept to a limit due to its operational capabilities and our need for the main to be functioning at all times or suffer the risk of the system failing if the main were to be interrupted or the SOC where to be overwhelmed with input connections.

7.4 - Water Flow Management

The management of water flow is an integral part of the sensor grid layer. The movement of water and nutrients from their respective containers is key to the growth of the plants in the hydroponic system. The components that make up this layer of the sensor grid is the water tank, nutrient tank, and the pumps associated with them.

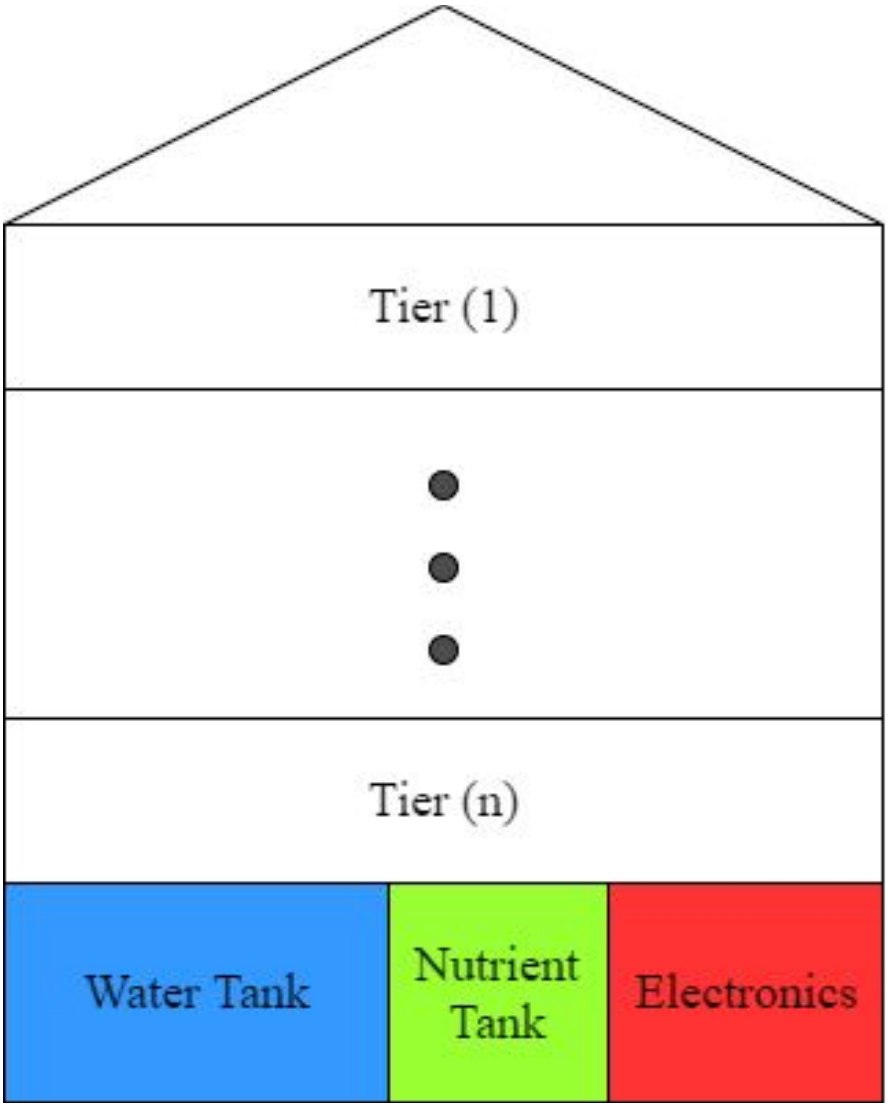
The water tank will act as a reservoir to keep a large amount of water in the system so active use. The nutrient tank will provide a similar use to the water tank, but will house the nutrient-rich liquid fertilizer that will be used to enrich the water for the plants. The pumps will be used for both the water tank and the nutrient tank, to supply whichever the specified tier requires. This means that each tier will have a pair of pumps, one that is connected to the water tank, and one that is connected to the nutrient tank.

The water management system will be controlled by the MCU on the hydroponic system. The MCU will be in communication with the SOC. This connection will be used to let the MCU know when the water and nutrient pumps need to be activated. The SOC will know when to communicate with the MCU based off of the signals sent from the electrical conductivity sensors, pH sensors, and water level sensors for each and every tier, provided by the MCU.

The combination of the readings from the electrical conductivity sensors and the pH sensor will be used to determine the activity of the nutrient pumps. While the pair of water level sensors will be used to determine the activity of the water pump. This will be what determines when the specified pumps will turn on and off. As shown in Figure 7.4.1, the water tank, nutrient tank and the majority of the electronics will be housed in the base of the system. This makes them easy to access when refills are needed.

Additionally, placing the water and nutrient tanks at the bottom of the system minimizes the amount of potential damage that could occur if the tanks happen to leak. Although we will minimize the number of failure points in the system, it is still best to plan for the worst case scenario, and leaks may occur despite our best efforts. With the tanks at the bottom of the greenhouse, leaking water and nutrients cannot affect the other tiers, and will also be less likely to impact the electronics. If they were placed above the seedlings, or above any of the tiers, leaks could flood the plants, or alter the nutrient balance of the fluid in the tiers.

Figure 7.4.1: Water Flow Management Layout



Water Tank

The water reservoir will be the largest component in the Pocket Ponics system. This is ideal because, the larger the water reservoir is, the longer the system can autonomously before a user has to fill the tank back up. The purpose of the water reservoir is to act as a mass storage of water that the system can use to maintain consistent water levels in all of the tiers of the system and provide a balance of nutrient-infused water composition. This is done by decreasing the concentration of nutrients for each tier.

The water tank will be housed at the base of the system. This is to make sure that the center of mass is not too high since the water tank will attribute to the majority of the overall mass of the system. If the tank were to be positioned too high, it would put the system at risk of falling over. A very important feature for the water tank to have is ease of access. Even though a key property of the water tank is its large size, which is meant to minimize the amount of time the user has to maintain the system.

Making sure that the tank is easy to access and maintain is very important for ease of use and maintainability. The tank will be positioned in such a way that the user will be able to easily access it to refill the reservoir to an appropriate level. As mentioned before, there will be a sensor inside the water tank that will measure the level of the water and will make sure that the level of the water is appropriate. If and when the water level becomes too low, the system will notify the user of the situation, and instruct the user how to return the water tank to recommended levels.

Nutrient Tank

The nutrient tank will provide a similar function to that of the water tank. Instead of holding large amounts of water, it will be housing high concentrations of mineral salt-based fertilizer in liquid form. It is important for the fertilizer to be in liquid form, such that the pumps will be able to transfer the nutrients up to all the levels in very accurate amounts. It is paramount that the pumps deliver accurate amounts of nutrients to the specified tiers, because small amounts of nutrients can drastically change the concentration of nutrients in the tier tanks. Similarly to the water tank, the nutrient tank will have an event that will be triggered when the levels of mineral fertilizers are running low. Again, similar to the water tank, it is important for the nutrient tank to be positioned in a convenient location for the user to easily replace and replenish the nutrient tank, such that the system can return to an optimal state.

Water Pumps

The implementation of water pumps into this hydroponics system is very important to make the system run efficiently. The use of pumps in this system makes it more efficient because it takes away the need for the user to manually maintain the water and nutrient levels. This is by far the most apparent reason for why most people don’t get into alternative farming.

This is most likely the case because there is a lot of work that goes into making sure the pH of the water is in the right range, and the electrical conductivity of the

nutrient-infused water is at the appropriate level for plant growth. If these levels are not consistently maintained, then the plants' growth will be inhibited. Implementing pumps to take the guesswork and constant checking out of the equation will fix this problem. This will give people access to a very efficient way to grow their own food, without having to worry about constantly making sure the system is meeting all of its requirements.

The system shown in Figure 7.4.2 will layout the pumps by having one pair of pumps for each tier. One that will be used to transfer water to the tier's water bed. While the other pump will be assigned to transferring the liquid nutrients from the nutrient tank to the tier's water bed. As stated above, the functionality of the water pumps will be controlled by the MCU.

The process for which the MCU will operate the water pumps and nutrient pumps is quite simple. The MCU will only activate the water pumps and nutrient pumps for small intervals of time, and wait for a response from the SOC before it makes any more adjustments. This means that when the nutrient infused water level becomes low enough to trigger the water level sensor at the low water level mark, the sensor will send a signal to the MCU stating that the water level is too low.

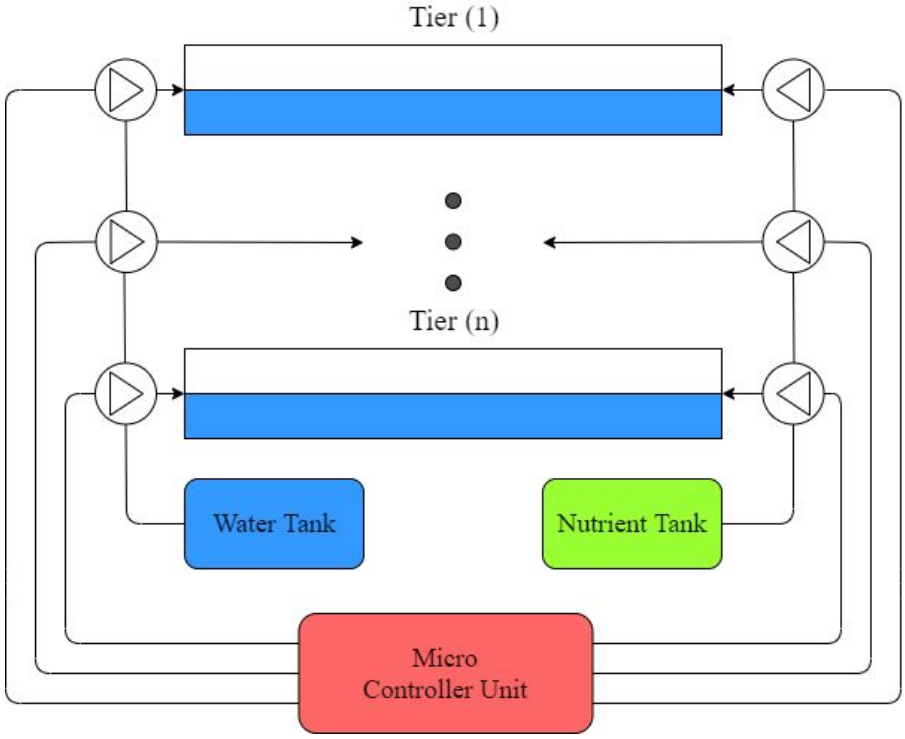
The MCUwill then notify the SOC that water level at the specified tier needs to be adjusted. At this moment, the SOC will send a function call to the MCU that will make the MCU activate the water pump at the specified level for a small time increment. The system will wait before taking another measurement to verify that the water level has not reached the high water level mark. The system will continue to do this until the high water level mark has been triggered. This signal will notify the SOC, the SOC will then notify the MCU that the water is now at the designated level.

The process for the nutrient pumps will be similar to that of the water pumps. The main differences will be that the electrical conductivity sensor and the pH sensor will be used to determine if the concentration of nutrients in the nutrient infused water is at the appropriate concentration. If it is not, then the SOC will notify the MCU that the concentration of nutrients of a specified level is too low and needs to be adjusted.

The MCU will then activate the nutrient pump for the specified level for a small discrete interval of time. The system will wait for an hour, and check the readings again. If the electrical conductivity and pH sensor read an appropriate concentration of nutrients in the water the SOC will no longer communicate to the MCU. This will verify that the system is in balance. There will be a range of acceptable nutrient concentration, such that the system will not be constantly attempting to add nutrients to the nutrient infused water, but also so that the system will not add more water to decrease the concentration of nutrients. This

range will benefit the system because it will make the system active less of the time and only add adjustments when necessary.

Figure 7.4.2: Water and Nutrient Pump Layout



Functionality

The functionality of the water management system, as mentioned before, is paramount to the overall usability of the entire system. Making sure that the system is as “hands-off” as possible, and is a key feature in our design of this project. The main events that are used to make sure the system runs appropriately is maintaining consistent nutrient-infused water levels in each tier’s nutrient bed, and the nutrient concentration of the tier’s nutrient bed. As shown in Figure 7.4.2, all of the water and nutrient pumps are connected to the MCU, and all the pumps are connected to its designated water tank or nutrient tank.

In the situation that the water level of the tier’s nutrient bed is too low, the water level sensor of that tier will signal the SOC of the situation. From the SOC, a signal will be sent to the MCU. This message will be sent over an SPI (Serial Peripheral Interface) connection. Through SPI, the SOC will send information specifying the tier, what component, and the amount that the component needs to be updated. The basis for making the SPI connection this general is to be able

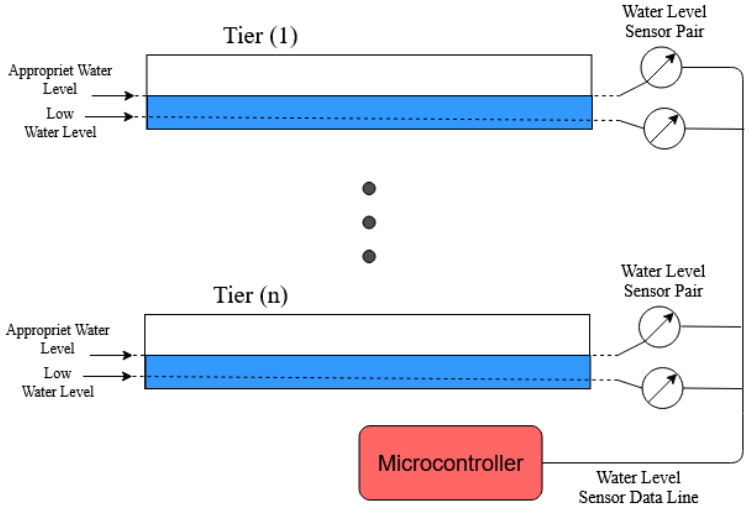
to easily make updates to any of the water pumps, nutrient pumps, or LEDs on any tier of the Pocket Ponics system.

This information form the SOC through the SPI connection is then broken down and then used as parameters for a function call that will then activate the desired component, whether it be a water pump, nutrient pump, or LED light strip. If it is a water pump, the MCU will activate the water pump of that tier to send water from the water tank at the base of the system to the designated tier. It will keep pumping until the level of the tier's nutrient-infused water bed is at an appropriate level. By adding more water to the tier's water bed, this will decrease the total concentration of nutrients in the solution.

If the SOC receives a signal that the water level is too low in a specific tier, it will activate that tier's water pump, and fill until the water level is at the appropriate amount. As shown in Figure 7.4.3, the SOC takes in a signal from the water level sensors data line from each tier when the level is at the "Low Water Level" mark. The MCU will then activate the water pump for that specified tier. The "Appropriate Water Level" is used to signal the MCU when to stop the pump such that the water level does not rise too high.

Similarly, if the nutrient level of the tier's water bed, measured by the electrical conductivity sensor and pH sensor, reaches a low enough level. These sensors will send a signal to the MCU that more nutrients need to be sent to the specified tier. This will then trigger the pump, designated for supplying nutrients to the tier, to activate and add an appropriate amount of nutrients to the tier. It will send nutrients in increments to the tier to allow it to dissolve in the solution such that it does not over-saturate the solution. As shown in Figure 7.4.3, if the SOC receives a signal that the nutrient level is too low for a specific tier, it will notify the MCU to activate the nutrient pump at that tier to supply the appropriate amount of nutrients that it needs.

Figure 7.4.3: Water Level Management



7.5 - Sensor Grid Prototype Testing

Prototype testing is an integral part of designing any type of system. For this Pocket Ponics system, testing is paramount because of the complexity and number of moving parts that are needed for this system to run efficiently. By testing the individual components, and eventually working up to the individual subsystems, a system can ensure to perform properly with the assurance that it will not have any unforeseen faults.

The testing that is done on the sensor section is of immense importance. For example, if the electrical conductivity sensor is reading a sample of nutrient enriched water and the measurements it reads are different from what was expected from the sensor's spreadsheet. This could pose a huge problem to the functionality of the system.

Since the possibilities of the components not measuring properly exist, it is important to test every sensor and component to make sure that the device performs to what is expected of it. For every test, it is important that the testing is done in a closed system, such that information that is gathered is accurate and useful.

The components that are to be tested for the Sensor Grid will include the electrical conductivity sensor, pH sensor, water level sensor, LED lights, water pumps, and the nutrient pumps. Even though the water pump and nutrient pumps are the same component, the functionality that they will perform will be different.

Therefore, they will need to be tested separately to verify that they will function properly in their respective roles. The purposes of these tests will be used to verify that all of the requirements that are set on the sensors and components are met. These tests are also used to verify that the key aspects of the system will function as expected.

Once all sensors have been tested independently, and their functionality has been verified, they will be tested in concert. Several integration tests will be run to determine the functionality of the independent tiers, as well as the system as a whole, to ensure that everything is in proper working order.

Electrical Conductivity Sensor

For the electrical conductivity sensor, testing for this device was done rather simply. In order to perform the testing in a closed system, we conducted the test in a clean tupperware container. The materials that we used to test the electrical conductivity sensor was the sensor itself, the TDS Meter V1.0 Gravity board that comes with the sensor, the SOC to get the readings from the sensor, distilled water, and the nutrient solution that will be used for the system.

The plan that was used to test the electrical conductivity sensor involved incorporating the nutrient solution with distilled water in discrete increments. A total of 1000 milliliters of water was used to make calculations easier with regard to ratios and other calculations. The electrical conductivity sensor will be connected to the Gravity board, and the Gravity board will be connected to the SOC. The connection between the Gravity board and the SOC is important because this is what will supply the power to the electrical conductivity sensor and allow for the sensor's readings to be read.

The way that the sensor will take its reading is by taking the average of multiple readings over a short period of time. This duration that was determined to be the best was ten reading over one second, or one reading for ever one hundred milliseconds. After the ten readings are stored, they are summed and divided to get an average reading. It is better to conduct readings this way to eliminate the possibility of a single false reading having a major impact on the functionality of the system.

After determining the appropriate sample rate of the device, the next aspect that needs to be tested is the effect that the nutrient solution will have on the sensor's reading. To determine the appropriate amount of nutrient concentration in the solution, the nutrients are added to the distilled water in discrete increments of 1 milliliter per reading.

After adding the nutrients to the water, the solution must be stirred until the nutrients are completely incorporated. After it is thoroughly mixed a reading can be done. This will continue until the electrical conductivity sensor reads the appropriate amount of total dissolved solids. Since the recommended amount of total dissolved solids of nutrients for hydroponic systems is around 1000ppm.

After the test solution reaches this value, the ratio between how many milliliters of nutrients are needed to reach the recommended TDS value can be

determined. This ratio will be very important to the functionality of the system, because it will be used to determine the appropriate amount of nutrients in milliliters will be needed to increase the total dissolved solids of the nutrient infused water for each individual tier of the Pocket 'Ponics system.

pH Sensor

The pH sensor will have a similar testing format to the electrical conductivity sensor. In this test, the materials that will be used are a clean tupperware container, the pH sensor, the SOC to gather the data from the sensor, distilled water, and the nutrient solution that will be used for the system.

The plan for testing the pH sensor will involve filling the tupperware container with 1000 milliliters of distilled water. It is important to use distilled water for these tests to ensure that the pH sensor is reading correctly. After the pH sensor is correctly connected to the SOC, the pH sensor will use a measuring method similar to the electrical conductivity sensor. This means that the sample rate will be about ten measurements over the duration of one second. This means that it will have a sample rate of about one reading every one hundred milliseconds.

This is a fundamental design choice, because it is important to have an average reading of a sample, rather than a single reading. If a single reading is used, it is possible that the single reading performed an error and gave an inaccurate measurement. By providing several measurements and determining an average, the risk of false readings are diminished.

After determining a sample rate for the pH sensor, a reading of just the distilled water without the addition of any nutrients is important to determine that the sensor is measuring properly. The pH of distilled water should be around a pH of 7.0, or neutral pH. The rest of the measurement that will be taken will occur after adding one milliliter of the nutrients to the distilled water.

It is also very important to thoroughly mix the nutrients into the water solution. If this step is not done, the reading from the pH sensor will not be accurate. After these steps are ensured, a reading will be taken from the pH sensor after the nutrients are thoroughly incorporated. These steps will continue until the pH readings come near the desired pH value. The recommended pH value for plants

can vary depending on the type of plant and sometimes,what stage of growth the plant is currently in. The value that will be used for this test will be around 5.5 pH.

The results from this test are important because a ratio can be determined between the amount of milliliters of nutrients used to get the solution to the appropriate pH value of 5.5, and the proportion of distilled water used. This ratio is very important to the functionality of the system because it can be used to determine how many milliliters of nutrients are needed to increase a designated amount of solution to the recommended pH level.

Water Level Sensor

There will also be several tests done of the water level sensors, to determine their full functionality. The type of water level sensors that will be used in the system are similar to ones used in aquariums. To make sure that the water level sensors will perform to the standards set for this project, two tests will be run. One test will be used to determine how well the sensors work in a closed system test individually, the second test will be done to determine how a pair of water level sensors will work in tandem with each other.

For the first test, the required materials will be a container large enough to house the sensor with room to spare, zip ties to secure it temporarily, an SOC to read in data that it is measuring, and the water that will be used to fill the container. The plan for this test will be to first secure the water pump to the container so no false readings can occur. Then, water will be added incrementally to the container. This will continue until readings from the water level sensor are recorded. This test is important, because the level where the water is at, in relation to the water level sensor, will be used when marking the appropriate and low water levels.

The second test will be conducted similar to that of the first test. The difference will be that this test will use two water level sensors. The purpose of this test will be to verify that both of the water level sensors that are in the container, positioned at the correct heights based off of the results from the first experiment, will function correctly. It is also important to understand how the water level sensors operate. Because the water level sensors are activated by a trigger, when the “bobber” part of the sensor reaches a specified height. It is important to know which orientation the sensor is in. If the water level sensor is installed inverted, the sensor will activate only when the water level lowers to a specified

height, as opposed to activating when the water level rises to a specified height. The plan for this experiment will be to mark two levels of the container that will be used as a reference for a high and low water level. The two water level sensors will then be attached to the container at the designated heights based on the results from the first experiment, such that the water level sensor for that level will be triggered when the water is at that level. Water will be added to the container and the responses will be documented when the water level sensors are triggered. The test will be a success if the sensors are triggered when the water is exactly at the level previously determined. This test will be repeated several times to determine the level of accuracy of the height levels relative to the water level sensors.

Based on the results of these tests, it will be easy to determine where to place the water level sensors at each tier and in the water reservoir tank, such that they will be triggered when the water nutrient solution is at the designated appropriate water level height, and the lower water level height.

LED Lights

There will be several tests done to determine the range of functionality that the LED lighting strips provide. There will be three different tests done on the LED lighting strips. The first test will be to determine the strength of the light from the LED's when the input voltage is varied. The second test will be to check the temperature of the LED strips as voltage is increased. The third test will be used to determine if the LED strips can be modified and mounted securely.

For the first test, a standard or unmodified strip of LEDs will be connected to twelve volt power source and also to a MCU which will be used as a potentiometer. The functionality of the potentiometer will be used to vary the voltage that the LED light strip will receive. This in turn will decrease or increase the strength of the LED light strip. The plan for this test involves increasing the potentiometer by set values, such that the strength of light can be measured in lumens, with the use of a spectrophotometer. The results from this test will be used to determine the ratio between the value of the potentiometer and the measured value of lumens from the LED light strip. This ratio is important because it will be used to determine what strength the LED light strip needs to be at depend on whether the plants at their tier prefer low, medium or high levels of light.

For the second test, temperature will be measured based on what voltage level is supplied to the LED light strip. This test will function very similarly to the previous test. Using the same plan as the first test, a potentiometer will be used to vary the voltage from a twelve volt power source. This variability in voltage will be determined by the MCU. As the voltage value changes, the temperature will be measured from the LED light strip for each interval. The results from this test will be used to determine if there is a voltage range that would be too high for the health of the plants. The worry being that, if the LED light strip were to be at full power, the plants that are in close proximity to the LEDs will be damaged or burned. If it is found that all the LED's temperatures are within an acceptable range, then no further action will be necessary, such as modifying the LED light strip so that the voltage that it receives is capped.

For the third test, it will be determined if the LED light strip can be securely mounted and modified. These are very important functions for the LED light strip. If it can not accomplish these functions, then the system will not operate properly. The plan for this test will be to cut the LED strip into varied lengths. The smallest will be only one LED and the largest will be the entire strip. Each of these strips will be attached to a constant DC voltage source of twelve volts and will be tested to make sure that every LED is functioning properly. These strips will also be mounted to the same surface material that the LED light strips will eventually be mounted to. They will then be tested on how well the strips adhere to the surface, and verify that varied length of strips do not inhibit the adhesion strength of the strip. The purpose of this test is to verify that even if the LED strips are modified to any length, the LEDs will still function correctly, and adhere properly.

Water Pumps

There will be several tests done on the water pumps, to ensure that they function properly. There will be two different tests done on the water pumps. The first test will be done to determine what the flow rate of the water pump will be at a specified voltage. The second test will be done to determine how much the voltage will need to be modified to give the same flow rate depending on how high the fluid needs to go. The materials needed for the first test will be a scale model of a tier's nutrient water bed, a water pump, a MCU used to vary the voltage that the pump will receive, a water tank reservoir and a 12 volt power source.

The plan for the first test will be to connect the water pump the MCU and the 12 volt power source. The output end of the pump will lead to the water bed. The water bed for this test will be positioned at the same height as the water pump. The water pump will then have it's receiving tube put into the water tank. With the pump receiving a constant voltage of 12 volts, the water pump will remain on for one hundred seconds. After this time period the water pump will be turned off, and the volume of water that was pumped into the bed will then be measured. This ratio will be used to determine the max flow rate of the water pump in milliliters per second. This value will be very useful to determine how long the water pumps will need to remain active to dispense an accurate volume of liquid.

For the second test the materials that are needed will be the same as the first test, a scale model of a tier's nutrient water bed, a water pump, a MCU used to vary the voltage that the pump will receive, a water tank reservoir and a 12 volt power source. The plan for this test will be to perform the same test as done before, but to repeat it with the water bed at varying levels of height. The difference in height level for each test will be ten centimeters, from ground level to one meter. At the end of each test the total volume of water will be measured and compared. From the results of the test a ratio can be used to determine how much the flow rate is decreased as height increases. This ratio will be very important for the system because it will be used to determine the total amount of time necessary to fill up a specific quantity of water.

Nutrient Pump

The testing that will be done on the nutrient pumps will be similar to the water pumps. There will be two different tests done on the nutrient pumps. The first test will be done to determine what the flow rate of the nutrient pump will be at a specified voltage. The second test will be done to determine how much the voltage will need to be modified to give the same flow rate depending on how high the fluid needs to go.

The materials needed for the first test will be a scale model of a tier's nutrient water bed, a nutrient pump, a MCU used to vary the voltage that the pump will receive, a nutrient tank reservoir and a 4.5 volt power source. The plan for the first test will be to connect the nutrient pump to the MCU and the 4.5 volt power source. The output end of the pump will lead to the nutrient water bed. The

nutrient water bed for this test will be positioned at the same height as the nutrient pump. The nutrient pump will then have it's receiving tube put into the nutrient tank. With the pump receiving a constant voltage of 4.5 volts, the nutrient pump will remain on for one hundred seconds. After this time period the nutrient pump will be turned off, and the volume of nutrients that was pumped into the bed will then be measured. This ratio will be used to determine the max flow rate of the nutrient pump in milliliters per second. This value will be very useful to determine how long the nutrient pumps will need to remain active to dispense an accurate volume of liquid.

For the second test, the materials that are needed will be the same as the first test, a scale model of a tier's nutrient water bed, a nutrient pump, a MCU used to vary the voltage that the pump will receive, a nutrient tank reservoir and a 4.5 volt power source. The plan for this test will be to perform the same test as done before, but to repeat it with the nutrient water bed at varying levels of height. The difference in height level for each test will be ten centimeters, from ground level to one meter. At the end of each test the total volume of nutrients will be measured and compared. From the results of the test a ratio can be used to determine how much the flow rate is decreased as height increases. This ratio will be very important for the system because it will be used to determine the total amount of time necessary to fill up specific quantity of nutrients.

This difference between the water pump and the nutrient pumps flow rate will be paramount to the implementation of the semi autonomous functionality of the system. This means that the amount of fluid that the water pumps will push through the system will be very different to the amount of fluid the nutrient pumps will push through the system at the same voltage and duration. This difference is caused solely from the fact that the nutrient fluid is much more dense than water. Because of this, the nutrient pump will have a much different flow rate, than that of the water pump. With this difference in mind, another very important factor to keep in mind is that the amount of fluid that the nutrient pump will put into the system will change the concentration of the nutrient infused water much more drastically than if the water pump were to put in the same quantity of water into the system.

7.6 - System On Chip (SOC)

The system on chip or SOC (Figure 7.6.1) monitors and controls the entirety of the greenhouse system on the hardware side of the configuration. It is the direct connection between the hardware and the software side of the platform. Designed to maintain control of the hardware while providing information to the backend system for updates, the provided information then helps to create adjustments on the fly for all of the tiers of the hydroponics system.

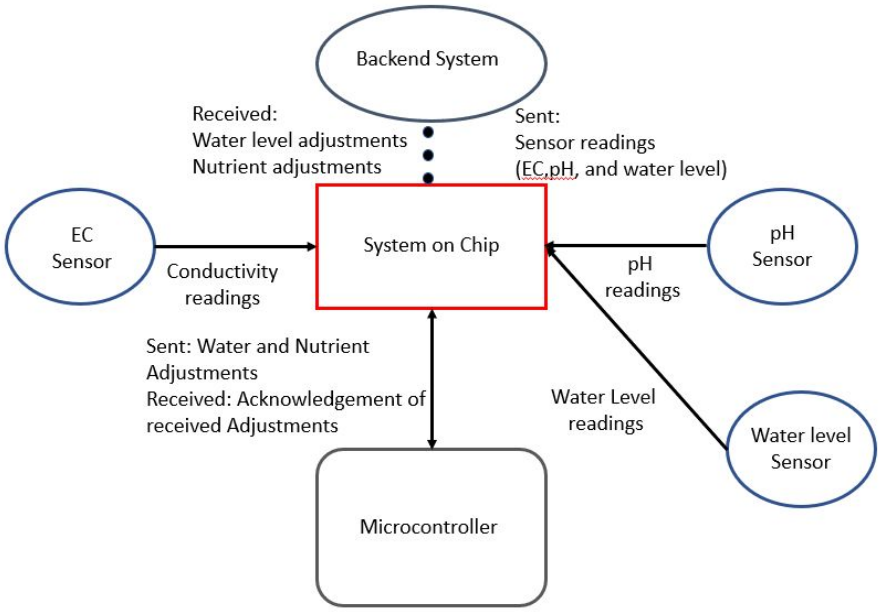
All communication in the greenhouse is routed and connected directly to the SOC, this is intended to keep the overall design of the system simple and keep the connections from the sensors to the backend clean and maintained.

The SOC is also the system in place to provide network connection to the system. The hydroponics system is supposed to be “live” at all times to send data and receive adjustments on the fly. Using a pre-fabbed system like a raspberry pi that is already tested and guaranteed to adhere to current standards for network connections was a simple solution to a complicated issue.

The system has to be connected to the backend at all times as stated above in the internet connection section. The SOC has to ”ping” or connect to the backend and send a reading from a sensor. Currently a python script is being implemented to handle and send the values from the sensors to the backend. The script is being implemented to be live at all times as the sensors will be polled every hour or so to provide their readings for connection.

The SOC and MCU systems need to be linked to provide updates and readings to the backend and to adjust the LED and pumping systems. Currently the two different systems are being linked through Serial Connection since the connection method is simple and easy to implement. Several communication functions between the two devices are used to ensure that information is sent and received efficiently.

Figure 7.6.1: SOC Coordination Diagram



Testing the SOC

As stated above the SOC is one of the more integral parts of the system so testing that it is operating properly is of utmost importance. The testing for the SOC will take place in multiple parts testing each section that of which the SOC needs to perform. These tests will be comprised of firstly the connections of the sensors to the SOC and ensuring that the signals coming from the sensors are being interpreted correctly in the SOC. The second test will be seeing if the SOC will connect and transfer data to the backend system. The third and last test of the series will be testing if the MCU and the SOC can communicate properly and acknowledge the changes sent to the MCU.

The first test is to ensure that the sensor are sending data to the SOC properly and that the SOC can interpret the signals coming from the sensors. This test is also split up into three different sections for each sensor having to test that each sensor can send its readings to the SOC respectively.

The first of these tests is for the electrical conductivity sensor that will go hand in hand with the testing of the sensor itself, while testing the sensitivity of the sensor with distilled water and nutrient solution we will be transferring sensor readings to the SOC. The required materials for this test is the SOC system, the The SOC needs to receive readings from the sensor over a small period of time 10+ readings within a few seconds time span. This test will not only ensure that the SOC can read the values from sensor but will also make sure that the SOC can

receive values from the sensor very quickly within such a short time span. If the SOC can receive the readings then its partially a success, it is fully a success if the SOC can handle the constant stream of readings coming from the sensor measuring the EC of the distilled water.

There is a second stage to this testing as well, that is focused on if the SOC can manipulate the readings from the sensors as defined in the EC sensors testing section. The code on the sensor needs to take in the multiple readings (10) over the allotted time period (one second) and average them. This test will help to determine what the proper sampling rate is needed for the EC sensor to have stable readings and have them be accurate at the same time. Accuracy is highly important for this since its the primary sensor to determine the amount of nutrients in the water. If the SOC can one take in the sensor readings and two manipulate the readings properly then the testing for SOC and this sensor are considered passing.

The second test is for the pH sensor this also will go along with the initial testing of the pH sensors. The required materials for this testing scenario is the SOC, pH sensor, a clean tupperware, distilled water, and the nutrient compound for the plants. The sensor will pick up 10 readings from the water and nutrient solution, if the SOC can properly receive the multitude of measurements from the sensor within the short time period then the results will be compared to a litmus test of the nutrient solution to ensure that the readings themselves are accurate and weren't corrupted when being sent to the SOC. If the sensor readings are available in the SOC then the test is marked as an initial success for just being able to pull the readings from the sensor.

The second section of this test is similar to the EC sensor test where the SOC needs to be able to manipulate the readings from the sensor as defined in the pH sensor testing section. The code for the pH sensor readings needs to take in the multiple readings (10) over the allotted time (one second) and average them to eliminate and chance of outliers and determine the proper polling rate of the sensor for the SOC. If the SOC can one take in the sensor readings and two manipulate the readings properly then the testing for SOC and this sensor are considered passing.

The final testing for the sensors is for testing the water level sensors. Maintaining a proper water level is essential for the entirety of the greenhouse system. The testing for this once again goes hand in hand with the testing as defined in the water level sensor testing. The required materials for this testing is two water level sensors, the SOC, distilled water, and the water container for the system.

This testing series is broken into three series, the first test is to see if the SOC can read values from a single water level sensor. This is similar to the previous sensor testing where only one sensor value is needed. This is just to ensure that the SOC can read the sensor itself and is the starting point for evaluation.

The second testing section is to test with two sensors in tandem, testing for a low water level and an appropriate water level for the system. The SOC needs to be able to take inputs from both of the sensors at the same time and the readings from both of them are needed to correctly assess if the water levels are within appropriate ranges. If both sensors can be read at the same time independently of each other then the measurement section of the testing is determined a success.

The last part of the testing for this section of the system is to see if the values of the sensors can be manipulated by the onboard code of the SOC. Both readings from the sensors are needed to determine the proper water level. If the water level falls below the low water sensor the the SOC will see that the low water level sensor has become untripped and will send a signal to the MCU to correct this with the pumping system. Once the appropriate water level sensor trips then the control signal from the SOC to the MCU will cease and the water level in the system will be marked as high enough. If both sensors read as marked then no action will be required, and if the low level is tripped and the appropriate level isn't then the SOC will make no corrections until the low level is tripped as well. If all variations of combinations for the sensor work as expected then the testing for the water level sensors is marked as a success and considered passing.

The second testing section for the SOC is to test if the SOC can connect to the backend system properly and also transmit the appropriate readings of the pH and EC sensors. After the sensor values are read then the proper adjustments are required to be made from the backend that has the averaged values of the multiple readings from the sensor. The SOC will send a pull request for the backend system, the SOC will then send the most recently polled sensor value to the backend system. The backend will respond to the SOC if an adjustment is needed to correct for a high or low reading.

To test this and make sure that this connection is working properly reading(s) will be fed into the SOC from one of the sensors. From these readings the SOC should average them to find the corrected value, the SOC should then generate a push request for the backend and forward the value to this system. If the value is outside of range then it will receive a correction/adjustment, if not then just an acknowledgement will be returned that the data was received.

This will be tested for all available sensors and for all available options of receiving a correction or not correcting. All possibilities need to perform as expected and as per the design of backend and SOC systems. If any combination of data transfer fails then that sensors communication with the backend is marked as a failure and the coding for that section will need to be addressed, edited and retested until it can be marked as a pass. All sections must be marked as a pass before the system can be fully constructed and implemented.

The final testing section for the SOC is to ensure a stable connection between the SOC and the MCU. The MCU controls the pumping and LED systems of the greenhouse and is dependent upon the SOC to make its corrections to the greenhouse. Because of this dependency a stable and clean connection is required for the MCU to operate as it is designed. The materials required for this testing will be the SOC and the MCUs and any connections between the two that are needed.

Firstly, we need to make sure that the SOC and the MCU are able to talk back and forth, a simpler code that implements either the SPI or UART (Universal Asynchronous Receiver/Transmitter) based connections will be used. Using a code on the MCU that populates a value equal to the string of “Hello World” is loaded and ran on the MCU, on the SOC side a code is initialized to read across the chosen connection type to read a certain bit value within the MCU. If this read value is equal to the string “Hello World” as also initialized in the SOC then a signal is sent to light an LED on the MCU. If the MCU lights when the code is run on the SOC then the communication testing is marked as successful and passed.

The second stage is to test if the SOC can send a control signal to the MCU and have it send an output from one of its pins, much like how the system will operate with the pump and LEDs. We will have the SOC send a control signal through to the MCU and read the selected PIN as defined in the testing code, if the PIN changes its output value to the predetermined value then we can determine that the MCU will properly accept the control signal from the SOC and will be marked as passed.

The last and full test for the system will go hand in hand with the full testing of the LEDs and pumping system which will involve for the LEDs adjusting the resistance for the LED strips to change the voltage going to the LEDs. So the SOC needs to send the signal to the MCU to adjust the resistance load of the LEDs, if the brightness and/or color of the LEDs shifts to the expected values then the first half of this testing is passed.

The second half for the pumping system will be to send a control signal to the MCU from the SOC to apply a voltage or send a signal to turn the pumps “on” for a period of time and then shut off. This needs to be done multiple times within a short time period to simulate as if it was running in the actual hydroponics system. If the MCU receives the signal to activate the pumps and can continue to activate them multiple times in the short time period then this will be marked as passed.

Both sections of the final stage of testing needs to be passed before full implementation of the system. If either of them do not pass then the boards, code, and sub-systems themselves will need to be assessed and debugged before retesting.

7.7 - Microcontroller Unit (MCU)

The Microcontroller Unit (MCU) is the system that runs in parallel with the SOC system, meaning that it is designed to handle the load of the pumps and the LED strips while only needing input from the SOC to affect how the MCU will interact with the pumps and the LEDs. This design is supposed to keep the whole of the system simple in design with the main logic components being handled by the SOC and adjustments and more “grunt work” being handled by the MCU.

The current design from the ECE team as listed above is to have all of the sensor readings feeding into the SOC which will be forwarded into the backend system for adjustments, these adjustments are then forwarded back to the SOC and trickled back down to the MCU system. Depending on the component being adjusted the MCU will send control signal to either the LED or pumping systems. For the LED systems the MCU will can either send a signal to a relay to activate or turn off the LED as a whole or it can send a control signal to a digital potentiometer that will adjust the overall brightness of the LED arrays. If it is adjusting the pump system the MCU will send a control signal to a relay dependent upon the tier that will keep the pump active for a predetermined period of time before turning back off again.

The MCU system that we are currently going to be implementing our systems on is the Arduino Mega 2560. This is a large MCU with a lot of output pins and a stronger onboard processor that will be able to handle adjustments to the LEDs and pumps more easily than the previous decision of the MSP430 would be able to. The supported language for the MCU is also an arduino based programming language that is a hybrid of the languages C and python. This will allow for easier communication between the SOC and the MCU due to the fact that the SOC will be running python based scripting for its main, and most of our experience with programming is in the use of the python and C languages. We are currently looking at using either a UART or SPI connection between the SOC and the MCU systems. We are currently testing both to determine which will allow for the best communication between the system and will allow for faster responses between the systems as well.

MCU Prototype Testing

The MCU is the physical part of the system that takes in requests/control signals from the SOC to make the required adjustments to the hydroponics system. There will be three sections of testing for the MCU needed to ensure that it is operating per design including testing each individual sensors connection and operation with the MCU and testing the MCUs connection with the SOC system.

The first test is for testing the MCUs control over the LED strips. The MCU LED control schematic is shown in Figure 7.7.1. The materials required for this test is the MCU and the LED strips. For this test the MCU is being treated as a potentiometer that is serving as a load for the LED strips to adjust the voltage that the LED strips are receiving. The MCU will slowly adjust the load seen by the LEDs to alter the voltage the LEDs is receiving. If the MCU properly adjusts the resistance to the correct values as specified in the LED testing plan then this section can be marked as pass.

The second test for the MCU is to test control over the pumping system. The MCU pump control schematic is shown in Figure 7.7.2. This section is directly related to the testing of the pumps as defined in the water pump testing section defined above. For this test the required materials are the MCE and pumps. To test if this system is operating properly a control signal will be sent from the MCU to the pump control to activate the pumps for a set amount of time. If the pumps activate properly, and can continuously operate the pumps repeatedly with no issue then the test can be marked as a pass.

Finally, we need to make sure that the SOC and the MCU are able to talk back and forth, a simple code that implements either the SPI or UART based connections will be used. Using a code on the MCU that populates a value equal to the string of “Hello World” is loaded and ran, on the SOC side a code is initialized to read across the chosen connection type to read a certain bit value within the MCU. If this read value is equal to the string “Hello World” as also initialized in the SOC then a signal is sent to light an LED on the MCU. If the MCU lights when the code is run on the SOC then the communication testing is marked as successful and passed.

The second stage is to test if the MCU can receive a control signal from the SOC and have the MCU adjusts the output of one of its pins, much like how the system will operate with the pump and LEDs. We will have the SOC send a control signal through to the MCU and read the selected PIN as defined in the testing code, if the PIN changes its output value to the predetermined value then we can say that the MCU will properly accept the control signal from the SOC, this will be tested with all pins being used for testing, if all pins can easily be adjusted to a preset value then it will be marked as passed.

The last and full test for the system will go hand in hand with the full testing of the LEDs and pumping system which will involve for the LEDs adjusting the resistance for the LED strips to change the voltage going to the LEDs. The MCU will receive a control signal from the SOC to adjust the resistance load of the LEDs, if the brightness and/or color of the LEDs shifts to the expected values then the first half of this testing is passed.

The second half for the pumping system will for the MCU to receive a control signal from the SOC to apply a voltage or send a signal to turn the pumps “on”

for a calculated period of time then cease. This needs to be done multiple times within a short time period to simulate as if it was running in the actual hydroponics system. If the MCU receives the signal to activate the pumps and can continue to activate them multiple times in the short time period then this will be marked as passed.

Both sections of the final stage of testing needs to be passed before full implementation of the system. If either of them do not pass then the boards, code, and sub-systems themselves will need to be assessed and debugged before retesting.

Figure 7.7.1: MCU LED Control Schematic

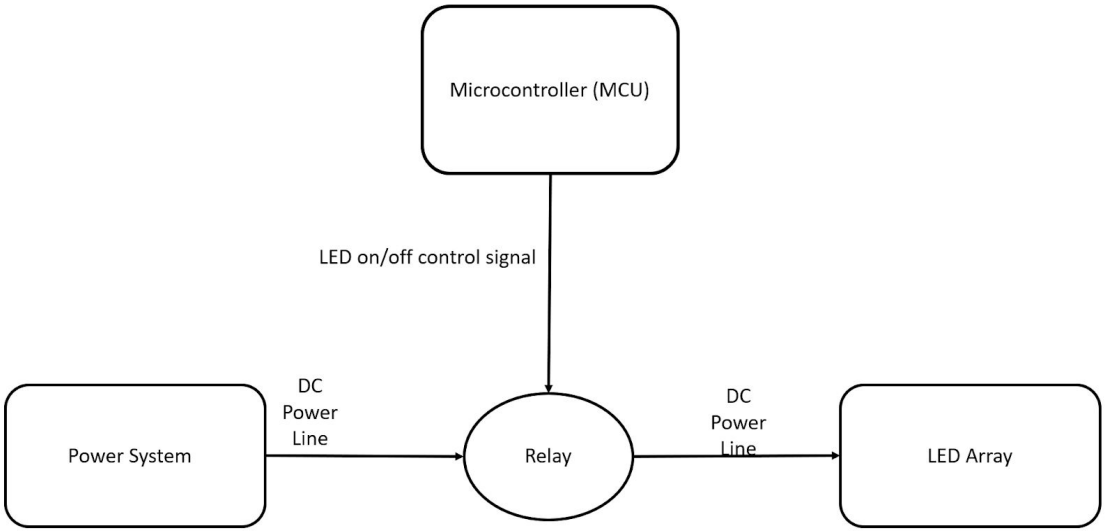
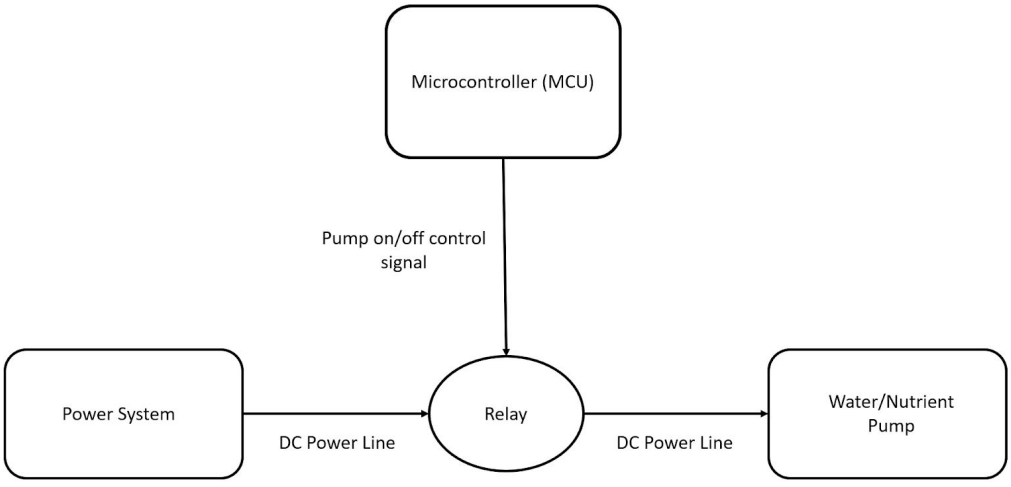


Figure 7.7.2: MCU Pump Control Schematic



7.8 - Power System

This hydroponics system is quite large with a lot of electrical components on each tier, and because of this a simple battery array will not provide enough power to keep the system functioning as intended. A simple prebuilt power supply is being implemented to use the standard power outlets available in the United states, this power source is 120V AC at 60Hz. The systems on our platform can not implement AC sources into the main components (most computer systems or MCUs require a stable DC supply). Because of this the power system requires a way to rectify and convert the AC signal into DC.

The current implementation of the power system is to take in the larger AC signal that will then be rectified down to three different 12V signals

We decided to move forward with a prebuilt and purchased power supply system that would alleviate the risk of us building and fabbing our own power supply system. A prebuilt system allows us to tweak the output of the terminals to the exact amount that we would like for the system, allowing for quick and easy connections that are simple to administer for power debugging. The prebuilt system will have three live terminals and three common ground terminals that allow us to distribute the load across the three different terminals, so all the power connections are shoved into one terminal.

To power the SOC we will be using a step down convertor that will step the 12 volt signal down to 5v and 3 amps for powering the raspberry pi. Since this system can't directly take a 12V signal it will have to be stepped down to prevent damage to one of our most critical systems. This power convertor will be directly connected to one of the three terminals of the power supply and will be plug and play from there for the SOC. To power the MCU we will have a direct 12V connection from the power supply to the MCU using a pigtail connection.

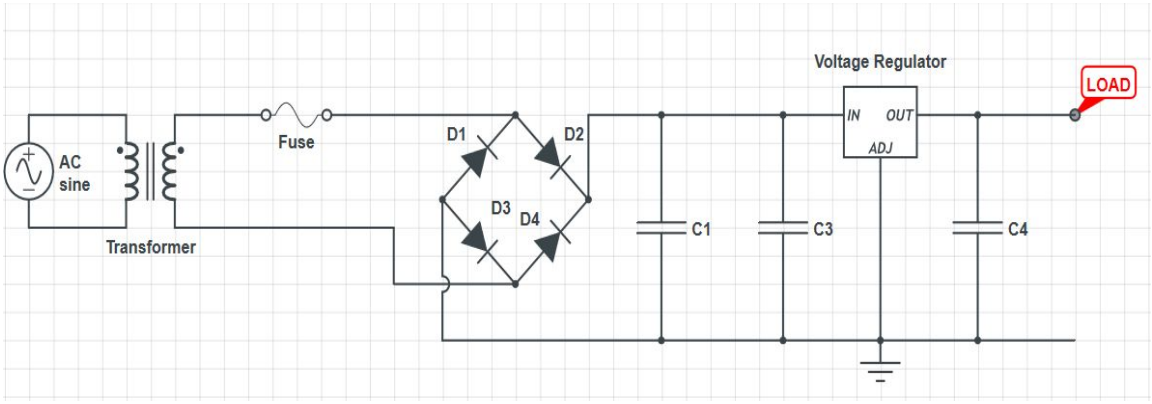
For all connections requiring a 5V connection the 12 volt signal will be connected to a step down buck convertor. The bucks have all been adjusted to output a signal of 5 volts down from the 12 volts. Each buck has been calibrated from being a throughput to the exact output voltage that we need. Each buck has two different voltage output terminals so the current plan for laying lines will be to have two sensors connecting to one output of the buck and the other terminal to have two water level sensors. Because of this we will be able to support a tier per each buck without having to strain the converters, this is mostly a safety precaution to reduce the risk of fire or damage to other components. With 4 bucks we should be able to power the entirety of the system.

To simplify the wiring of the power system we will be using a larger gauge wire of 14 AWG and use smaller 18 and 24~26 AWG wires splitting off from this larger bus to deliver power to all the components. For the lines to the Pumps and the LEDs we will be using a 14 AWG wire as the main bus to carry the full load and

will use 18 AWG that will connect into the relays that are used for the control of those systems. For the smaller components we will be using a 18 AWG wire that will connect to a bundle of 24~26 AWG wires that will split and connect to the bucks and from there to the power relays to power them.

To keep the ground reference clean to help distinguish clarity for the digital highs and low we will be using a system of ground busses, these will function similarly to the live busses previously mentioned but instead of carrying the live it will be connected to one of the ground terminals of the system. Once again most of these busses will be composed of 14AWG wire that will split off into 18AWG or 24~26AWG wires that will ground the individual components.

Figure 7.8.1: Power System Schematic



Power System Prototype Testing

Most of the testing will be done incrementally and independent of each other ensuring that the used parts are going to operate as specified, i.e. testing the transformer, testing the rectifier, testing the regulator. Once individual parts of the system are tested then they will be wired together part by part and test if they are still operating as expected.

Testing the power system will be broken into these sections: component based testing, full layer design testing, and lastly testing design with loads (i.e. pumps, LEDS, SOC and MCU, and all attached). All tests will require either a signal generator or DC source, as well as an oscilloscope or digital multimeter, all power system components, and any extra load resistors that are needed. Testing to monitor thermals of the design will be run as well, this test will require the signal generator and resistors to act as loads for the power supply.

The first component tested will be the AC transformer, the transformer will be attached to a signal generator that will be creating a signal of 120V ac at 60Hz to simulate as closely as possible an AC wall socket that would be commonly found

in the US. An oscilloscope will be attached to the output ends of the transformer to check if the voltage has been stepped down. The used transformer should step the voltage down to 12V ac.

The next component to be tested is the bridge rectifier, a signal generator will be hooked up to the bridge rectifier at 12V ac and an oscilloscope will be attached to the output. The expected output for the rectifier is to generate a mostly smoothed DC signal with a small amount of wiggle that is accounted for with smoothing capacitors. If the output is mostly smoothed and maintained at around 12V dc then a smoothing capacitor will be attached to the output and the oscilloscope will be moved to get then new output after the capacitor. The system will be retested to see if the output signal becomes smoother and more stable if that is the case then the rectifier and smoothing capacitor are marked as passed, if not then other components will sub them out and be retested to with the same process.

The last component to be tested is the voltage regulator, this component is to ensure that the output voltage stays stable to the load(s). The regulator will be hooked up to a 12V DC source and the load will be varied to ensure that the regulator can maintain a set voltage level.

The second stage to testing the power system will be to test the entirety of the design for the base power system. First steps for this testing will be to wire the AC transformer and the bridge rectifier with the smoothing capacitor together. The input to the system will be the same signal generator as the testing for the transformer initially at 120V ac and 60Hz. The output of the rectifier will be attached to an oscilloscope this will measure if there is still an AC component of the signal that needs to be corrected. If the output is around the 12V dc as expected then the voltage regulator can be attached to the system and the oscilloscope moved to the new output.

The last section of testing for the power system is to start attaching loads in the form of the different subsystems. Depending on the subsystem being tested a few different extra components like a 12v to 5v transformer can be employed along with resistors to manage current flow.

Once all components pass initial testing then other components will be introduced with each other to test the load that the power supply can handle. If the current starts to spike in any of the branches for the different loads then fuses will blow that have been added in to protect against over amperage. If all loads can be added and operate as needed and expected then this testing will be marked as passing and the system on an electrical end can operate properly.

Thermal testing will be implemented for the power supply itself to test if the heat producing components will produce heat over the acceptable operating temperatures for the system. For this test the rectifier will be attached to a breadboard and a 12V AC signal generator. The rectifier will be attached to a

resistor based load to simulate being connected to the essential components of the network and sensor array.

The test will be conducted with different resistor loads to simulate the LEDs or pumps being disconnected or running at a high load. Each different load will be run for at least 20 to 30 minutes to ensure that the rectifier has had enough time to be considered in a stable operating state. Periodically after the 5 minute mark temperature readings will be taken from the rectifier. The rectifier should continue to heat and become stable at a set temperature.

Once this is complete the test can be rerun with a similar rectifier that has had the makeshift heat sink attached to the rectifier. If the heatsink reduces the temperature of the rectifier then a heatsink will be properly attached and a more formal mounting system will be implemented for the heatsink.

The second section of testing will be to perform the same test, with or without the heatsink but now enclosing the rectifier in a small container. This will simulate the rectifier being enclosed by the case that will protect the power supply. The same test will be run as before at a run time of about 30 minutes, the only change is the temperature readings will be more frequent at 2.5 minute intervals. This is to ensure that the rectifier runs no risk of overheating in the enclosure which could cause a fire or for the component to fail. If this test passes as well then the rectifier can be marked as thermally safe and can be implemented fully into the system.

7.9 - Sensor Layer Software Design

The software design of the sensor layer incorporates several aspects of the Pocket Ponics system. The Sensor Layer will include all of the electrical components that require software that reside in the Pocket Ponics system. These components will include the MCU, and the SOC.

The MCU will be in charge of maintaining the data collection from all of the sensors, water flow management system, and the LED lighting system. This will include controlling when the water or nutrient pumps for each level will turn on. The MCU will also be in charge of turning on and off the LED lighting strips, and make adjustments to the LED light strips for every level.

The SOC will be in charge of several aspects of the system. These will include making sure to, every hour, check over the whole greenhouse through the MCU. The SOC will also be in charge of the communication with the API . This means that the SOC will be able to send and receive information to and from the API, depending on the data that the API will send to the SOC.

MCU Software Design

For the MCU software, as stated above, will be in charge of the components that are part of, sensor system, the water management system, and the lighting system. The components in the sensor system are the pH , electrical conductivity, and water level sensors. The components in the water management system that the MCU will operate are the water and nutrient pumps. The way that the software will work on the MCU will be relatively simple. It was designed this way such that it could easily be modified and understood. The MCU will not perform any actions unless notified by the SOC through two main communication functions.

The first of the two communication functions between the SOC and the MCU will be in the form of a simple function call callMCU(). The three parameters of the callMCU() function will be read in from the SOC and stored in local variables on the MCU. The first parameter will be an integer labeled “tier” which will refer to the tier that needs to be accessed. The second parameter will be a string labeled “component”, which will refer to the type of component that needs to be accessed. The third and final parameter will be an integer labeled “amount”, which will refer to the amount that the specified component at the specified tier needs to be adjusted. Depending on the component, the amount value will refer to different units. The purpose of using a universal function call from the SOC makes it easier to incorporate all of the functionality the MCU will provide. This is because all of the components that the MCU will be in charge of will receive the same types of inputs, and so making a universal function call that incompasses

all the functionality of the MCU makes the software design much more simple, and easy to understand.

The second communication function between the MCU and the SOC is the getMCUData() function. This function takes in one parameter called tierNum in the form of an integer. This function will use the tier number to find the values of all of the sensor readings for the requested tier. The function will then return all of the values of the readings from the sensors for that tier, as well as the water level value of the water and nutrient reservoir tanks.

Water Pumps Function

In the case that the component that the second parameter is referring to is the water pump, the MCU will begin to operate the water pump. The MCU will do this by using a nested switch case statement. Depending on the composition of the parameters of the function, the MCU will decide whether or not to turn on the water pump.

The first parameter will refer to the water pump at the specified tier. For example, if the first parameter is ‘2’, then the MCU will know that the water pump that it needs to operate will be on the second tier. The third parameter for both the water pump, and the nutrient pump will be largely ignored. This is because, the call to operate either of the pumps will involve the pumps to stay on for a preset interval of time.

The preset interval of time that the water pump will stay on will be hard coded into the MCU. This value will be constant, because the interval time value should never change. This value will be determined after the testing phase of the project as stated in the Sensor Grid Prototype Testing section under the water pump subsection.

Nutrient Pumps Function

In the case that the component that the second parameter is referring to is the water pump, the MCU will begin to operate the water pump. The MCU will do this by using a nested switch case statement. Depending on the composition of the parameters of the function, the MCU will decide whether or not to turn on the nutrient pump.

The first parameter will refer the nutrient pump at the specified tier. For example, if the first parameter is ‘2’, then the MCU will know that the nutrient pump that it needs to operate will be on the second tier. The third parameter for both the water pump, and the nutrient pump will be largely ignored. This is because, the

call to operate either of the pumps will involve the pumps to stay on for a preset interval of time.

The preset interval of time that the nutrient pump will stay on will be hard coded into the MCU. This value will be constant, because the interval time value should never change. This value will be determined after the testing phase of the project as stated in the Sensor Grid Prototype Testing section under the Nutrient Pump subsection.

LED Lighting Strips

In the case that the component that the second parameter is referring to is the LED array, the MCU will begin to adjust the LEDs. The MCU will do this by using a nested switch case statement. Depending on the composition of the parameters of the function, the MCU will decide whether or not to turn on or off the specified LED strip.

The first parameter of the call_MCU function will determine the tier that the LEDs are being adjusted on. For example, if the parameter is ‘2’, then the MCU will know that the LED array that needs to be adjusted will be on the second tier of greenhouse. The third parameter for the LED will be used to determine whether to turn on or off the LED array.

The third parameter of the function can be easily implemented with the Arduino. The Arduino makes this easy because we can use the digital pins of the Arduino to activate a relay that will toggle an LED either on or off. Since the digital pin of the Arduino can be set to either be used as an input or output pin, toggling LEDs can be done quite effectively.

7.10 - SOC Software Design

The SOC software is in charge of the majority of components that are attached to the network/sensor grid, this would include the EC sensor, pH sensor, and water level sensors. The SOC is also in charge of maintaining a connection to the backend API system to administer the changes as defined above to the MCU.

It will operate off of five main functions currently. These functions are getMCUData(tierNum), callMCU(tierNum, component, amount) sendAPIData (tierData, tierNum), getAPIData(tierNum), and performAction(tierData, desiredRange, tierNum). The first function takes in the number of the tier that data is being requested of. The function will then communicate to the MCU that data is being requested for a specific tier. The MCU will then gather the data from the specified tier's sensors and return a string with all the required information. The callMCU() function will take in the tier number, the component type as a string, and the amount that the component object may need. The sendAPIData() function is called every time the main loop goes through a tier and receives the data readings of a tier. This tier reading is sent to the API so that it can be stored in the database for future use. The getAPIData() function is used to retrieve the desired range of the a specified tier. The performAction() function takes in a class object that stores the data from a tier that has been read, and another class object that stores the desired range that the tier data should be at, and the specified tier number. These parameters are generated by the two previous functions, and with that information the performAction() function, using if statements, makes calls to the MCU to adjust the tiers using the callMCU() function. This will include whether or not to activate the water or nutrient pumps depending on the data that it was given.

This single function design of the performAction() function to call to the MCU was decided upon due to the fact that it will keep the communication between the two components simple and will allow the code flow to be easier to understand.

Electrical Conductivity Sensors

The EC sensor will continuously be sending a value reading to the MCU board, per the sensor design it is just monitoring the parts per a million of compounds present in the water, but when the function calls the pin that this sensor is set to will report a reading, this reading will be reread multiple times and averaged out for find the actual reading. This variable is stored and marked as ready to be used for further calculation to be able to send to the API for adjustments. The expected nutrient concentration for each plant will be stored in the API but a local copy will also be brought down into the SOCs memory for comparison during actual calculations. This value is one of the variables used to determine the proper concentration of nutrients and because of this, this variable is one that directly affects the pumping system in the main function of performAction().

pH Sensors

The pH sensor will continuously be sending a value reading to the MCU board, per the sensor design it is just monitoring the concentration of hydrogen ions that are present in the water, but when the function calls the pin that this set sensor reading is connected to will be read for the reported value. This reading will be reread multiple times and averaged out to calculation of the actual reading.This variable is stored and marked as ready to be used for further calculation to be able to send to the API for adjustments. The expected nutrient concentration for each plant will be stored in the API but a local copy will also be brought down into the SOCs memory for comparison during actual calculations. This value is one of the variables used to determine the proper concentration of nutrients and because of this, this variable is one that directly affects the pumping system in the main function of performAction().

Water Level Sensors

The water level sensors are actually two bobber based aquarium sensors with the appropriate level being oriented properly and the low level sensor being inverted. The low level water sensor behaves similarly to the other sensors already mentioned, meaning that it will be always at the ready to be polled for information. After the appropriate function calls are done on the SOC, the getMCUData() function will be called and the MCU will then poll all of the sensors for the specified tier and that data will be collected and then sent back to the SOC for further function calls.

continue to report that it is low until it has been corrected. When the check tiers function is called it will read the pin that this low water level sensor is connected to for each associated tier, this sensor is simple in design as stated and only operates in a true or false. This means that no further calculation is needed to interpret the reading.The SOC will store this in a tierReading object and will return it to the API for record keeping. The water level is manipulated entirely upon the SOC meaning that the SOC will issue the callMCU function for as many times as it takes for the appropriate water level sensor to activate at this point the conditional statement that is allowing for the repeated issuing of the function will break and end. Since this function relies entirely upon the SOC it will execute more quickly which will account for the fact that the system will have to wait for each iteration of the pumping to occur until the appropriate level is reached. The last condition tied to the use of the water level sensor is if the appropriate water level sensor is triggered then in any callMCU function that will adjust a fluid level either water or nutrient will not be completed. The callMCU function to adjust the water or nutrient levels will then be completed when the water levels have gone down accordingly and there is no longer a risk of the tier overflowing.

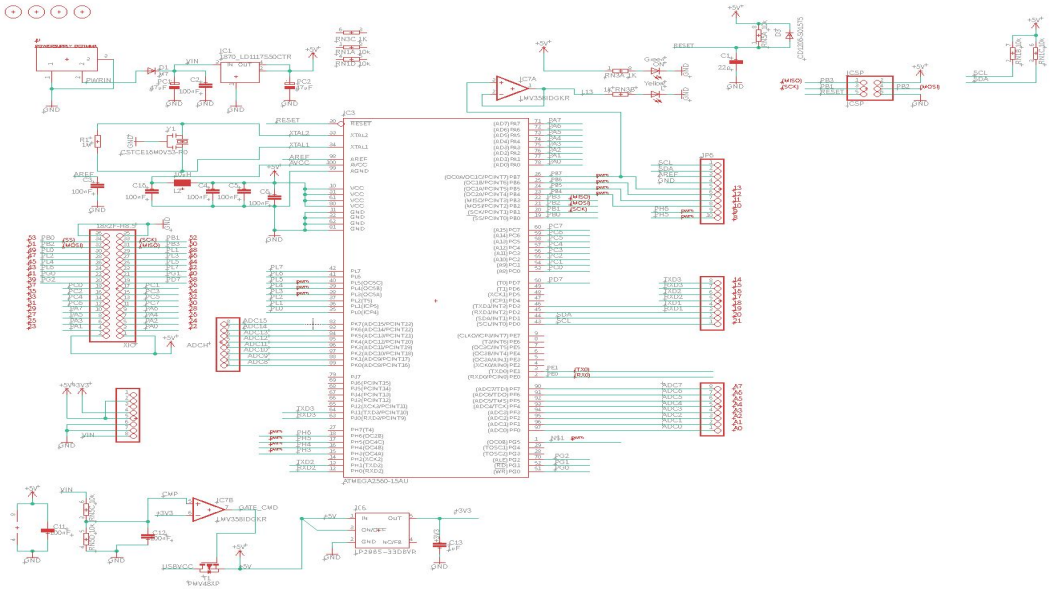
7.11 - MCU Hardware Design

After a lot of deliberations the team chose to go with a MCU model based upon the Arduino Mega 2650. This board has allowed us to have a multitude of pins and communication types to communicate with the SOC and because of that we decided that this board has enough capability to allow us to base our models off of it. Most of the core functionality is being maintained and is going to remain in the system. The current plan will be to remove some of the analog pins connections and strip down some of the digital pin connections to make the board more compact and since we only need 20 pins maximum to maintain the connections to the relays and potentiometers that are needed in the system to control the LEDs and pumps. The final MCU schematic can be seen in Figure 7.11.1.

Since most of the board is completely functional for what we need the core functionality will be identical to the arduinos. The central difference between the two boards will be the rerouting of the traces in the boards and the changes in the locations of the elements and components that are located on the board.

Due to the outbreak of the Covid-19 virus we weren't able to get a fully working PCB and will have to use a development board in our system. This will complicate our design due to the fact we won't be able to directly solder wires to our pcb and will have to try and mount them to the development board instead. We are going to mount them with more silicon epoxy that was used to waterproof the rest of the water system which will be easy to strip off when we want to use the development board later.

Figure 7.11.1: MCU Schematic Layout



8.0 - Backend Design Details

When building the backend for our project, the team considered the tools and frameworks needed, the API endpoints that would be implemented and the process of authentication and authorization. Additionally, the team reviewed the use cases for our backend as well as the components that would make up the backend.

8.1 - Libraries and Modules

npm: npm is a popular JavaScript package manager that allows developers to install custom packages for use in your project. We chose to use npm as our package manager because of the previous experience we have using this package manager, as well as the large variety of useful packages available to use through npm.

Express: Express is a web application framework that provides developers with an easy-to use-set of features for building a custom REST API. We chose Express as a framework for building the API because of past experience working with this framework. Additionally, Express allows us to create an API quickly while offering useful features without sacrificing performance. We will be using Express on our Node.js server to allow the sensor grid to communicate with the mobile application and the SQL database. More details regarding Express will be included further in the documentation.

MySQL Community Server: MySQL Community Server is an open-source database we will use to store sensor data, greenhouse data and user information. While building our application, we will be hosting our MySQL database on the Senior Design server for testing purposes. When the project is nearing completion, we will migrate our database to Amazon Relational Database Service. More information regarding Amazon Relational Database Service will be provided. To avoid incurring high database and server costs, we will build and test the application on the Senior Design server and move our backend over to the cloud computing offered by Amazon once the project is completed.

Amazon Relational Database Service: Amazon Relational Database Service (RDS) is a distributed relational database service hosted in the Amazon Cloud. We chose Amazon RDS because of its popularity in the industry, as well as its scalability and ease to set up. Because Amazon provides a free-tier for users, we will build and test our MySQL database and backend using the Senior Design server and migrate over to Amazon Web Services upon completion of the project. Amazon RDS currently supports MySQL Community Server.

8.2 - API Endpoints

The API endpoints are divided into four groups. Authentication endpoints are grouped together under the /auth route. Endpoints used by the sensor grid are grouped together under the /sensorgrid route. Endpoints utilized by the mobile app when interacting with the backend are grouped together under the /mobileapp route. Endpoints utilized by the administrator's portal are grouped together under the /adminportal route.

Authentication

GET - Get authentication token
Get an authentication token for given user credentials
/auth/get_token
Response:
{
token: "QI+mlFAbqtzvLxroEVFbZOswfmDPs4nwKtZ+hGtXE+A="
}
AUTHORIZATION HEADER

email	user_test@gmail.com
password	passw0Rd1

GET - Get authentication token for admin user
Get an authentication token for given admin user credentials
/auth/get_token_admin
Response:
{
token: "QI+mlFAbqtzvLxroEVFbZOswfmDPs4nwKtZ+hGtXE+A="
}
AUTHORIZATION HEADER

email	admin@gmail.com
password	passw0Rd1

POST - Reset password
Send a reset password command to backend
/auth/reset_password
Response:
{
200:"OK"
}
HEADERS

Content-Type	application/x-www-form-urlencoded

BODY

email user_test@gmail.com

POST - Change password
Change the user's password
/auth/change_password
Response:
{
    200: "User Password Changed"
}
HEADERS
Content-Type application/x-www-form-urlencoded
BODY
new_password pa$sw)rd1
old_password passw0RD1
email user_test@gmail.com

POST - Create user
Create a user with provided username and password
/auth/create_user
Response:
{
    200: "User Created"
}
HEADERS
Content-Type application/x-www-form-urlencoded
BODY
email test@gmail.com
password pas$W0rD!

Mobile App

GET - Get all greenhouses
Retrieves all greenhouses for a specific user
/mobileapp/greenhouses/
Response:
{
    greenhouses: [298379827453, 872932938740, 234234235467]
}
HEADERS
Content-Type application/x-www-form-urlencoded
AUTHORIZATION HEADER
Bearer Token token
PUT - Update a specified tier of greenhouse
Update a specified tier of the greenhouse with provided values
/mobileapp/tiers/:greenhouse_id/:tier

Response:
{
	200: "Updated Tier"
}
HEADERS
Content-Type application/x-www-form-urlencoded
AUTHORIZATION HEADER
Bearer Token token
BODY
plant_id 9
light_start 2
cycle_time 32:00:00

GET - Get a specified tier of greenhouse
Get all data for a specified tier of the greenhouse
/mobileapp/tiers/:greenhouse_id/:tier
Response:
{
	plant_id: 029468109359,
	growth_stage: 2,
	plant_type_id: 3,
	water_level: 23.00,
	cycle_time: 24.00,
	ph_level: 7.80,
	ec_level: 2.10
}
AUTHORIZATION HEADER
Bearer Token token
POST - Create a new greenhouse
Create a new greenhouse with provided values
/mobileapp/greenhouses/
Response:
{
	200: "Greenhouse Created"
}
HEADERS
Content-Type application/x-www-form-urlencoded
AUTHORIZATION HEADER
Bearer Token token
BODY
name greenhouse_test
serial_no 234234234234
grid_ password password
seedling_time 2020-01-01

PUT - Update a greenhouse
Update greenhouse with provided values
/mobileapp/greenhouses/:greenhouse_id
Response:
{
    200: "Updated Greenhouse"
}
HEADERS
Content-Type application/x-www-form-urlencoded
AUTHORIZATION HEADER
Bearer Token token
BODY
name new_greenhouse
seedling_time 2019-10-10

DEL - Delete a greenhouse
Delete greenhouse with specified greenhouse_id
/mobileapp/greenhouses/:greenhouse_id
Response:
{
    200: "Greenhouse Deleted"
}
AUTHORIZATION HEADER
Bearer Token token

GET - Get sensor reading for a single sensor type
Get the current sensor reading for a single sensor type
/mobileapp/sensor/:greenhouse_id/:tier/:sensor_type
Response:
{
    reading: 23.00
}
AUTHORIZATION HEADER
Bearer Token token

GET - Get sensor readings for single tier
Get all sensor readings for specified tier in greenhouse
/mobileapp/sensor/:greenhouse_id/:tier
Response:
{
    water_level: 23.00,
    ph_level: 7.90,
    ec_level: 1.00
}
AUTHORIZATION HEADER

Bearer Token token

GET - Get sensor readings for greenhouse
Get all sensor readings for all levels of greenhouse

/mobileapp/sensor/:greenhouse_id
Response:

```
{
        "readings": [
                { "ph_level": 0.00, "ec_level": 0.00, "water_level": 0.00, "tier": 1 }, ]
}
```

AUTHORIZATION HEADER
Bearer Token token

GET - Get a greenhouse
Get values for a greenhouse with specified greenhouse_id
/mobileapp/greenhouses/:greenhouse_id
Response:

```
{
        "water_level": 102.00,
        "nutrient_level": 0.00,
        "light_level": 10.00,
        "seedling_time": "2019-09-09T16:00:00.000Z",
        "backup_batt_level": 90.00,
        "power_source": 1,
        "greenhouse_name": "My Greenhouse 1"
}
```

AUTHORIZATION HEADER
Bearer Token token

GET - Get greenhouse history
Get readings for greenhouse for last 30 days
/mobileapp/greenhouses/history/:greenhouse_id/
Response:

```
{
        "history": [
                {
                        "date": "2019-11-08T16:21:53.000Z",
                        "water_level": 102.00,
                        "nutrient_level": 200.00,
                        "battery": 94.00,
                        "power_source": 0,
                        "greenhouse_id": 1,
                        "light_level": 10.00
                }]
```

}
AUTHORIZATION HEADER
Bearer Token token

GET - Get greenhouse and tier data
Get greenhouse information and its tier data
/mobileapp/greenhouses/detail/:greenhouse_id/
Response:

```
{
   "greenhouse_id": 6,
   "name": "YO ITS A GREENHOUSE",
   "water_level": 0,
   "nutrient_level": 0,
   "battery": 90,
   "light_level": 10,
   "power_source": 1,
   "seedling_time": null,
   "tiers": [
      {
         "tier": 1,
         "plant_id": null,
         "ph_level": 8.2,
         "ec_level": 2.5,
         "water_level": 34,
         "cycle_time": null,
         "light_start": null
      },
   ]
}
```

AUTHORIZATION HEADER
Bearer Token token

POST - Add device key for user
Add device key for receiving notifications
/mobileapp/devices
Response:

```
{
        200: "Device Key Added"
}
```

HEADERS
Content-Type application/x-www-form-urlencoded
AUTHORIZATION HEADER
Bearer Token token
BODY
devicekey ExponentPushToken[g5paj4PLG4CBPLrBTCwEaz]

DEL - Delete a device key
Delete device key for user
/mobileapp/devices
HEADERS
Content-Type application/x-www-form-urlencoded
BODY
devicekey ExponentPushToken[g5paj4PLG4CBPLrBTCwEaz]

Response:
{
200: "Device Key Deleted"
}
AUTHORIZATION HEADER
Bearer Token token

POST - Classify plant image
Classify plant image using CNN
/mobileapp/classification
Response:
{
"200": "Classification Complete",
"prediction": "ripe-tomato",
"probability": 0.9990648627281189
}
HEADERS
Content-Type application/x-www-form-urlencoded
BODY
devicekey ExponentPushToken[g5paj4PLG4CBPLrBTCwEaz]

Sensor Grid

POST - Post sensor readings for greenhouse
Post all sensor readings for all levels of greenhouse
/sensorgrid/sensor/greenhouse
Response:
{
200: "Sensor Readings Recorded"
}
HEADERS
Content-Type application/x-www-form-urlencoded
AUTHORIZATION HEADER
Basic Auth username:password
BODY
power_supply 0

backup_battery_level	94.00
tier1_water	34.00
tier1_ph	8.20
tier1_ec	2.50
tier2_water	35.00
tier2_ph	7.80
tier2_ec	3.20
tier3_water	32.00
tier3_ph	8.00
tier3_ec	2.10
tier4_water	33.00
tier4_ph	7.90
tier4_ec	1.90
water_level	102.00
nutrient_level	342.00
light_level	10.00

POST - Post general readings for greenhouse
Post general readings for all tiers of greenhouse
/sensorgrid/sensor/greenhouse/general
Response:
{
    200: "General Greenhouse Readings Recorded"
}
HEADERS

Content-Type	application/x-www-form-urlencoded

AUTHORIZATION HEADER

Basic Auth	username:password

BODY

power_supply	0
backup_battery_level	94.00
water_level	102.00
nutrient_level	342.00
light_level	10.00

POST - Post sensor readings for single tier
Post all the sensor readings for a specific tier
/sensorgrid/sensor/:tier
Response:
{
    200: "Sensor Readings Recorded"
}
HEADERS

Content-Type	application/x-www-form-urlencoded

AUTHORIZATION HEADER

Basic Auth username:password
BODY
ph_level 6.80
water_level 22.00
ec_level 1.10

POST - Post sensor reading for a single sensor type
Post a reading for a single sensor
/sensorgrid/sensor/:tier/:sensor_type
Response:
{
200: "Sensor Reading Recorded"
}
HEADERS
Content-Type application/x-www-form-urlencoded
AUTHORIZATION HEADER
Basic Auth username:password
BODY
reading 23.00

PUT - Update power source
Update the greenhouse's current power source
/sensorgrid/powersource
Response:
{
200: "Power Source Reading Recorded"
}
HEADERS
Content-Type application/x-www-form-urlencoded
AUTHORIZATION HEADER
Basic Auth username:password
BODY
source 0 wall = 0, battery = 1

PUT - Update backup battery level
Update the current backup battery level
/sensorgrid/backuplevel
Response:
{
200: "Battery Level Reading Recorded"
}
HEADERS
Content-Type application/x-www-form-urlencoded
AUTHORIZATION HEADER
Basic Auth username:password

BODY
battery_level 90.00

GET - Get tiers' ideal values
Get the ideal values for each plant in each tier of the greenhouse
/sensorgrid/adjustments/tierdata
Response:

```
{
   "tiers": [
      {
         "user_id": 24,
         "tier": 1,
         "greenhouse_id": 6,
         "plant_id": null,
         "light_start": null,
         "ph_level_high": null,
         "ph_level_low": null,
         "ec_level_high": null,
         "ec_level_low": null,
         "light_time": null
      },
   ]
}
```

AUTHORIZATION HEADER
Basic Auth username:password

Admin Portal

GET - Get all plant ideals
Get plant ideal values from the database
/adminportal
Response:

```
{
        {
                "plant_id": 1,
                "ph_level_low": null,
                "ec_level_low": null,
                "temp_low": null,
                "cycle_time": null,
                "ph_level_high": null,
                "ec_level_high": null,
                "temp_high": null,
                "name": "empty",
                "light_time": null,
                "steps": null,
                "plant_url": null,
```

```
            "harvest_url": null,
            "num_plants": null
        },
}
```


POST - Create a plant ideal
Add new plant to database
/adminportal
Response:


```
{
    200: "Plant Ideal Added"
}
```


HEADERS	
Content-Type	application/x-www-form-urlencoded
AUTHORIZATION HEADER	
Bearer Token	token
BODY	
ph_level_low	5.0
ph_level_high	9.0
ec_level_low	1.0
ec_level_high	2.0
temp_low	15.0
temp_high	35.0
name	Raddish
cycle_time	85
light_time	10
steps	“This is a test step. This is another test step.”
plant_url	www.googl1.com
harvest_url	www.google.com
num_plants	10

DEL - Delete a plant ideal
Delete plant ideal from database
/adminportal/plant_id
Response:


```
{
    200: "Plant Ideal Deleted"
}
```


HEADERS	
Content-Type	application/x-www-form-urlencoded
AUTHORIZATION HEADER	
Bearer Token	token

PUT - Update plant ideal

Update existing plant in database
/adminportal/plant_id
Response:
{
200: "Plant Ideal Updated"
}

HEADERS

Content-Type	application/x-www-form-urlencoded

AUTHORIZATION HEADER

Bearer Token	token

BODY

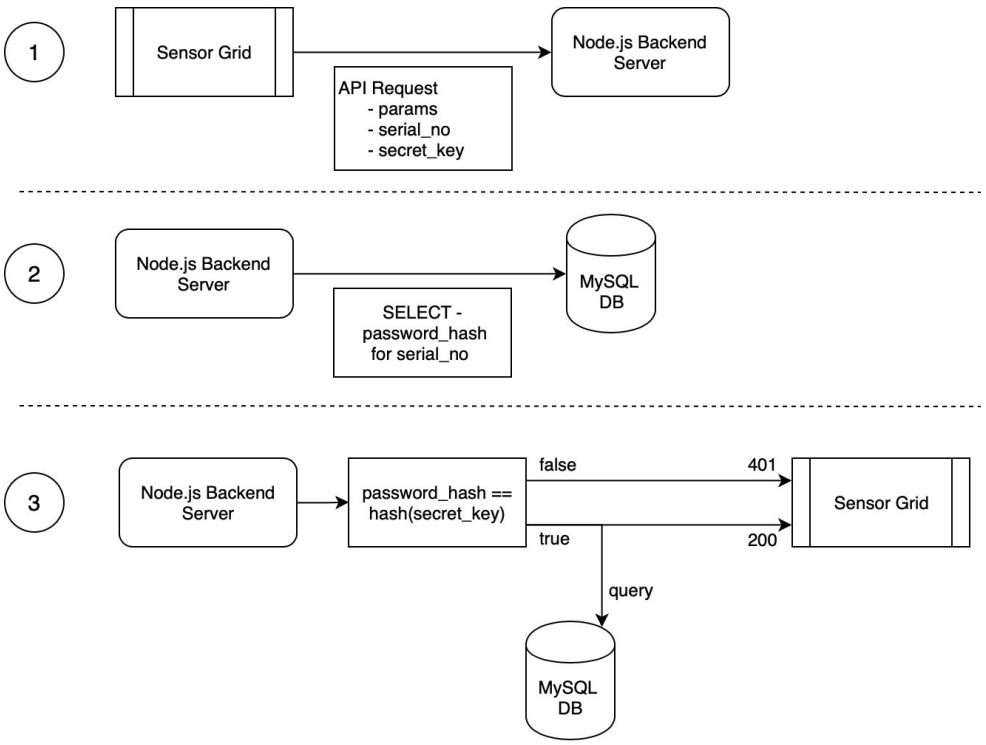
ph_level_low	5.0
ph_level_high	9.0
ec_level_low	1.0
ec_level_high	2.0
temp_low	15.0
temp_high	35.0
name	Raddish
cycle_time	85
light_time	10
steps	“This is a test step. This is another test step.”
plant_url	www.googl1.com
harvest_url	www.google.com
num_plants	10

8.3 - Authentication Flow

The authentication flow follows three steps to ensure security, and to provide a secure handshake with the sensor grid (Figure 8.3.1).

1. Sensor Grid makes an API request to the Node.js Backend Server with the appropriate parameters of the request and the sensor grid's credentials: serial number and secret key
2. Node.js Backend Server queries the MySQL database for the password hash that matches the provided serial number
3. Node.js Backend Server compares the queried password hash with a hash of the provided secret key
    a. If the provided secret key's hash matches the hash found in the database, perform the action requested by the Sensor Grid
    b. If the provided secret key's hash doesn't match the hash found in the database, return a 401 Unauthorized response.

Figure 8.3.1: Authentication Flow Steps

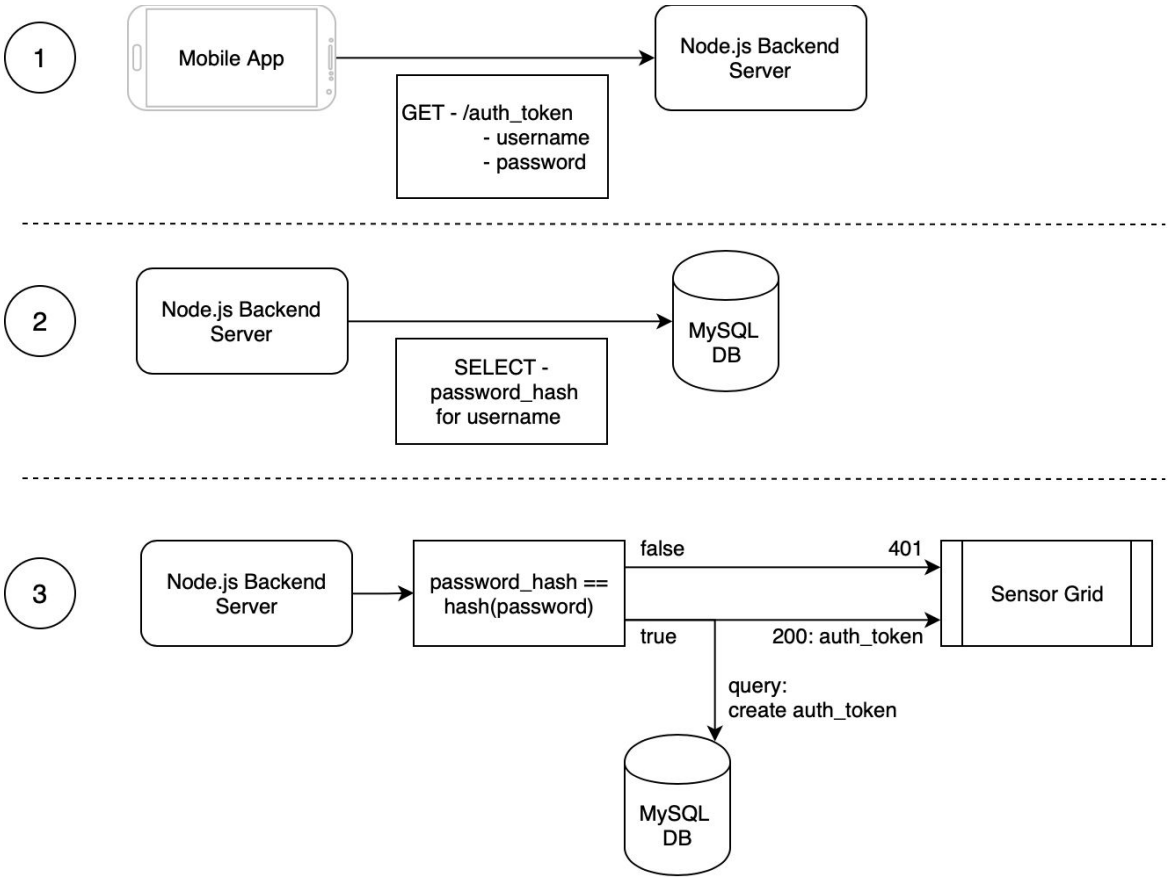


Because the sensor grid doesn't make as many requests as the mobile app, we decided to use the serial number and secret key for every API request instead of issuing a temporary authorization token. The sensor grid will make POST requests on a set time interval so issuing a temporary token would be unnecessary.

The authentication flow follows a similar three step process to connect with the app, ensuring security. This authentication flow occurs every time the app needs to reconnect with the server after an authentication token has expired or after the user logs out (Figure 8.3.2).

1. Mobile app makes a GET request for an authentication token and provides the user credentials to the Node.js Backend Server
2. The Node.js Backend Server queries the MySQL database for the password hash that matches the provided username
3. Node.js Backend Server compares the queried password hash with a hash of the provided password
    a. If the provided password's hash matches the hash found in the database, create a new authentication token and return the token as a response to the GET request
    b. If the provided password's hash doesn't match the hash found in the database, return a 401 Unauthorized response.

Figure 8.3.2: Authentication Flow of Sensor Grid



8.4 - Authorization Flow

When the Node.js backend server receives an API request from the Sensor Grid, query the database for user_id, the secret key hash and the greenhouse_id that correspond to the provided serial number. If the hash of the provided secret key matches the stored secret key hash in the database, use the user_id and greenhouse_id that was returned by the database to perform the API request the Sensor Grid made. This allows the backend to verify that the sensor grid is allowed to perform actions on the database records and ensures that the sensor grid can only perform actions on behalf of the user it is authorized by.

When the Node.js Backend Server receives an API request from the mobile app, query the database for the user_id that corresponds to the authentication token provided in the request. Verify that the token exists and is still valid. If the token has expired, return a 401 Unauthorized response. But if the token is valid, the same record in the database that verifies that the user is allowed to perform actions on the database records also provides the backend with the user_id that the device is allowed to interact with. This prevents a user from accessing another user's data because their token can only access records that have the same user_id as the token's user_id.

The database will contain two tables that manage the authentication and authorization. The first table is the Active Sessions table. Each record in this table will store an authentication token, the expiration date for ensuring the token is only valid temporarily, and the user_id this token belongs to. The user_id ensures that a token can only be used to access the corresponding user's data and no one else. The tokens are temporary to ensure that password changes affect all logged in devices.

The second table is the Sensor Grids table. Each record in this table will store a sensor grid, the user it belongs to, the greenhouse it is associated with and the sensor grid's secret key hash. This ensures that each sensor grid can only access a single user's data, but more specifically a certain greenhouse that user owns. The secret key hash and serial number combination only allow the sensor grid to perform actions for a specific user_id and greenhouse_id. Without the correct combination of serial number and secret key the sensor grid cannot access another user_id and greenhouse_id combination.

Secret key values and password values are hashed by the backend before being stored in the database. This is to ensure that user credentials cannot be stolen from the database in case of a breach. If an attacker was able to maliciously access and steal credentials from our database, they would be unable to log into our app using the password hash or publish sensor data from a sensor grid using the secret key hash.

8.5 - Backend Interaction Use Cases

One of the main use cases of our project is the collection of sensor readings by the sensor grid. The sensor grid will collect the readings from each greenhouse level and send them to the backend for storage. The backend will receive sensor readings from the sensor grid and store them in the appropriate tables in the database.

First the sensor grid collects data points for sensors on a single level of greenhouse. Next the sensor grid repeats step 1 for all sensors in the greenhouse. After that, the sensor grid collects power source information, backup battery level and water and nutrient levels for greenhouse. Then the sensor grid makes POST requests to backend providing sensor readings and other general greenhouse information. The backend receives a POST request from the sensor grid, and then the backend executes queries to store data points in MySQL database.

Another main use case of our project is the monitoring of the greenhouse from the mobile app. The user will open the mobile app and select the greenhouse they would like to view. The mobile app will retrieve the information about the current status of the greenhouse.

First the mobile app user selects the greenhouse they would like to view data about. Then the mobile app makes a GET request to the backend to collect information about the water tank and nutrient tank levels as well as power source information and backup battery level. After that, the backend queries the database for the appropriate records that correspond to the greenhouse_id provided by the mobile app in the GET request. Finally, the backend provides the data as a response to the initial GET request made by the mobile app.

For each greenhouse level, the user may use the mobile app to monitor more granular data points such as water level, pH level, and conductivity levels. The mobile app will retrieve the information from the backend for the selected level of the greenhouse

First the mobile app user selects a specific level of the greenhouse. Then the mobile app makes a GET request to the backend to collect information about the current water level, pH level and electrical conductivity for that specific level. Finally, the backend queries the database to retrieve the sensor readings for the specified level and provides the data as a response to the initial GET request made by the mobile app.

One of the use-cases the user will carry out after purchasing the product is creating a greenhouse. The user will create a greenhouse using the mobile app and link the greenhouse_id to the sensor grid. The mobile app will make a POST

request to the backend providing the greenhouse's information and the backend will query the database to create the appropriate records.

First the mobile app user will create a new greenhouse and give it a name. Then the mobile app will make a POST request with the greenhouse information. The mobile app user will link the sensor grid to the greenhouse by providing the sensor grid's serial number and secret key, and then the mobile app will make a PUT request to update the greenhouse with appropriate information. The backend will make queries to database to change the records accordingly. The mobile app user will add new levels to the greenhouse, providing a plant type. The mobile app will make a PUT request to the backend to update the greenhouse level with the new information. Finally, the backend will make queries to database to change the records accordingly

When the user downloads the app, the first use case they will perform is the login process. The mobile app user will input their credentials. The mobile app will then send these credentials to the backend. The backend will verify the credentials and issue a token if these credentials are valid.

First, the mobile app user inputs username and password and initiates login. Then the mobile app makes a GET request to the backend for an authentication token, with the username and password as parameters. Then the backend receives a GET request for the authentication token and queries the database for the password hash for the username provided in the request. The backend takes the password provided by the request and performs hashing operation, and then the backend compares the password hash found in the database to the hash value to the calculated hash in step 4. If the password hash found in the database matches the calculated value, the backend issues an authentication token as a response to the initial GET request. Finally, the backend executes a query to input the new token into the database with the token value, the user it belongs to and its expiration date.

8.6 - Backend Components Overview

The backend of the Pocket Ponics will contain two components: a MySQL database and a Node.js server. The MySQL database and Node.js server will be hosted on the senior design servers as we build and test the project but when the project is complete the backend will be migrated over to the Amazon Web Services platform. The two components and their interactions will be described in more detail below.

The MySQL database was selected because of the clear relationships that exist between our data points. Storing the data in a relational database made storage, retrieval and insertion easier and more organized. Data is stored in tables with relationships between those tables. While relational databases are more rigid, we don't anticipate any changes to the structure of the data that would be difficult to implement after the fact. The rigidity of the data structure is overshadowed by the ease of querying the data from the database when the mobile app makes requests. The MySQL database will store authentication information such as the active tokens and who they belong to, as well as their expiration date.

Additionally, greenhouse information and the information about the levels in each greenhouse will be stored. When storing each greenhouse's levels, current sensor readings for pH, water level and electrical conductivity will be stored as well as growth stage, the type of plant growing and the number of plants in the level. General information about each greenhouse will be stored, such as water tank levels and nutrient levels. Power source and backup battery levels will be stored as well.

The Node.js server will run the API and will house our endpoints for the interactions between the sensor grid and the backend, as well as interactions between the mobile app and the backend. The sensor grid and mobile app will interact with the MySQL database indirectly, through the backend.

The API will contain endpoints grouped into three routes: /sensorgrid, /mobileapp and /auth. The /sensorgrid route will contain endpoints that the sensor grid will use to record sensor readings in the database, as well as the endpoints for checking for any adjustments the sensor grid needs to make to the greenhouse levels' water and/or nutrient levels.

The /mobileapp route will contain endpoints that the mobile app will use to indirectly interact with the MySQL database. Endpoints for create, delete, get and update operations will be located here, so that the mobile app can perform these operations on greenhouses and their levels. Endpoints for interacting with greenhouses and levels will be separate from endpoints that access sensor readings. For example, getting information about the specific level such as the plant type and number of plants in the level is completed through a separate

endpoint than the endpoint used for accessing the current pH reading, water level and electrical conductivity reading for that specific level.

The /auth route will contain endpoints that the mobile app uses for authentication operations such as login, creating a user, changing a user's password and resetting a user's password.

Each group of endpoints will be stored in its own file. For example, there should be three files for the routes that exist; the mobileAppController.js, the sensorGridController.js and the authenticationController.js. Additionally, there will be another file named databaseController.js that contains the functions for interacting with the MySQL database.

Database Controller Functions

- getSensorReading(String user_id, String greenhouse_id, int level, int sensorType)
    - This function gets the sensor reading for specified greenhouse, level and sensor
- getSensorReadingsForLevel(String user_id, String greenhouse_id, int level)
    - This function gets the sensor readings for specified greenhouse and level
- getSensorReadingsForGreenhouse(String user_id, String greenhouse_id)
    - This function gets the sensor readings for a specified greenhouse that belongs to a certain user, or returns an error if the user does not have permissions to access the specified greenhouse
- setSensorReading(String user_id, String greenhouse_id, int level, int sensorType)
    - This function stores the sensor reading for the specified greenhouse, level and sensor
- setSensorReadingsForLevel(String user_id, String greenhouse_id, int level, JSON readings)
    - This function sets the sensor readings for specified greenhouse and level
- setSensorReadingsForGreenhouse(String user_id, String greenhouse_id, JSON readings)
    - This function sets the sensor readings for specified greenhouse
- createUser(String username, String password)
    - This function creates a user with specified username and password
    - This function also hashes the password before storage, ensuring it is protected
- getUserPasswordHash (String username)
    - This function gets the password hash for the specified username
- getUserID(String token)

    - This function gets the user_id for the specified token
- setUserPasswordHash(String username, String password)
    - This function sets the password hash for the specified username
    - This function also hashes the new password before storage, ensuring it is protected
- hashPassword(String password)
    - Given a password, this function produces the hash value
- getGreenhouses(String user_id)
    - This function gets all the greenhouse_id values for a specified user_id
- getGreenhouse(String user_id, String greenhouse_id)
    - This function gets the greenhouse information for a specified greenhouse, or returns an error if the user does not have permissions to access the specified greenhouse
- updatePowerSource(String user_id, String greenhouse_id, int source)
    - This function updates the power source for a specified greenhouse with provided value
- updateBackupBatteryLevel(String user_id, String greenhouse_id, int level)
    - This function updates the backup battery level for a specified greenhouse with provided value
- updateGreenhouseLevel(String user_id, String greenhouse_id, int level, int plant_type_id, int growth_stage, int num_plants, int cycle_time, int light_level)
    - This function updates the specified greenhouse's level with the provided information, or returns an error if the user does not have permissions to access the specified greenhouse
- getGreenhouseLevel(String user_id, String greenhouse_id, int level)
    - This function gets the level data for the level of a specified greenhouse, or returns an error if the user does not have permissions to access the specified greenhouse
- createGreenhouse(String user_id, String name)
    - This function creates a new greenhouse for specified user with specified name
- updateGreenhouse(String user_id, String greenhouse_id, String name, String sensor_grid_serial, String sensor_grid_secret)
    - This function updates the specified greenhouse with the provided information, or returns an error if the user does not have permissions to access the specified greenhouse
    - This function also hashes the sensor grid secret before storage, ensuring it is protected
- deleteGreenhouse(String user_id, String greenhouse_id)
    - This function deletes the specified greenhouse, or returns an error if the user does not have permissions for the specified greenhouse

8.7 - Testing Strategies

The testing for the backend will be divided into two groups. First, the backend will undergo unit testing to ensure each endpoint is tested for expected results. Afterwards, the interaction between components will be tested as part of integration testing.

Unit Tests

Backend unit tests will be conducted using Jest and Postman. API endpoint testing will be written and executed in Postman, using the Collection Runner. Testing will be conducted using a test environment, with a test version of the backend connected to a MySQL database clone designated for testing purposes. When tests are conducted, Postman will attempt to insert sensor readings and greenhouse data into the database, simulating the process that the sensor grid will take. Postman will also attempt to retrieve data from our test database given a test user's credentials. We will be using the Jest framework to test the database controller functions.

For evaluating the test coverage, we will be using the Postman Collection Runner and Jest's included coverage feature. The Postman Collection Runner will indicate how many tests exist for each endpoint in the collection, as well as the number of tests that passed and failed during the last test run. Jest will check the test coverage for our database controller. For Jest tests, the tests will be contained in separate files, separated by the file they test. For example, the databaseController.js file will contain the functions for interacting with the database and the tests will be contained in the databaseController.test.js file.

Integration Testing

Integration testing can be done using the Postman Collection Runner. Running the tests in Postman Collection Runner will also require manual verification that the database contains the correct data after running the integration tests. An example integration test attempts to insert sensor data collected by the sensor grid into the database using the backend API endpoints. A Postman test will attempt to make an API request with the appropriate information, and then the developer would verify that the information in the database is correct after this request is made to ensure both parts of the backend's integration with the project work correctly. This test would ensure the sensor grid can post data to the backend successfully and that the backend can write to the database correctly.

8.8 - Relational Database

Although some consideration was originally given to using a non-relational database, it was eventually decided that it was beneficial to have a relational database. The system has to analyze the incoming data in order to adjust itself, thus MySQL was chosen as the management system for the database. This will allow the data to be connected according to user information and the respective greenhouse the data is being stored under. The reason why we decided to go with a relational database is because of the way the data needs to be retrieved. Since the data will all be layered and interconnected, for example tiers will need to be associated with greenhouses and users, it is beneficial to use MySQL (or other relational database systems) because they have a fast retrieval time. In addition, after being created the data structure will not be changing.

The data is going to be coming from sensors in the greenhouses. There will be a sensor grid table that they will be stored under. They will communicate with the database using the backend API, which will then feed and retrieve data between the database and the sensors. Using the retrieval functions in the API and the database queries, the system will be able to react to the sensor readings (conditions) inside the greenhouses in accordance to the plant's best interest. They will also be able to communicate to the user when it's time for them to refill tanks, water seedlings, water plants, etc. so that the user can maintain the system and keep it running efficiently.

As mentioned above, the way our project is going to process the data received and stored is by using functions and queries. The backend functions will be able to retrieve sensor data, create and update user information, retrieve, update and delete greenhouse data. Similarly to those functions, various queries will allow the system to add, retrieve and manipulate data from the database. This will function as the semi-automatic part of our project. In addition, the queries will update historical data with the day's readings, and also be able to compare the numbers in the tiers table with the plant ideal table in order to determine what the adjustments table needs to have.

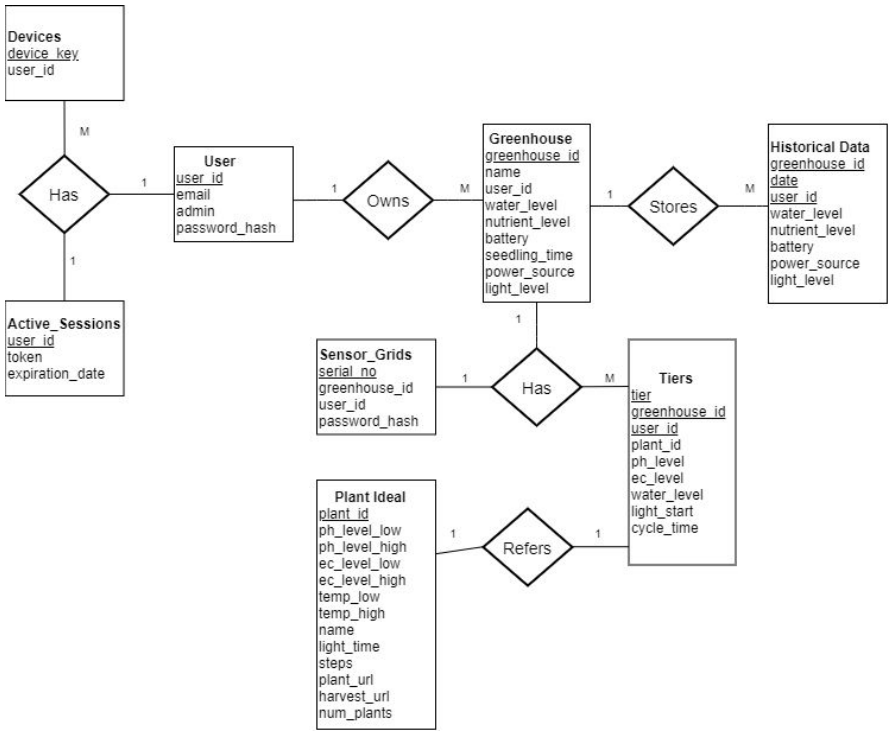
After determining what the system needs to care for the plans, the API will then be able to send to the correct sensor the actions that they need to take to fix the problems. This will include sending more water to the plants and changing the nutrient amounts to each tier. The sensor grid will make the necessary changes to the greenhouse, which will then show up the next time that the greenhouse sends data to the API. While the greenhouse makes adjustments, the database will show that the tier is in an undesirable state, and the app will display the fact that the greenhouse is adjusting.

8.9 - Entities and Attributes

The database is going to have eight tables. The tables will be user, active session, sensor grids, greenhouse, historical data, tiers, plant ideal, and adjustments. User will account for individual users, active session will track when a user is logged in, sensor grids will keep track of each individual sensor in every greenhouse, greenhouse will store their tanks data, historical data will store that data for future retrieval, tiers will store the current sensor data, plant ideal will store the numbers that the sensor readings should be reading and adjustments will keep track of what tier needs to be changed in order to match ideal growing conditions. Each table must have a primary key(s), data that should be held and the foreign keys that link the tables to other tables in the database.

The tables are also linked together with a series of chained relationships, as shown in Figure 8.9.1. The user has a single active session, leading to a one-to-one relationship, between those two tables. The user also owns several greenhouses, leading to a one-to-many relationship between those tables. From there, the greenhouse stores many historical data records and tiers, giving the greenhouse a one-to-many relationship with both of those tables. The tier also has a many-to-one relationship with the plant ideal table, as a single ideal plant record can correspond to many different tiers across various greenhouses.

Figure 8.9.1: Database entity-relational diagram



User Table

The user table will be used to store user information accordingly. They will be able to use their account and link to greenhouses. This table will assign them a local number as an identifier. Their email and secure password will also be stored. An example of how user data is stored is shown in Table 8.9.1.

Attributes:

- user_id: The primary key of this table. An integer, used locally, for identifying users
- email: A string, stores the email the users used to sign up
- password_hash: The user's password, hashed for security

Table 8.9.1: User Sample Table

user_id	email	password_hash
1	johndoe@example.com	D98HNOgsNpDczl
2	janesmith@email.com	qm3Jq7KnEd1rVAG
3	bobjohn@edu.com	J8Pv3Fykk1aSJG5

Active_Sessions Table

This table will be used as a means of storing a user's active token. This will allow them access to their data as long as they're logged in. In addition, it will keep an expiration date for the authentication token, which will eventually need to be deleted and created again to update the token as needed. An example of how session data is stored is shown in Table 8.9.2.

Attributes:

- user_id: A foreign key, relates this table with a specific user
- token: Authentication token
- expiration_date: Allows tokens to expire

Table 8.9.2: Active Sessions Sample Table

user_id	token	expiration_date
1	eyJ0eXAiOiJKV1QiLCJ	2019-11-30 03:43:31
2	iJ9.otJtZXNzYWdlIjo	2019-12-06 14:26:17
3	iSldUIFJ1bGVzISIsIml	2019-12-19 07:54:13

Devices Table

This table is used to keep track of the devices associated with the user. One user can set up and monitor more than one greenhouse, and this will track the devices using their device key.

Attributes:

- user_id: A foreign key, relates the users
- device_key: The device key

Table 8.9.4: Devices Sample Table

user_id	device_key
1	BICAV6OAJ
2	IPA85BSIDV
3	ALD39JF921

Sensor_Grid Table

This table is used to relate users and their greenhouses to the sensors that monitor the plants. The sensors will be kept track via their serial numbers and the password to access the data will also be stored securely. An example of how sensor data is stored is shown in Table 8.9.3.

Attributes:

- user_id: A foreign key, relates the sensors with users
- greenhouse_id: A foreign key, relates the sensors with a specific greenhouse
- serial_no: Identifier for each sensor, to get single sensor readings
- password_hash: The sensor grid password, hashed for security

Table 8.9.4: Sensor Grids Sample Table

user_id	greenhouse_id	serial_no	password_hash
1	1	LS1DGK8DLS	BFFB760DB
2	2	4CE0460DG6	12CAD6AFA
3	3	BK564TF9L	F96ABBF41

Historical Data Table

Multiple times during the day, the readings of the greenhouse will be copied to this table and assigned the current date. Along with the greenhouse's readings, the greenhouse and user identifier are included to allow users to access past data directly. The data stored here will be used to show the user levels over time so they can spot patterns and plan accordingly. An example of how historical data is stored is shown in Table 8.9.4a and Table 8.9.4b.

Attributes:

- date: The date for this data
- greenhouse_id: Foreign key, relates this table to the greenhouse table
- user_id: Foreign key, relates this table to the user table
- water_level: How full the water tank is
- nutrient_level: How full the nutrient tank is
- battery: Whether the greenhouse is being powered with the battery
- power_source: If the system is running on battery power
- light_level: How much light the greenhouse got that day

Table 8.9.5a: Historical Data Sample Table

date	greenhouse_id	user_id	water_level
2019-11-15 11:45:00	1	1	90.54
2019-11-20 11:45:00	2	2	40.64
2019-11-28 11:45:00	3	3	68.65

Table 8.9.5b: Historical Data Sample Table (Continued)

nutrient_level	battery	light_level	power_source
15.86	90.25	75.40	0
27.62	95.90	29.31	0
13.54	41.46	64.54	1

Greenhouse Table

This table will keep track of the general information of the greenhouse. This includes the levels of both, water and nutrient, tanks, light level in the whole

greenhouse and the dates the seedlings were planted. In addition, it will monitor how the greenhouse is being powered (power or battery) and the battery charge. Each greenhouse will be assigned a local number as an identifier which will then be used to link each unit to the user and their id. An example of how greenhouse data is stored is shown in Table 8.9.5a and Table 8.9.5b.

Attributes:

- greenhouse_id: The primary key of this table. Identifies the greenhouses with integers
- user_id: Foreign key relating this table to the user table
- name: Stores the greenhouse's name. Users can name each greenhouse
- water_level: How full the water tank is, this will be used to alert the user when it drops below a certain percentage
- nutrient_level: How full the nutrient tank is, this will be used to alert the user when it drops below a certain percentage
- battery: Stores the battery level
- seedling_time: When the seedlings were planted, so that the system can alert the user when it's time to water them
- light_level: The amount of light in the greenhouse
- power_source: Whether the greenhouse is being powered with the battery

Table 8.9.6a: Greenhouse Sample Table

greenhouse_id	user_id	name	water_level	nutrient_level
1	1	John's Plants	90.54	15.86
2	2	Garden	40.64	27.62
3	3	Bob's Food	68.65	13.54

Table 8.9.6b: Greenhouse Sample Table (Continued)

battery	seedling_time	light_level	power_source
90.25	2019-05-05 12:45:27	75.40	0
95.90	2019-01-13 17:21:34	29.31	0
41.46	2019-07-31 08:17:34	64.54	1

Tiers Table

The plant's latest readings will be stored in this table. The tier they're on will serve as the primary identifier, as there can only be similar type plants on the same shelf. Each plant will be identified according to a number assigned to them, which will be used to compare the current readings with the ideal reading in the Plant Ideal table. The number of plants and the growth stage will also be kept track of, the former allows the adjustments of calculations depending on the demands and the latter accounts for the fact that every stage will have different needs. Tracking these two will make sure the conditions are being adjusted using the correct data for comparison. An example of how tier data is stored is shown in Table 8.9.6a and Table 8.9.6b.

Attributes:

- tier: The primary key. The level in which the plants are on. This will be used to know which level needs adjusting
- greenhouse_id: Foreign key, the greenhouse in which the plants are in
- user_id: Foreign key, the user these plants belong to
- plant_id: The type of plant that is on this tier. This is used to compare to the plant ideal table and check that the conditions match for ideal growth
- ph_level: The level of the ph in the water
- ec_level: The nutrient level of the water
- water_level: The water level in the tier
- cycle_time: How long the plant has been growing
- light_start: When the lights were turned on

Table 8.9.7a: Tiers Sample Table

tier	greenhouse_id	user_id	plant_id	growth_stage
1	1	1	7	2
1	2	2	4	1
3	3	3	3	1

Table 8.9.7b: Tiers Sample Table (Continued)

ph_level	ec_level	water_level	cycle_time	num_plants
5.82	2.56	7.25	18:45:00	5
6.01	1.32	4.38	17:02:00	2

4.85	1.20	2.66	19:32:00	4

Plant Ideal Table

The plant's ideal conditions will be stored here. The tiers table can link with this table using the unique plant identifier. The medium ideal conditions are stored and the current conditions can be compared and adjusted accordingly. An example of how plant data is stored is shown in Table 8.9.7a and Table 8.9.7b

Attributes:

- plant_id: The primary key. The number identifier for a particular plant
- growth_stage: The primary key. The stage of growth of the plant
- ph_level_med: The medium ph level acceptable for this plant
- ec_level_med: The medium nutrient level acceptable for this plant
- water_level_med: The medium water level acceptable for this plant
- temp_med: The medium temperature acceptable for this plant
- cycle_time: How long the plant should receive light

Table 8.9.8a: Plant Ideal Sample Table

plant_id	growth_stage	ph_level_med	ec_level_med
1	1	6.25	3.00
1	2	5.75	2.75
3	2	4.50	1.75

Table 8.9.8b: Plant Ideal Sample Table (Continued)

water_level_med	temp_med	cycle_time
7.00	75.00	12:30:45
5.50	60.00	12:28:41
6.75	85.00	12:56:12

8.10 - Database Schema

This section will provide information on the tables for the database. Each table will have their attributes along with the data type and the initial values they should be set to, if any. It will also show setting up the primary keys, constraints and unique indexes as they should apply.

Creating the Schema

In this schema, the foreign keys were defined using constraints to allow for a user-defined name to each foreign key created. Since there are several foreign keys associated with multiple tables, the order in which the tables are created matters. MySQL will give an error if the table attempts to reference a foreign key for a table that hasn't been created yet.

User Table

	Column name	Datatype	Initial Value
CREATE TABLE `user`	`user_id`	int(11)	NOT NULL
	`email`	varchar(128)	NOT NULL
	`password_hash`	varchar(128)	NOT NULL

Primary keys:
- `user_id`

Constraints:
- Unique keys: `user_id`, `email`, `password_hash`

Active Sessions Table

	Column name	Datatype	Initial Value
CREATE TABLE `active_sessions`	`token`	varchar(128)	NOT NULL
	`expiration_date`	datetime	NOT NULL
	`user_id`	int(11)	NOT NULL

Primary keys:
- `token`

Foreign keys:
- `user_id`

Constraints:
- Foreign key: `user_id_fkas`

Sensor Grid Table

	Column name	Datatype	Initial Value
CREATE TABLE `sensor_grids`	`serial_no`	varchar(45)	NOT NULL
	`password_hash`	varchar(128)	NOT NULL
	`user_id`	int(11)	NOT NULL
	`greenhouse_id`	int(11)	NOT NULL

Primary key:
- `serial_no`

Foreign keys:
- `user_id`
- `greenhouse_id`

Constraints:
- Unique keys: `greenhouse_id`, `password_hash`, `serial_no`
- Foreign keys: `greenhouse_id_fksg`, `user_id_fksg`

Historical Data Table

	Column name	Datatype	Initial Value
CREATE TABLE `greenhouse`	`date`	datetime	NOT NULL
	`light_level`	decimal(5,2)	NOT NULL
	`water_level`	decimal(5,2)	NOT NULL
	`nutrient_level`	decimal(5,2)	NOT NULL
	`battery`	decimal(5,2)	NOT NULL
	`power_source`	tinyint(4)	NOT NULL
	`greenhouse_id`	int(11)	NOT NULL
	`user_id`	int(11)	NOT NULL

Primary keys:
- `date`,`user_id`,`greenhouse_id`

Foreign keys:
- `user_id`, `greenhouse_id`

Constraints:
- Foreign keys: `greenhouse_id_fkhd`, `user_id_fkhd`

Greenhouse Table

	Column name	Datatype	Initial Value
CREATE TABLE `greenhouse`	`greenhouse_id`	int(11)	NOT NULL
	`name`	varchar(45)	NOT NULL
	`water_level`	decimal(5,2)	DEFAULT `0.00`
	`nutrient_level`	decimal(5,2)	DEFAULT `0.00`
	`battery`	decimal(5,2)	DEFAULT `0.00`
	`seedling_time`	datetime	DEFAULT NULL
	`power_source`	tinyint(4)	DEFAULT `0`
	`light_level`	decimal(5,2)	DEFAULT `0.00`
	`user_id`	int(11)	NOT NULL

Primary key:
- `greenhouse_id`

Foreign keys:
- `user_id`

Constraints:
- Unique keys: `greenhouse_id`
- Foreign keys: `user_id_fkgr`

Adjustments Table

	Column name	Datatype	Initial Value
CREATE TABLE `devices`	`user_id`	int(11)	NOT NULL
	`device_key`	varchar(45)	NOT NULL

Unique keys:
- `device_key`

Tiers Table

	Column name	Datatype	Initial Value
	`tier`	int(11)	NOT NULL
	`growth_stage`	int(11)	DEFAULT `0`
	`plant_id`	int(11)	DEFAULT NULL

CREATE TABLE `tiers`	`ph_level`	decimal(5,2)	DEFAULT '0.00'
	`ec_level`	decimal(5,2)	DEFAULT '0.00'
	`water_level`	decimal(5,2)	DEFAULT '0.00'
	`cycle_time`	time	DEFAULT NULL
	`num_plants`	int(11)	DEFAULT `0`
	`greenhouse_id`	int(11)	NOT NULL
	`user_id`	int(11)	NOT NULL

Primary keys:

- `tier`,`user_id`,`greenhouse_id`

Foreign keys:

- `user_id`, `greenhouse_id`, `plant_id`

Constraints:

- Foreign keys: `greenhouse_id_fkt`, `user_id_fkt`, `plant_id_fkt`

Plant Ideal Table

	Column name	**Datatype**	**Initial Value**
CREATE TABLE `plant_ideal`	`plant_id`	int(11)	AUTO-INCREMENT
	`ph_level_low`	decimal(5,2)	DEFAULT NULL
	`eh_level_low`	decimal(5,2)	DEFAULT NULL
	`temp_low`	decimal(5,2)	DEFAULT NULL
	`cycle_time`	int(11)	DEFAULT NULL
	`ph_level_high`	decimal(5,2)	DEFAULT NULL
	`ec_level_low`	decimal(5,2)	DEFAULT NULL
	`temp_high`	decimal(5,2)	DEFAULT NULL
	`name`	varchar(45)	DEFAULT NULL
	`light_time`	int(11)	DEFAULT NULL
	`steps`	varchar(45)	DEFAULT NULL
	`plant_url`	varchar(45)	DEFAULT NULL
	`harvest_url`	varchar(45)	DEFAULT NULL
	`num_plants`	int(11)	DEFAULT NULL

Primary keys:

- `plant_id`

Constraints:

- Unique keys: `plant_id`, `name`

Sample Schema Queries

Using the details outlined in previously, this section will demonstrate some of the complete queries in making the database. There are three examples; a basic table (no foreign keys, no default values), a table with foreign keys (no default values) and a table with foreign keys and default values.

Basic Table

User Table

```
CREATE TABLE `user` (
        `user_id` int(11) NOT NULL AUTO_INCREMENT,
        `email` varchar(128) NOT NULL,
        `password_hash` varchar(128) NOT NULL,
        PRIMARY KEY (`user_id`),
        UNIQUE KEY `user_id_UNIQUE` (`user_id`),
        UNIQUE KEY `email_UNIQUE` (`email`),
        UNIQUE KEY `password_hash_UNIQUE` (`password_hash`)
);
```

Table with Foreign Key

Sensor Grid Table

```
CREATE TABLE `sensor_grid` (
         `serial_no` varchar(45) NOT NULL,
         `password_hash` varchar(128) NOT NULL,
         `user_id` int(11) NOT NULL,
         `greenhouse_id` int(11) NOT NULL,
         PRIMARY KEY (`serial_no`),
         UNIQUE KEY `greenhouse_id_UNIQUE` (`greenhouse_id`),
         UNIQUE KEY `password_hash_UNIQUE` (`password_hash`),
         UNIQUE KEY `serial_no_UNIQUE` (`serial_no`),
         KEY `greenhouse_id` (`greenhouse_id`),
         KEY `user_id` (`user_id`),
          CONSTRAINT `greenhouse_id_fksg` FOREIGN KEY (`greenhouse_id`)
        REFERENCES `greenhouse` (`greenhouse_id`),
          CONSTRAINT `user_id_fksg` FOREIGN KEY (`user_id`) REFERENCES
        `user` (`user_id`)
);
```

Table with Initial Values

Greenhouse Table

```
CREATE TABLE `greenhouse` (
        `greenhouse_id` int(11) NOT NULL AUTO_INCREMENT,
        `name` varchar(45) NOT NULL,
        `water_level` decimal(5,2) DEFAULT '0.00',
        `nutrient_level` decimal(5,2) DEFAULT '0.00',
        `battery` decimal(5,2) DEFAULT '0.00',
        `seedling_time` datetime DEFAULT NULL,
        `power_source` tinyint(4) DEFAULT '0',
        `light_level` decimal(5,2) DEFAULT '0.00',
        `user_id` int(11) NOT NULL,
        PRIMARY KEY (`greenhouse_id`),
        UNIQUE KEY `greenhouse_id_UNIQUE` (`greenhouse_id`),
        KEY `user_id` (`user_id`),
        CONSTRAINT `user_id_fkgr` FOREIGN KEY (`user_id`) REFERENCES
      `user` (`user_id`)
);
```

Inserting Sample Data into the Database

To check the basic integrity of the database as its being created, inserting some sample data locally will be useful. The following section outlines a couple queries for locally inserting data into several of the tables.

There are two ways of formatting the INSERT INTO statement to add data into the tables. The first way is by adding the information to all the columns in a table at once (see example 1). This way just requires the table name and the values to be inserted.

Example 1: Inserting a new user

```
INSERT INTO user
VALUES (‘1’, ‘johndoe@example.com’, ‘password’);
```

The second way of formatting INSERT INTO is by specifying what columns in the table you are adding the data to (see examples 2 and 3). This way requires the table name, the columns that are being modified and the values being inserted.

Example 2: Inserting a new greenhouse

```
INSERT INTO greenhouse (greenhouse_id, name, user_id)
VALUES (‘1’, ‘Johns Plants’, ‘1’’);
```

Example 3: Inserting a new user

```
INSERT INTO user (email, password_hash)
```

VALUES (‘johndoe@example.com’, ‘password’);

Since the primary key of the user table is set to AUTO_INCREMENT, the user id will automatically get filled in when a new row is added. This way the backend takes care of assigning identifiers to users as they sign up rather than have to do it manually.

Functions are also allowed to be inserted as values (see example 3). This is especially useful when adding ‘datetime’ datatypes into the database. The two main functions that can be used when inserting data are NOW(), which returns the current date and time, and TIME(), which only returns the current time.

Example 4: Inserting a new active session

```
INSERT INTO active_sessions
VALUES (‘token’, NOW(), ‘1’);
```

To retrieve the information just added, you can run the following simple query to see the information inside a selected table (SELECT * FROM table_name)

In this case, when the query is run thrice using `user`, `active_sessions` and `greenhouse` for the table names, they should return the following data:

User

user_id	email	password_hash
1	johndoe@example.com	password

Active Sessions

token	expiration_ate	user_id
token	2019-11-06 15:30:41	1

Greenhouse*

greenhouse_id	name	user_id
1	Johns Plants	1

*Along with all the greenhouse attributes and their respective default values.

8.11 - Database Queries

These queries can be used to manipulate data inside the tables, and are leveraged by the API endpoints. Data is modified mostly by the app, but the greenhouses also need to access the information to maintain proper nutrient ratios and to manage the lighting system. The queries are called from within the Javascript, and are not stored directly within the database.

Inserting Information into the Database

These queries will be used to insert information into the database tables using the given values.

- INSERT INTO tiers (tier, greenhouse_id, user_id)
  VALUES (tier, ‘greenhouse_id’, ‘user_id’)
  - Inserts tier number, greenhouse id and user id into the tiers table
  - Used in function createEmptyTiersAndGridForNewGreenhouse()
  - This is part of the transaction (see Transactions) for creating a new greenhouse
- INSERT INTO sensor_grid (serial_no, password_hash, user_id, greenhouse_id)
  VALUES ('serial_no', 'password_hash', user_id, greenhouse_id)
  - Inserts a new sensor grid associated with a new greenhouse
  - Used in function createEmptyTiersAndGridForNewGreenhouse()
  - This is part of the transaction (see Transactions) for creating a new greenhouse
- INSERT INTO historical_data (date, water_level, nutrient_level, battery, power_source, greenhouse_id, user_id, light_level)
  VALUES (NOW(), water_level, nutrient_level, battery, power_source, greenhouse_id, user_id, light_level)
  - Inserts the current readings of the greenhouse into the historical data table
  - Used in the functions updateReadingsForGreenhouse(), updatePowerSourceForGreenhouse() and updateBatteryForGreenhouse()
  - This is part of the transaction (see Transactions) for updating a greenhouse’s readings
- INSERT INTO adjustments
  VALUES (adjustment_type, amount, user_id, tier, greenhouse_id
  - Inserts an adjustment into the table for the greenhouse and tier specified along with the amount and its type
  - Used in the function createAdjustmentForGreenhouse()

- INSERT INTO user (email, password_hash)
  VALUES ("email", 'password_hash')
    - Inserts a new user into the database, it assigns them a unique user identifier automatically and stores the email and password provided
    - Used in the function createUser()
- INSERT INTO active_sessions (token, expiration_date, user_id)
  VALUES ('token', 'expiration', user_id)
    - Inserts an active authentication token with an expiration date, associated with a user, into the active sessions table
    - Used in the function insertTokenForUser()

Retrieving Information from the Database

These queries will be used to retrieve specific information from the database using the given values. Queries will only succeed if the user has the correct permissions to access the requested data, and will return an error if the user does not have the correct permissions.

- SELECT tiers.water_level FROM tiers
  INNER JOIN plant_ideal
  ON (tiers.plant_id = plant_ideal.plant_id and tiers.growth_level = plant_ideal.growth_level)
  WHERE tiers.water_level <> plant_ideal.water_level_med
    - Returns water levels that are below or above the ideal conditions stored, so that the greenhouse can make adjustments to the water level using the pumps
- SELECT tiers.ec_level FROM tiers
  INNER JOIN plant_ideal
  ON (tiers.plant_id = plant_ideal.plant_id and tiers.growth_level = plant_ideal.growth_level)
  WHERE tiers.ec_level <> plant_ideal.ec_level_med
    - Returns nutrient levels that are below or above the ideal conditions stored so that the greenhouse can make adjustments to the nutrient level using the pumps
- SELECT tiers.ph_level FROM tiers
  INNER JOIN plant_ideal
  ON (tiers.plant_id = plant_ideal.plant_id and tiers.growth_level = plant_ideal.growth_level)
  WHERE tiers.ph_level <> plant_ideal.ph_level_med
    - Returns pH levels that are below or above the ideal condition so that the greenhouse can make adjustments to the nutrient level using the pumps

- SELECT ph_level, ec_level, water_level FROM tiers
  WHERE user_id =user_id and greenhouse_id = greenhouse_id and tier = tier
    - Returns pH, nutrient and water levels from the specified user, greenhouse and tier so that the app can display the information
    - Used in the function getReadingsForGreenhouse()
- SELECT amount FROM adjustments
  WHERE user_id = user_id and greenhouse_id = greenhouse_id and tier = tier and adjustment_type = adjustment_type
    - Returns the adjustment amount of given adjustment type from the specified user, greenhouse and tier so that the app can display the information
    - Used in the function createAdjustmentForGreenhouse()
- SELECT user_id, password_hash FROM user
  WHERE email = email
    - Returns the user identification and secure password associated with a particular email
    - Used in the function getHashForUser()
- SELECT user_id, greenhouse_id, password_hash FROM sensor_grid
  WHERE serial_no = serial_no
    - Returns the user and greenhouse identifiers as well as the secure password associated with a specific sensor
    - Used in the function getHashForSensorGrid()
- SELECT * FROM historical_data
  WHERE user_id = user_id and greenhouse_id = greenhouse_id and date >= start_date and date < end_date
    - Returns all the information stored in the historical data table for specified user and greenhouse for the selected dates
    - Used in the function getGreenhouseHistoricalData()
- SELECT greenhouse_id FROM greenhouse
  WHERE user_id = user_id
    - Returns the identifier for a greenhouse associated with the given user id
    - Used in the function getGreenhousesForUser()
- SELECT user_id from active_sessions
  WHERE expiration_date > NOW() and token = given_token
    - Returns an active authentication token for the user, and errors out if there are no active tokens avalible
    - Used in the function getUserForToken()

Updating Information from the Database

These queries will be used to update existing information in the database tables using the given values.

- UPDATE tiers
  SET plant_id = plant_id, growth_stage = growth_stage, cycle_time = cycle_time, num_plants = num_plants
  WHERE user_id = user_id and tier = tier and greenhouse_id = greenhouse_id
    - Updates the tiers table with the newly added plants in the greenhouse
    - Used in the function updateTierForGreenhouse()
- UPDATE greenhouse
  SET name = name, seedling_time = seedling_time
  WHERE user_id = user_id and greenhouse_id = greenhouse_id
    - Updates a greenhouse's name and seedling time for the given greenhouse and user
    - Used in the function updateReadingsForGreenhouse()
- UPDATE tiers
  SET water_level = tier_number.water_level, ph_level = tier_number.ph_level, ec_level = tier_number.ec_level
  WHERE user_id = user_id and greenhouse_id = greenhouse_id and tier = tier_number
    - Updates the water level, ph level and nutrition level of a given tier associated with the user and greenhouse
    - Used in the function updateReadingsForGreenhouse()
- UPDATE adjustments
  SET amount = amount
  WHERE user_id = user_id and greenhouse_id = greenhouse_id and tier = tier and adjustment_type = adjustment_type
    - Updates the adjustment table with the given amount using the specified user, greenhouse, tier and amount type
    - Used in the function createAdjustmentForGreenhouse()
- UPDATE tiers
  SET sensor_type = reading
  WHERE user_id = user_id and greenhouse_id = greenhouse_id and tier = tier
    - Updates a specific sensor's readings (water_level, ec_level, etc.)
    - Used in the function updateReadingsForSensorType()
- UPDATE user
  SET password_hash = password_hash
  WHERE user_id = user_id

    - Updates the user's hashed password
    - Used in the function updateUserHash()
- UPDATE greenhouse
  SET power_source = power_source
  WHERE user_id = user_id and greenhouse_id = greenhouse_id
    - Updates the power source of a given greenhouse
    - Used in the function updatePowerSourceForGreenhouse()
- UPDATE greenhouse
  SET battery = battery
  WHERE user_id = user_id and greenhouse_id = greenhouse_id
    - Updates the battery of a given greenhouse
    - Used in the function updateBatteryForGreenhouse()

Deleting Information from the Database

These queries will be used to delete information from the tables.

- DELETE FROM active_sessions
  WHERE user_id = user_id
    - Deletes rows matching the given user id
    - Used in the function revokeTokens()
- DELETE FROM tiers
  WHERE greenhouse_id = greenhouse_id and user_id = user_id
    - Deletes the tiers associated with the given greenhouse and user identifiers
    - Used in the function deleteGreenhouseForUser()
- DELETE FROM adjustments
  WHERE greenhouse_id = greenhouse_id and user_id = user_id
    - Deletes any pending adjustments to the given greenhouse
    - Used in the function deleteGreenhouseForUser()
- DELETE FROM historical_data
  WHERE greenhouse_id = greenhouse_id and user_id = user_id
    - Deletes any stored data to the given greenhouse
    - Used in the function deleteGreenhouseForUser()
- DELETE FROM sensor_grid
  WHERE greenhouse_id = greenhouse_id and user_id = user_id
    - Deletes the sensor grid associated to the given greenhouse and user identifiers. It is used to cascade deletions when deleting a greenhouse
    - Used in the function deleteGreenhouseForUser()
- DELETE FROM greenhouse
  WHERE user_id = user_id and greenhouse_id = greenhouse_id
    - Deletes the greenhouse associated to the given greenhouse and user identifier
    - Used in the function deleteGreenhouseForUser()

Transactions

A transaction is a group of mySQL statements that are performed as if they were a single unit. Transactions are a way to ensure the integrity of the database remains when inserting, updating or deleting elements in the tables. The way they work is by ensuring that all the statements that need to be run succeed, or the transaction fails and the database returns to the state it was in before the transaction started.

This database has several places where a transaction is necessary in order to perform certain statements. The different cases that warranted the use of transactions are listed here.

Creating a greenhouse
When creating a greenhouse there are two tables that must be simultaneously created as well; the tiers and sensor grid tables. Greenhouses cannot be inserted into the database without these two tables because tiers and sensor grid are found within the greenhouse structure.

Deleting a greenhouse
Similarly to creating a greenhouse, when deleting a row from the greenhouse table all the data that is connected to that row must also be deleted from the database. The tiers inside that greenhouse, the sensor grid connected to it, the historical data it has stored and any adjustments it may still have pending must also be deleted with it.

Updating a greenhouse's readings
As with creating and deleting a greenhouse, the tables depending on the readings for a greenhouse must also be updated when it gets updated. In this case, there are only two other tables that must be updated alongside the greenhouse table; the tier and historical data tables.All of the tiers in the greenhouse will get an update on their sensor readings, which include water and nutrient levels amongst other attributes. In addition, the greenhouse's history must be updated to reflect the latest changes.

Updating a greenhouse's power source
If there is an update that requires a change to the greenhouse's power source, then both the greenhouse and the greenhouse's historical data must be updated.

Updating a greenhouse's battery
As with updating the power source, when a change in the greenhouse's battery is detected, the change must be recorded alongside the historical data.

9.0 - Front End Design Details

The front end mobile app is the main point of interaction for the users. The mobile app retrieves information about the database that is stored in the backend, and then displays it to the user in an easily understandable format. The user can then use the information that the app presents and the step-by-step guides provided in the app to manage the installation and harvesting of their greenhouse.

9.1 - Use Case Diagram

Figure 9.1.1 shows the frontend use case diagram. When the user opens the app, there are three major actions that they can take. First, they can set up a new greenhouse, which prompts the app to create a greenhouse and register it with the backend. Users can also check the status of existing greenhouses, which prompts the app to display the greenhouse and to compile historical data for the greenhouse (after retrieving it from the database). The user can also harvest plants, which prompts the app to update the tiers and set the tier status on the backend. There is also one database prompt that occurs; when a timer expires or levels get too low, the database will prompt the app to display a notification, so that the user can take an action.

Figure 9.1.1: Frontend Use Case Diagram

9.2 - Libraries and Modules

The app will be written in Javascript, with React Native providing the user interface framework. The Javascript base allows us to write modular code that is easily testable, and it allows us to maintain a single coherent language across the code base, matching up with the Node.js backend, which is also written in Javascript. This consistency between the frontend and backend makes the code easier to maintain, and reduces the learning curve needed for this project.

Similarly, React Native, in conjunction with the Expo command line interface, allows us to create dynamic, modular views that can be reused across the code base. This reduces the complexity of our codebase, and ensures that bugs can be identified and corrected more easily. Furthermore, since Expo complies the React Native code into both iOS and Android applications, React Native allows us to provide a consistent user interface across all devices, regardless of operating system or screen size.

In addition to these basal libraries, we are using React Navigation to coordinate the navigation between different screens of the app. Since React Navigation has many flexible types of navigation, it allows us to provide both the stack based navigation that we need for our main greenhouse, tier view, and registration screens, while also allowing us the flexibility that we need for authentication flows like the login and logout processes. Furthermore, it lets us adaptively control what the top navigation bar contains, allowing us the flexibility of custom, adjustable titles and specialized navigation buttons.

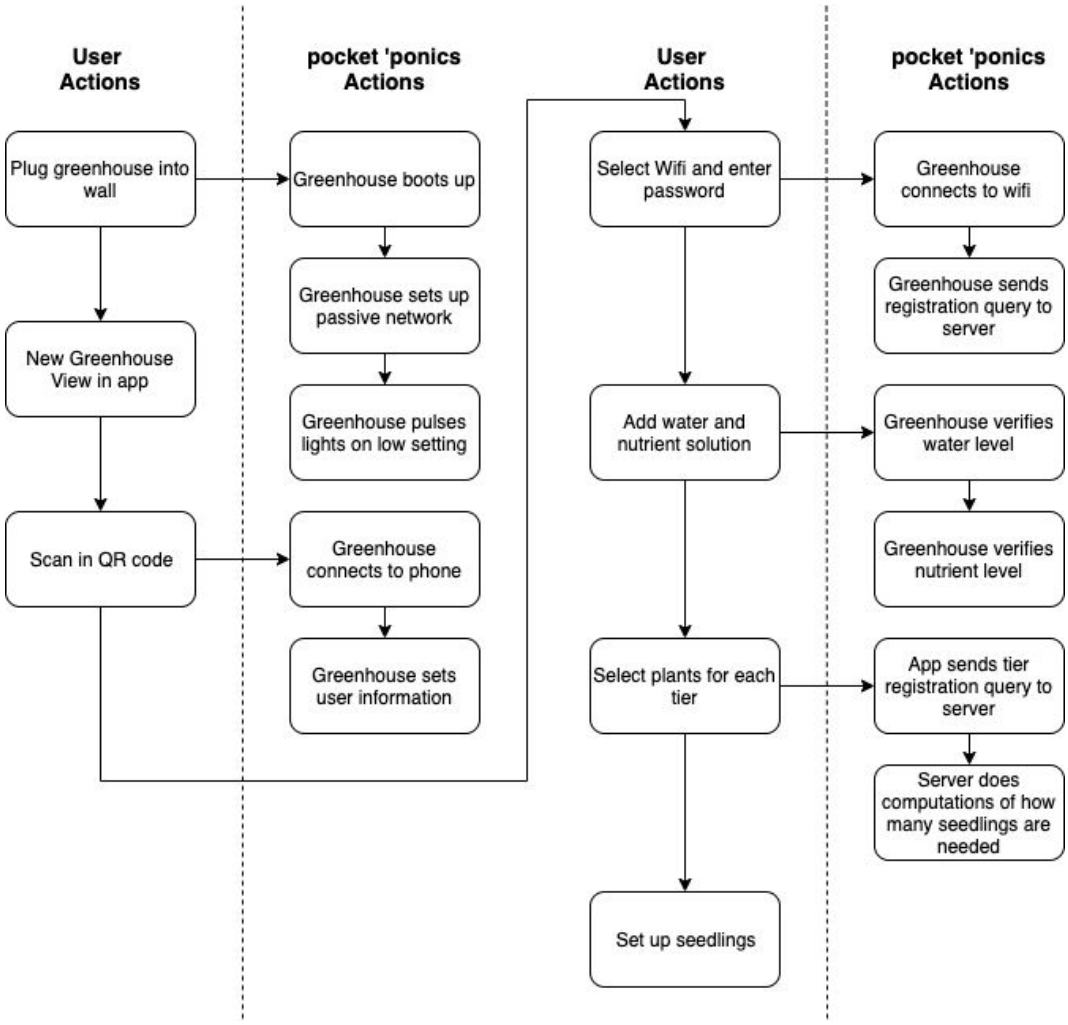
For the greenhouse list, we decided to leverage the react-native-snap-carousel library, which allows us to create a fluid carousel so that users can swipe left and right between greenhouses with ease. Although we looked into several simpler libraries before selecting react-native-snap-carousel, none of the other libraries were able to provide a swiping interface that only worked in a single direction. Since the user's ability to scroll down through the cards in a single greenhouse was very important, we eventually determined that react-native-snap-carousel was our best option.

The final library that we decided to use is the react-native-chart-kit. This will allow us to display the historical data of the greenhouse in a meaningful way, and also allow us to customize the color scheme and design so that it integrates well with the user experience of our app. It also accepts data in a wide array of formats, making it easy to display the mixed types of data that will need to be displayed on the historical data chart.

9.3 - Greenhouse Registration Flow

Setting up a new greenhouse should be as simple as possible for the user, with most of the details of the registration invisibly handled by the app and the backend, with a little bit of input from the sensor grid (Figure 9.3.1). The user goes through a seven-step process to set up the greenhouse, with prompts that tell them what to do during each step. As they take these steps, the frontend, the backend, and the sensor grid coordinate on the two or three steps needed to prepare for the next stage of setup. By the end, the user has experienced a simple, hassle-free setup process, and a new greenhouse is operational.

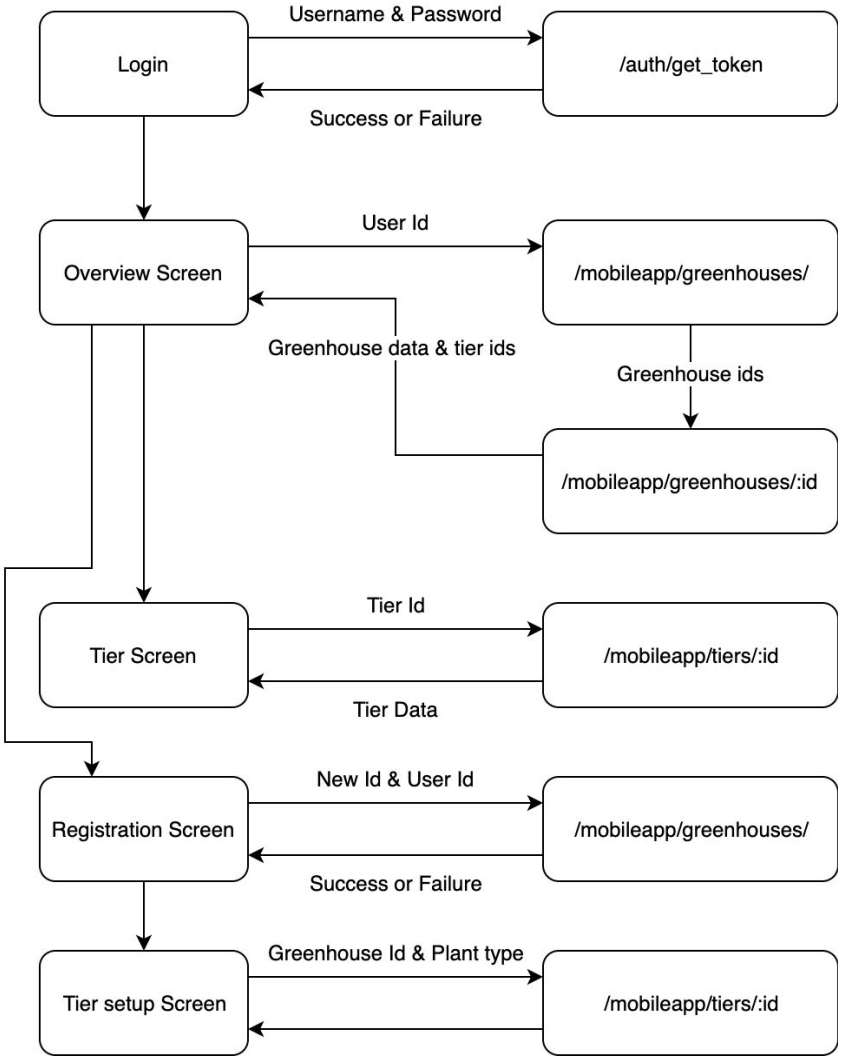
Figure 9.3.1: Data Flow Diagram for Greenhouse Registration



9.4 - Backend API Calls

Each section of the app makes very specific API calls to the backend in order to request the data it needs to display to the user (Figure 9.4.1). Some of these calls may be dependant on one another; information that was retrieved in an earlier call may be needed to make future calls. For example, the login API call might return an authentication token and a user id, which can later be used to retrieve a list of all greenhouses that belong to a user. Similarly, the greenhouse ids can be used to retrieve a list of the tiers within each greenhouse and their particular information.

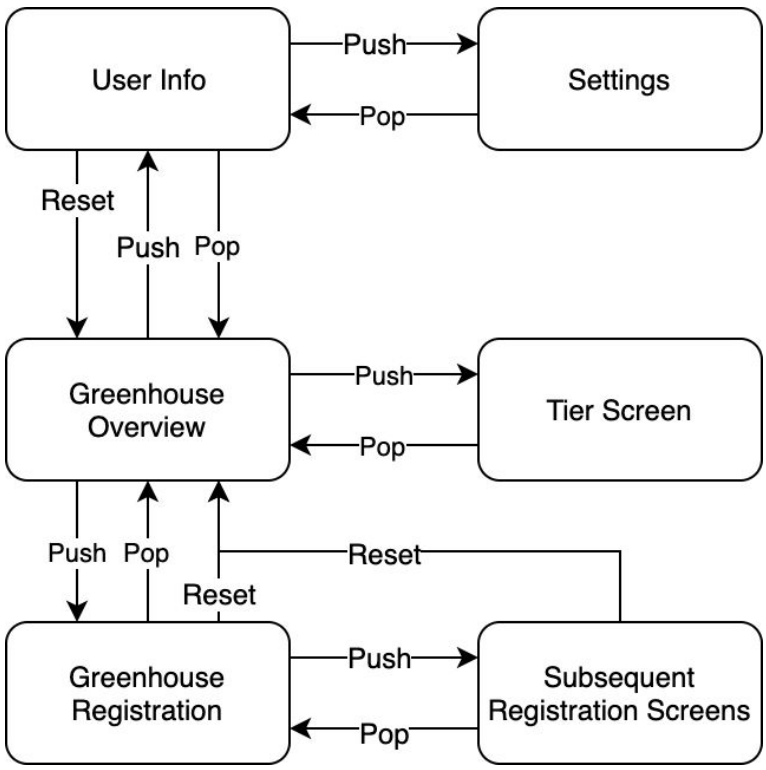
Figure 9.4.1: API Calls Overview



9.5 - Navigation Stack Overview

Navigation, written using React Navigation, is primarily stack-based, with the main navigation stack being based out of the greenhouse overview screen (Figure 9.5.1). Swiping left and right exchanges cards in the greenhouse list, without changing the navigation stack. Clicking on a tier on the greenhouse overview pushes the tier screen onto the navigation stack, and clicking the back button at the top of the tier screen pops the tier screen from the navigation stack. Clicking on the User Info button pushes the user screen to the navigation stack, and then further clicking the settings button pushes the settings screen to the stack. Pressing the back button from either the settings screen or the user info screen pops those screens from the stack, advancing back along the navigation. Clicking logout on the user info screen clears the stack entirely and navigates to the login screen. Swiping or clicking over to the greenhouse registration screen pushes it to the stack, and each consecutive screen in the greenhouse registration pushes to the stack, allowing users to go back to previous steps. Clicking on the Cancel Registration or Complete Registration buttons clears the navigation stack and returns to the greenhouse overview screen.

Figure 9.5.1: Navigation Stack



9.6 - Color Palette

When selecting colors for the app, it was important that the color scheme be bright and eye-catching, but also match the theme of growing and hydroponics. With that in mind, a green was selected as the primary color, with a pink for highlights. Blue and brown were selected as a secondary pair, to help complement the green and pink. The app color palette is shown in Figure 9.6.1, and the HEX values for the colors are shown in Table 9.6.1.

Figure 9.6.1: App Color Palette

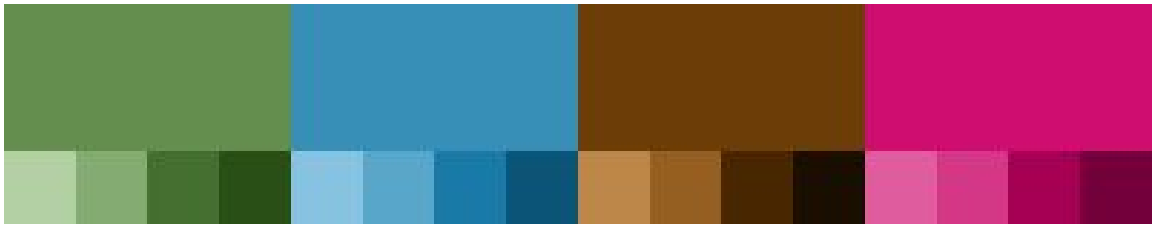


Table 9.6.1: Hex Values for colors

Green: #638E4E Green (lightest): #B2D0A4 Green (lighter): #84AB71 Green (darker): #456E31 Green (darkest): #294F16	**Blue:** #378EB7 Blue (lightest): #87C3E0 Blue (lighter): #59A6CB Blue (darker): #1979A7 Blue (darkest): #0B5477	**Brown:** #6C3D07 Brown (lightest): #BD8749 Brown (lighter): #955F22 Brown (darker): #472600 Brown (darkest): #1A0E00	**Pink:** #CF0D70 Pink (lightest): #DF5D9F Pink (lighter): #D53787 Pink (darker): #A60054 Pink (darkest): #73003A

It was also important that items drawn in these colors be visible even to those with color blindness or other vision impairments. The color palette was tested for contrast under protanomaly (1% of men), deuteranomaly (5% of men, 1% of women), and tritanomaly. It was also tested under protanopia, deuteranopia, tritanopia, dyschromatopsia, and full achromatopsia. When it failed the contrast measures for one of these vision impairments, it was adjusted and retested until it passed.

Care was also taken to ensure that the color contrast would be up to WCAG standards for color contrast. Each possible combination of colors was assigned a contrast score based on the WCAG guidelines (Figure 9.6.2), and the corresponding charts were filtered based on general contrast (2.0 contrast minimum, Figure 9.6.3), minimal contrast for large elements (3.0 contrast minimum, Figure 9.6.4), and minimal contrast for small text (4.5 contrast minimum, Figure 9.6.5). Black and white were added for comparison.

Figure 9.6.2: Full Chart of Color Contrast Values

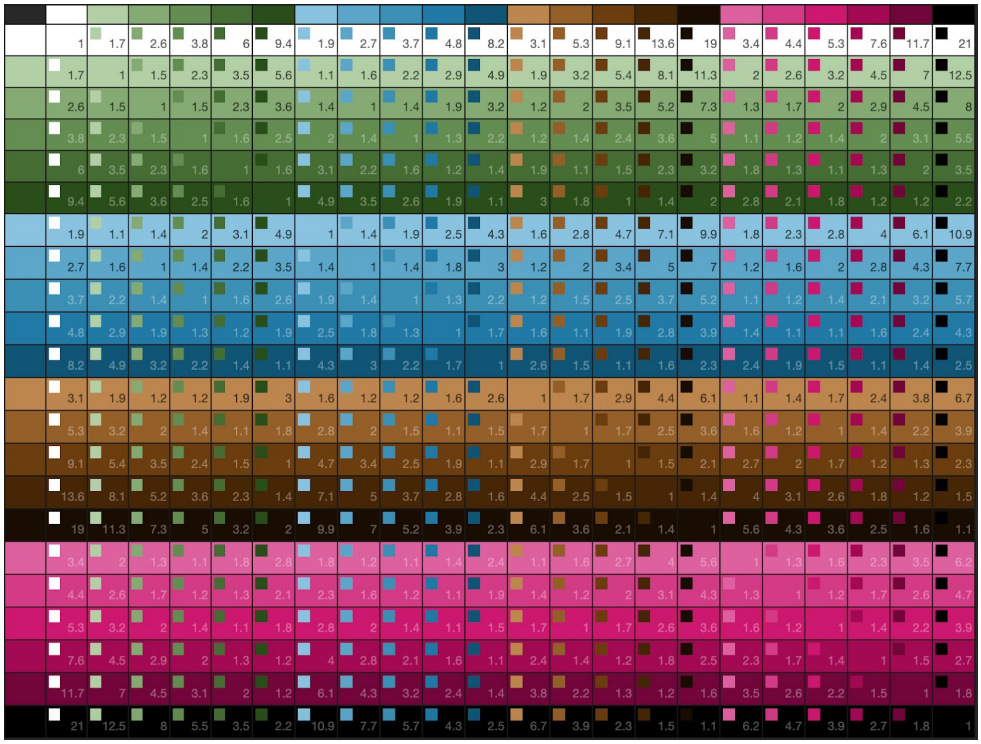


Figure 9.6.3: Color Contrasts 2.0 and Higher

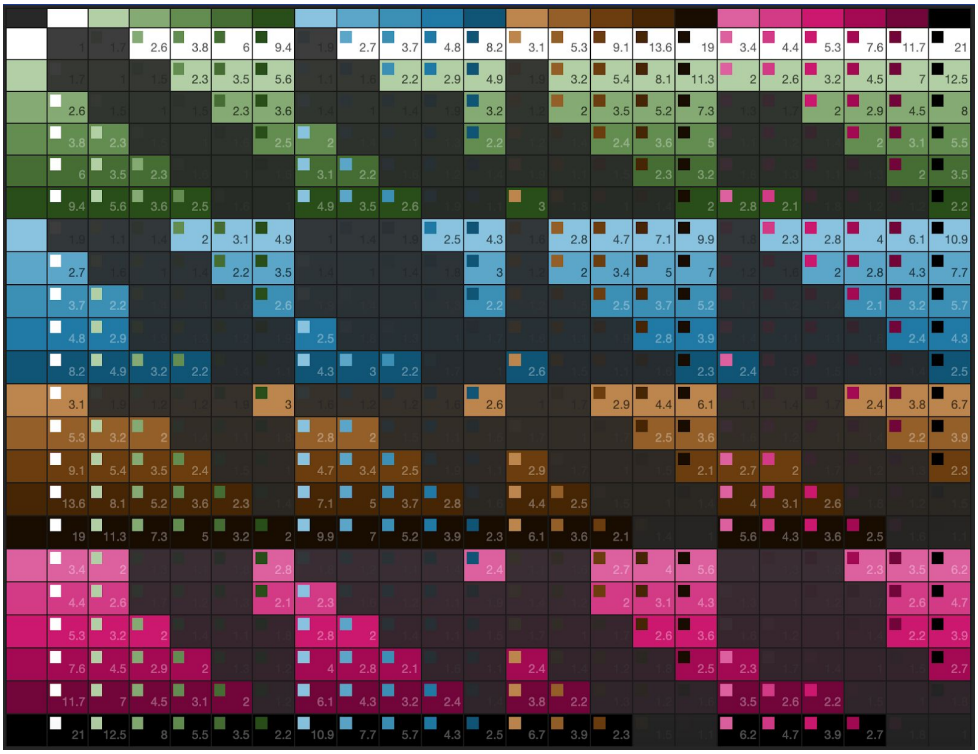


Figure 9.6.4: Color contrasts 3.0 and Higher

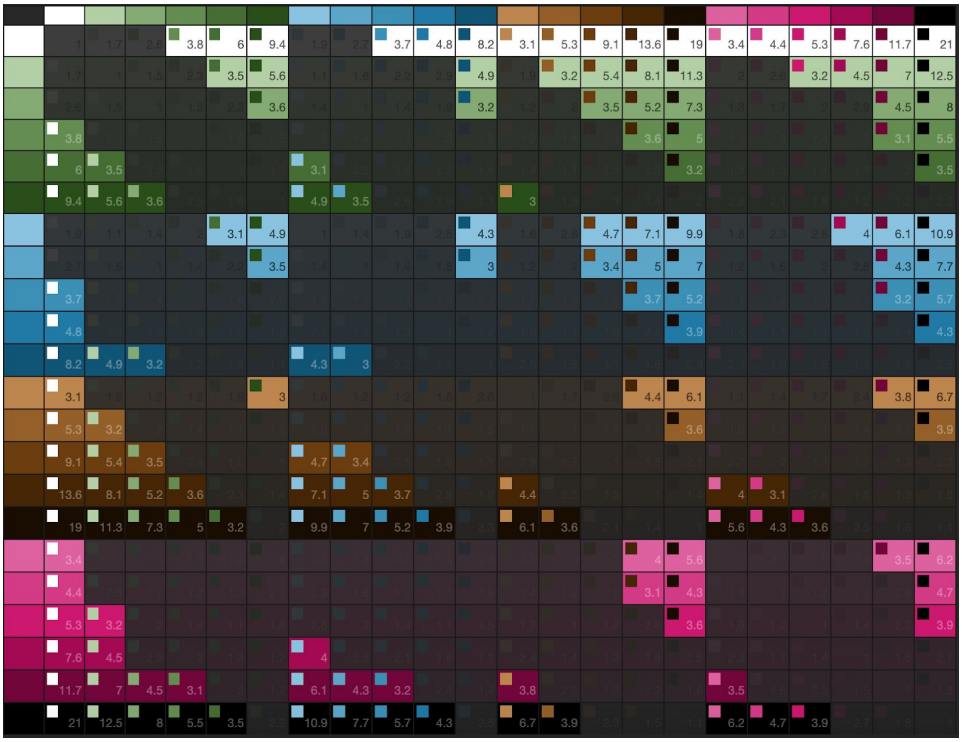
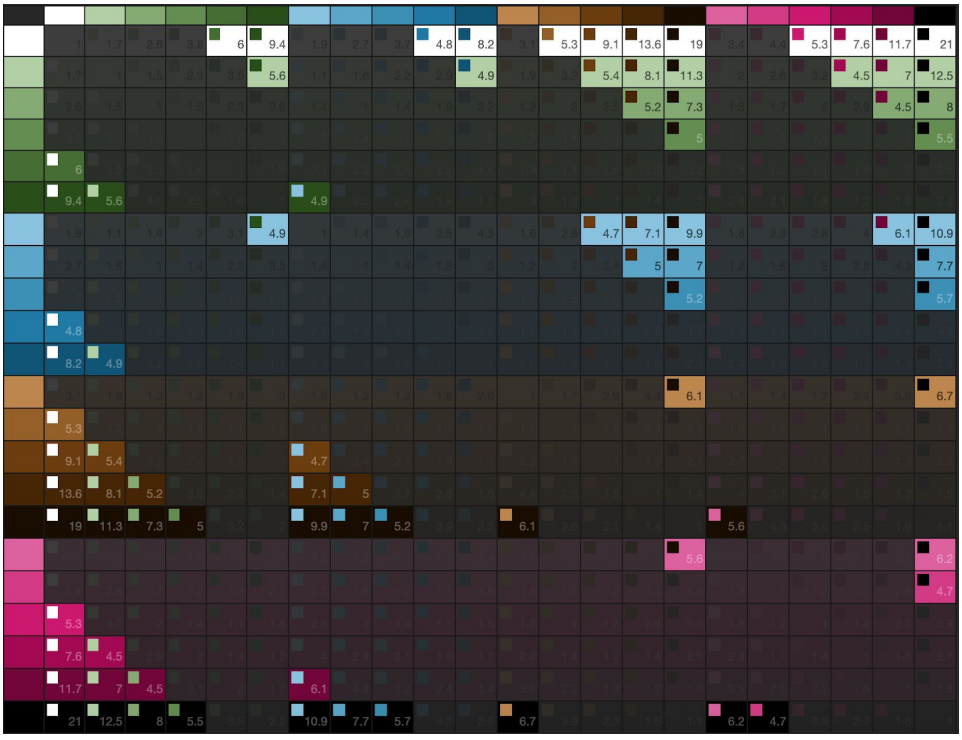


Figure 9.6.5: Color Contrasts 4.5 and Higher



9.7 - Image Assets

When designing the image assets for the app, simplified representations of plants, greenhouses, and actions were needed. In order to make sure that the images would look good on all screen sizes, the app would need either .png images in a variety of resolutions, or vector images. Since the images would be simplified any way, it was decided that vector graphics would be the best choice. Using the color scheme (plus a few additional colors, when needed), vector assets were drawn up for the greenhouse tiers (Figure 9.7.1), the four different plants: tomatoes, green beans, turnips, and spinach (Figure 9.7.2), the power levels (Figure 9.7.3), the water levels (Figure 9.7.4), the nutrient levels (Figure 9.7.5), saplings (Figure 9.7.6, left side), and for the app icon (Figure 9.7.6, right side). These assets will be reused throughout the app, to provide a consistent UI experience for all users.

Figure 9.7.1: Greenhouse Image Assets (shown over dark background)

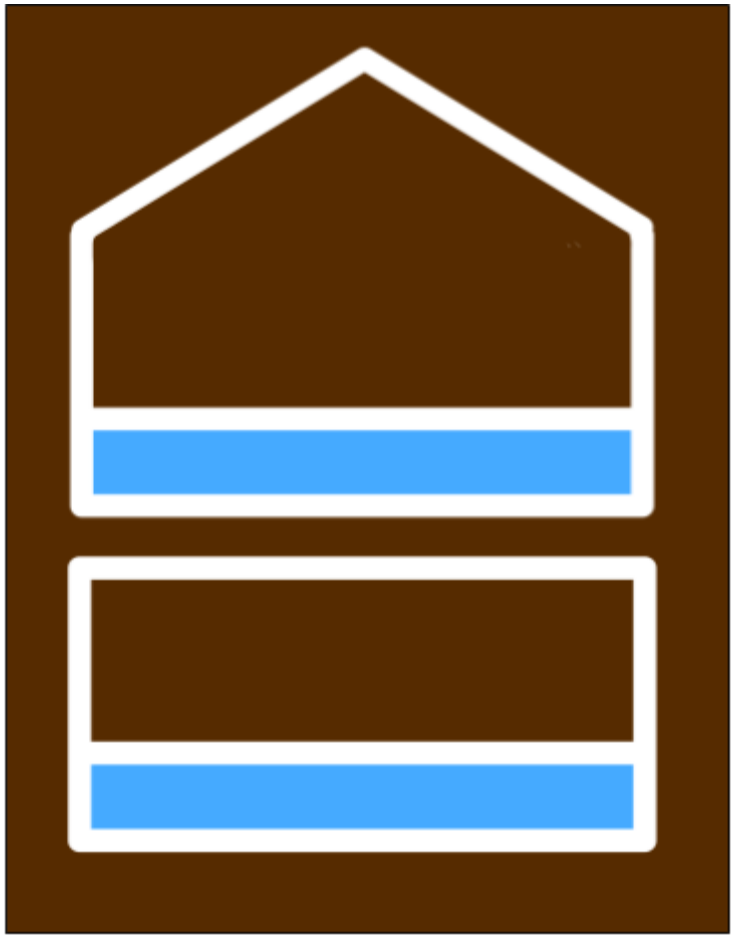


Figure 9.7.2: Plant Image Assets

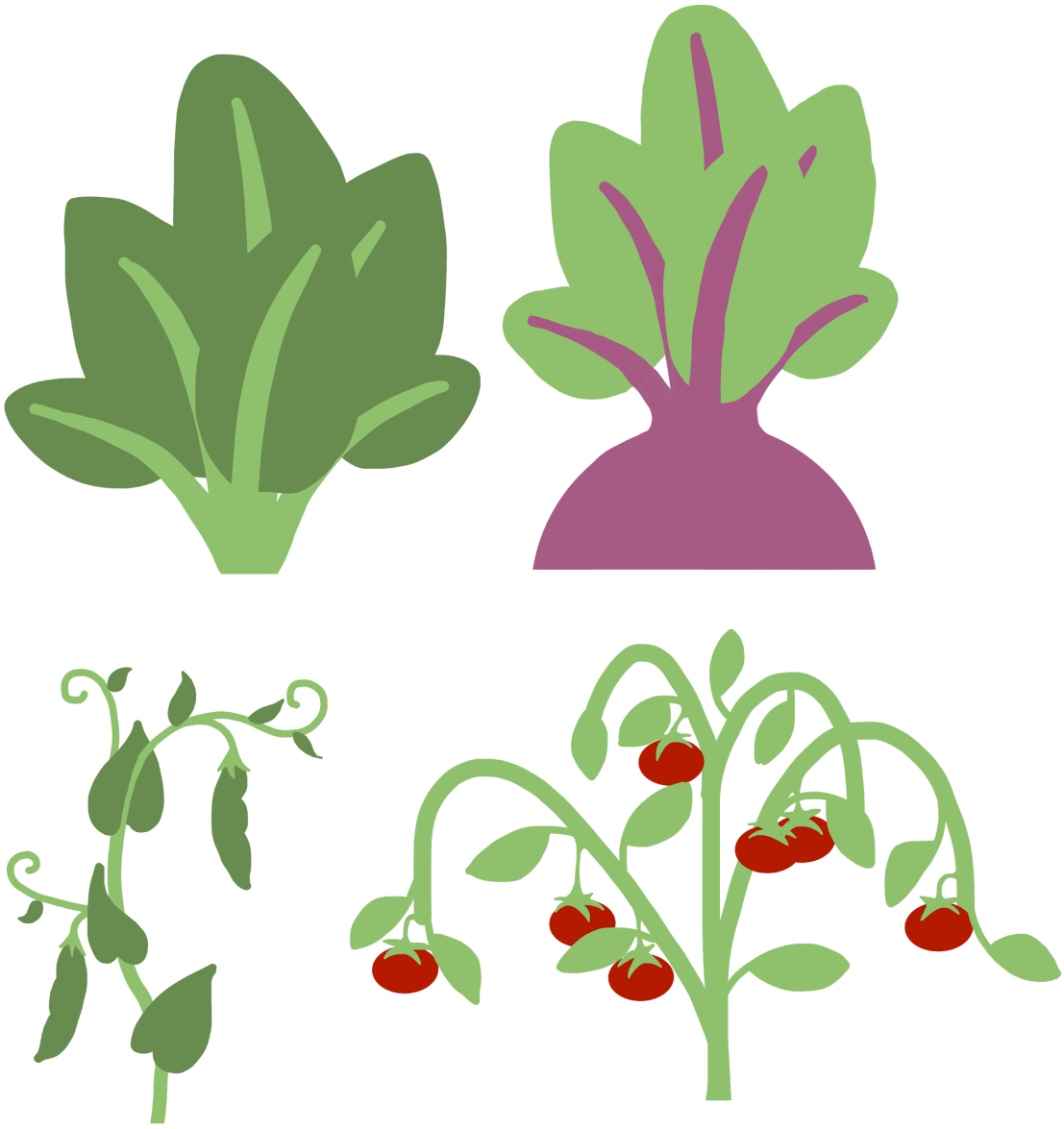


Figure 9.7.3: Battery Level Assets

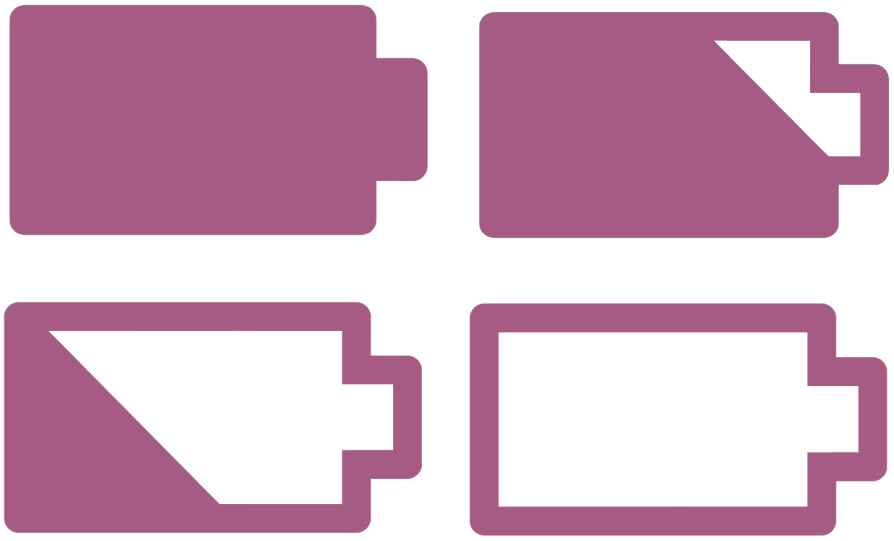


Figure 9.7.4: Water Level Assets

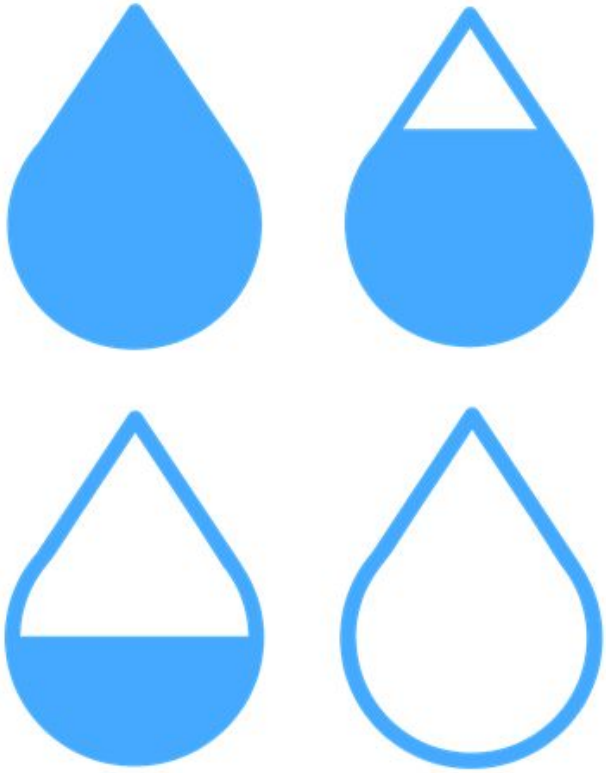
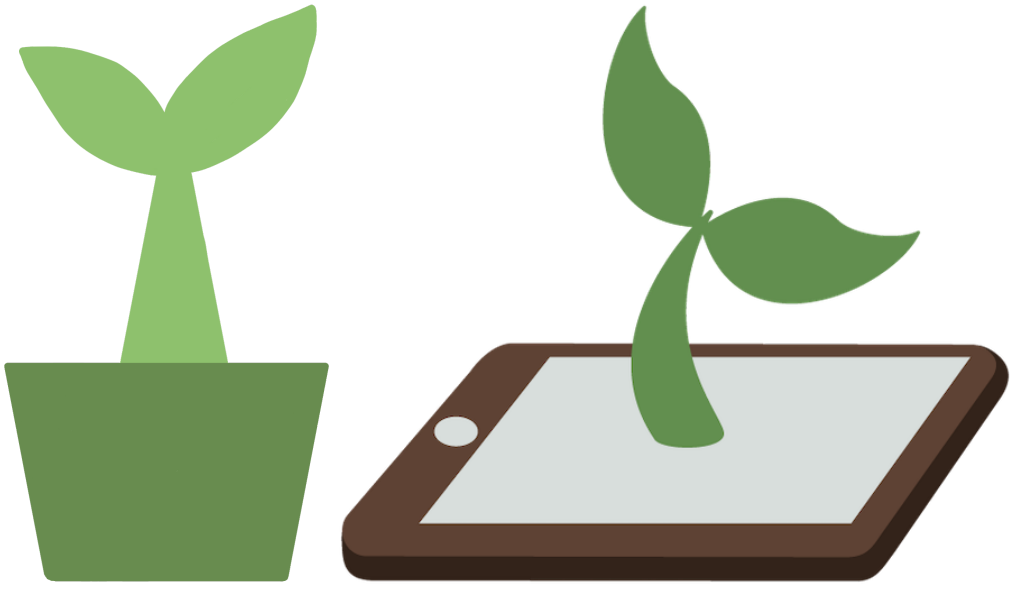


Figure 9.7.5: Nutrient Level Assets

Figure 9.7.6: Miscellaneous App Assets: Sapling and App Icon



9.8 - Screen Breakdown

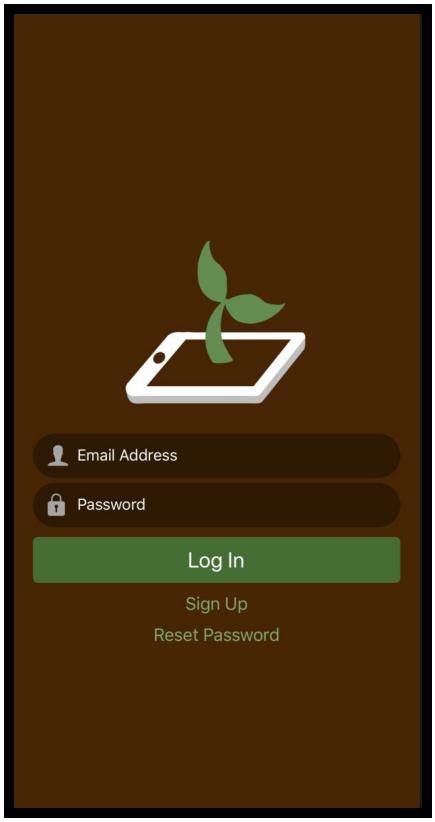
There are 15 major screens that the user navigates through within the mobile app. Each has undergone a detailed design process to ensure that they are easy for users to navigate and that their design looks professional.

Login Screen

The very first time a user ever downloads and opens our app, they will be directed to the login screen. A mock-up of the login screen is shown in Figure 9.8.1. The user will input their username and password, and as long as the app remains downloaded on their device and they do not log out of their account, they will remain logged in on that device. We have hopes of integrating a Facebook login option so that user information will be even more secure and it would be more convenient for users.

We have integrated the “ios-person” and “ios-lock” icons from the React Native library to appear to the left of the text entries. We have also taken the initiative to use secure text entry for the password field of the login screen.

Figure 9.8.1: Login Screen

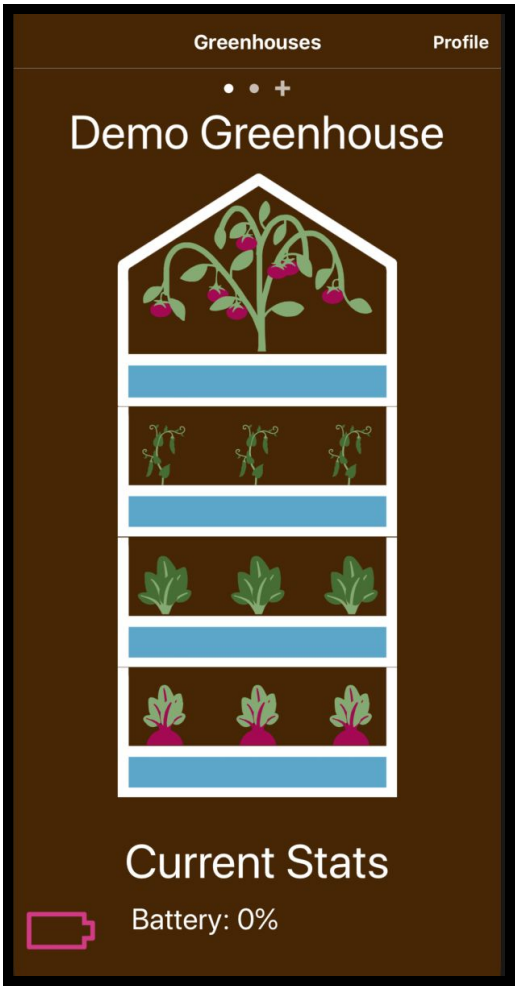


Greenhouse Overview

The ‘Greenhouse Overview’ screen can be thought of as our ‘home’ screen (Figure 9.8.2). Upon opening the app, any returning user who has a registered greenhouse and is logged in to their account, will be automatically directed to this page. This page is extraordinarily important because it allows the user to navigate to the ‘Single Tier View’ by clicking on the tier they would like to access.

Alternatively, any user who does not have a registered greenhouse will automatically be directed to the ‘register a new greenhouse’ page, since they do not have a greenhouse to view. Once a new greenhouse is finalized, users are automatically redirected to the ‘Greenhouse Overview’ screen of the greenhouse they just created.

Figure 9.8.2: Tier Overview Screen

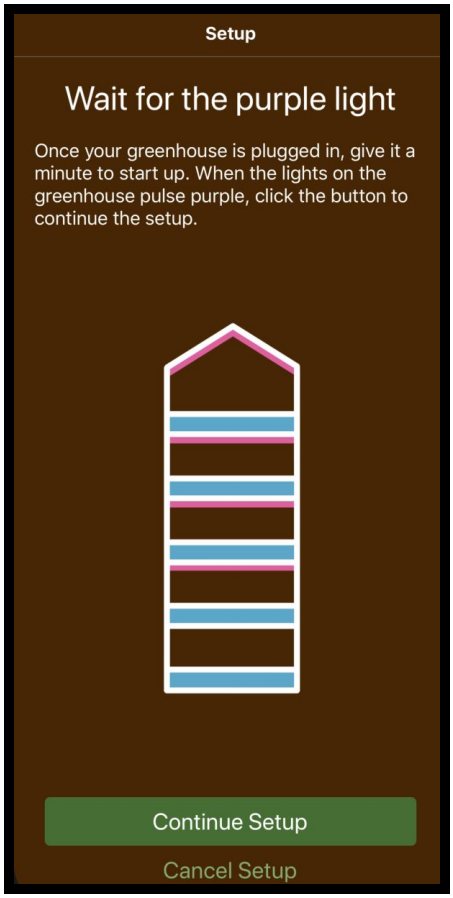


Register a New Greenhouse - Purple Light

As previously stated, users who do not have a greenhouse associated with their account will be automatically directed to the ‘Register a New Greenhouse’ page (Figure 9.8.3). This page is the first of many in the registration process of a new greenhouse. This first page advises users to wait for the purple light to appear and then press continue to move on to the next step. Users also have the option to cancel the setup at any time throughout the process.

Users are able to have multiple greenhouses associated with their account. To navigate between greenhouses, users may swipe right/left and they will see the card view of each individual greenhouse. There are symbols at the bottom of the screen that denote the number of greenhouses associated with their account and which greenhouse they are currently viewing. If a user swipes all the way to the end of their greenhouses, they can swipe right once more to land on the ‘Register a New Greenhouse’ page. This allows them to add another greenhouse to their account.

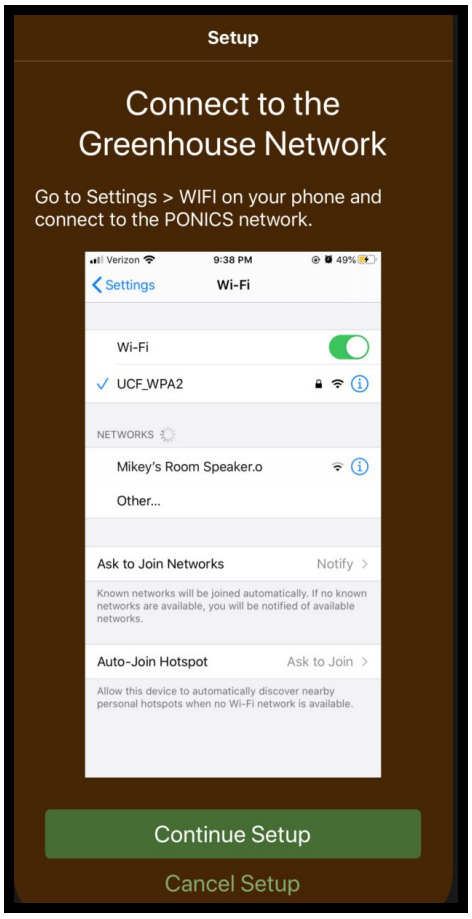
Figure 9.8.3: Register a New Greenhouse - Purple Light Screen



Register a New Greenhouse - Network

Once they have selected the continue button on the previous page, users come to the Network registration screen (Figure 9.8.4). The network is created by each greenhouse to send and receive information before it connects to the server. Once the phone has connected to the greenhouse network, the registration process automatically continues to the next page, with no required input whatsoever from the user. This is yet another feature we have incorporated to make the registration process as simplistic as possible. We were also very intentional in adding a descriptive image to this page, as we recognize that not all of our users will be familiar with LAN networks or know where to look for them. This way, we include any technological skill level and age range in our potential customer pool. Users also have the option to cancel the setup at any time throughout the process by pressing the button at the very of the screen.

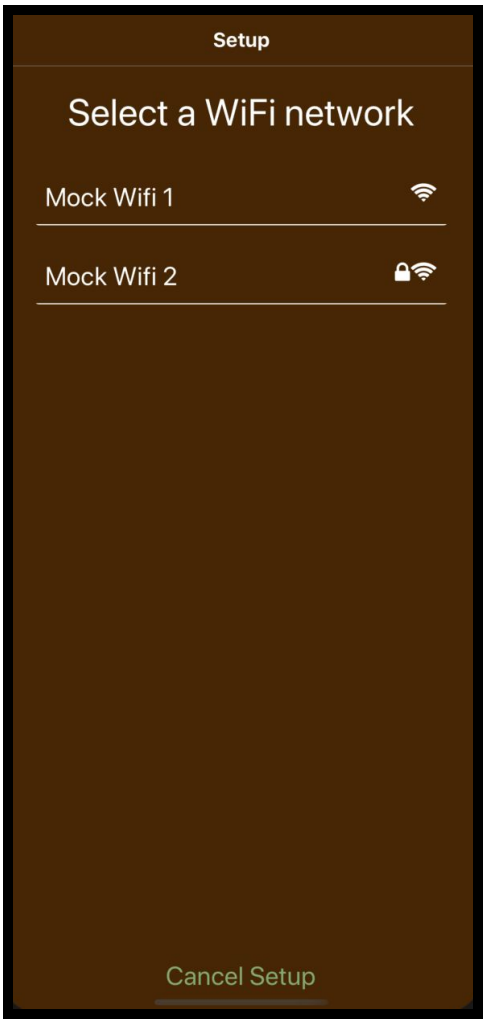
Figure 9.8.4: Register a New Greenhouse - Network Screen



Register a New Greenhouse - Wifi

The page shown in Figure 9.8.5 is probably the most simplistic, because users are expected to merely join a wifi network of their choosing. However, this step is crucial for the proper functioning of our hydroponics system. An internet connection is required for our backend system to be able to communicate with our frontend display. Without wifi, Pocket 'Ponics would not be able to monitor or make changes to the greenhouse in any way, rendering it useless.Users also have the option to cancel the setup at any time throughout the process by pressing the button at the very of the screen.

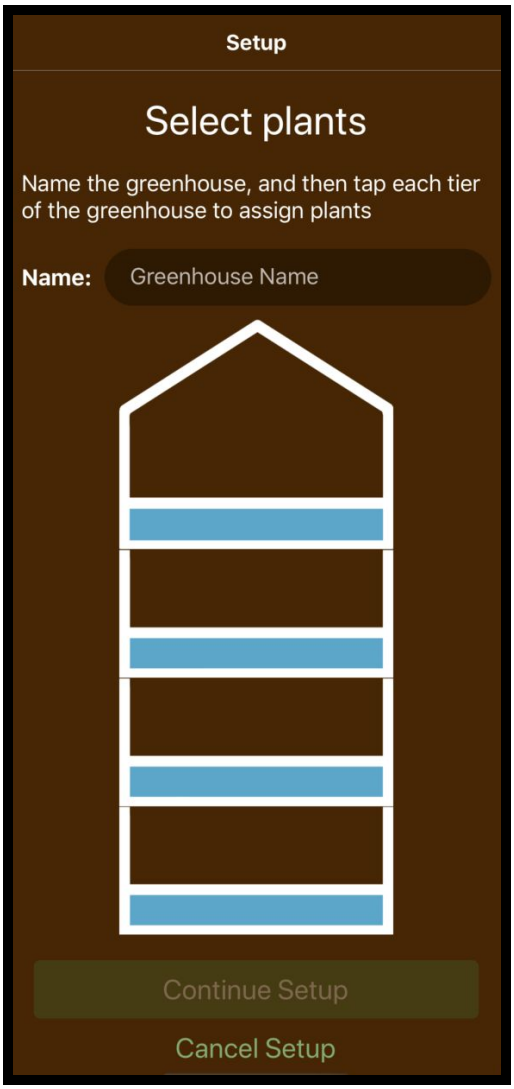
Figure 9.8.5: Register a New Greenhouse - Wifi Screen



Register a New Greenhouse - Tier Selection

Intuitively, this page directs users to a blank greenhouse screen where they are instructed to tap on a single tier in order to select which plant they'd like to grow on that specific tier (Figure 9.8.6). Once the user selects a tier, a popup window comes up displaying which plants the user can choose from (described in more detail on the following page). Once a plant is selected, the popup screen diminishes and the selected plant appears on the tier. This process continues until the user selects continue. It is important to note that the user is not required to fill every single tier of the greenhouse if they do not wish to, and they can reassign a tier by simply tapping on it again. Users are also able to name their greenhouse at this point.

Figure 9.8.6: Register a New Greenhouse - Tier Selection Screen



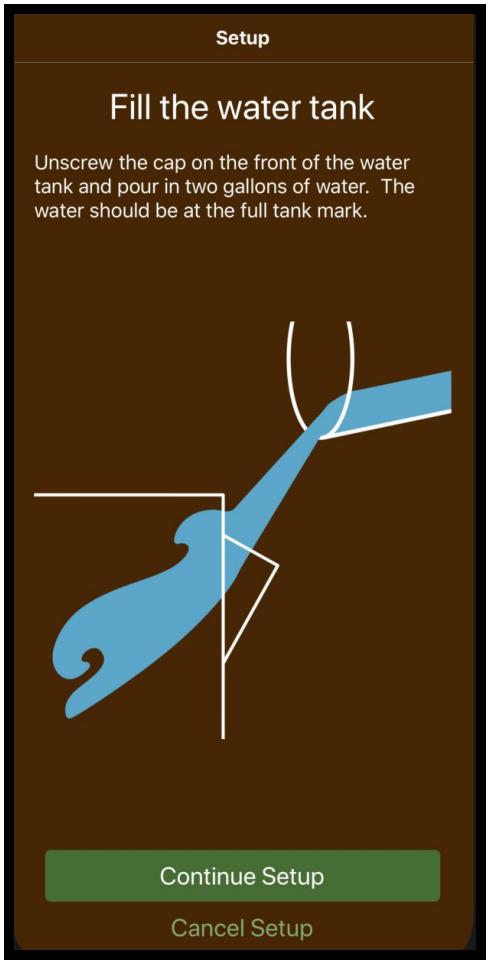
This step in the registration process is probably the most fun, because users are finally able to select which plants they'd like to grow in their hydroponics system. As mentioned above, once the user has selected a tier, this pop-up screen rises from the bottom of the page (Figure 9.8.7). There are dark brown circles which contain small images of the plant options along with the name of the plant directly beneath it. Once a plant option is selected, images of the plant will be auto generated into that tier of the greenhouse.

Figure 9.8.7: Register a New Greenhouse - Tier Selection with popup

Register a New Greenhouse - Fill Water

Now that the user has determined which plants they would like to grow, it's time to get to work! The first step is to fill the system with water. Users are instructed to pour water into each tier of the hydroponics system until the water level reaches the designated marking (Figure 9.8.8). This is the only time that users will ever fill the individual tiers themselves, because from this point on, the water levels will be automatically monitored and replenished. We've also incorporated a "water fill" animation to play on the screen as the user fills the water tanks.

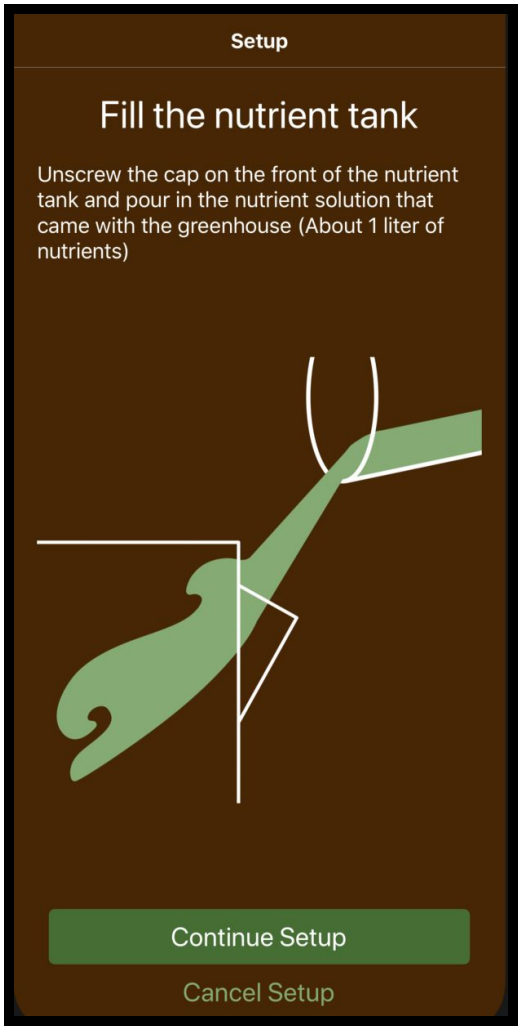
Figure 9.8.8: Register a New Greenhouse - Fill Water Screen



Register a New Greenhouse - Fill Nutrients

After the user has properly filled each tier of the greenhouse and the storage tank with water and pressed the continue button on the screen, they will be directed to the fill nutrients page (Figure 9.8.9). We designed all of these screens to be extremely self explanatory and simple, as it is vital to our business model that absolutely anyone be able to use our product with ease. Similarly to the previous page, we have incorporated a fill nutrient animation that will play on the screen while the user is supplying nutrients.

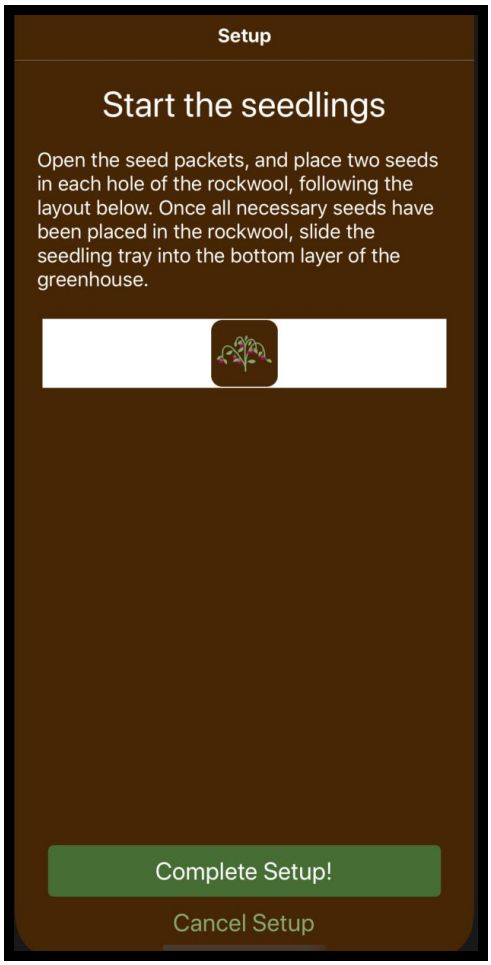
Figure 9.8.9: Register a New Greenhouse - Fill Nutrients Screen



Register a New Greenhouse - Seedlings

Planting the seedlings is the third and final physical step needed in order to get a greenhouse up and running; it's also the very last step necessary before your new greenhouse is fully functional! With consistency in mind, we decided to add images to this screen as well (Figure 9.8.10). The images show the user what seedling they should be starting and the order in which they appear in the greenhouse. In this particular figure, the user chose only to grow one tier of tomatoes, that is why only one image is being displayed.

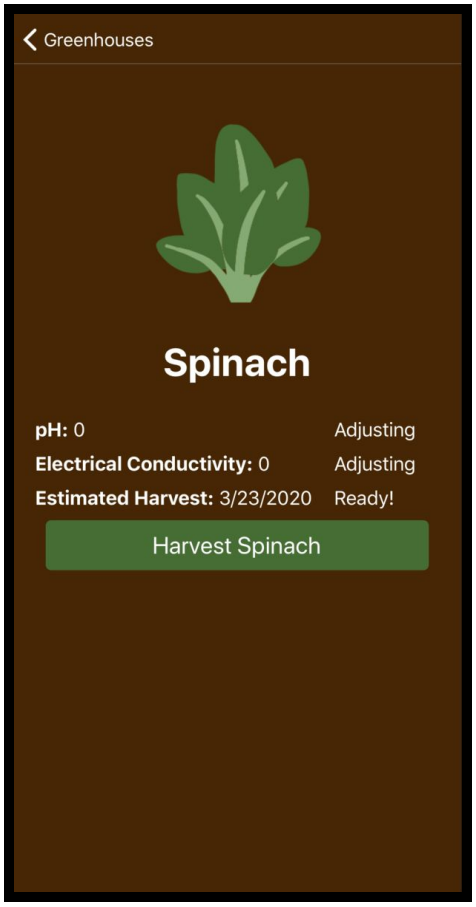
Figure 9.8.10: Register a New Greenhouse - Seedlings Screen



Tier View

The screen shown in Figure 9.8.11 allows the user to take a more indepth look at a specific tier of their greenhouse. This screen is extremely important because it provides crucial information on the health and wellbeing of each plant. This information can only be found on this specific page, as it does not appear anywhere else throughout the app. This information includes: overall health, water level, temperature/light supply and an estimated harvest date. There is also a button at the bottom of the page that allows the user to designate that they have harvested the plant. This clears the plant from the greenhouse overview screen and allows the user to add a new plant in its place.

Figure 9.8.11: Tier View Screen

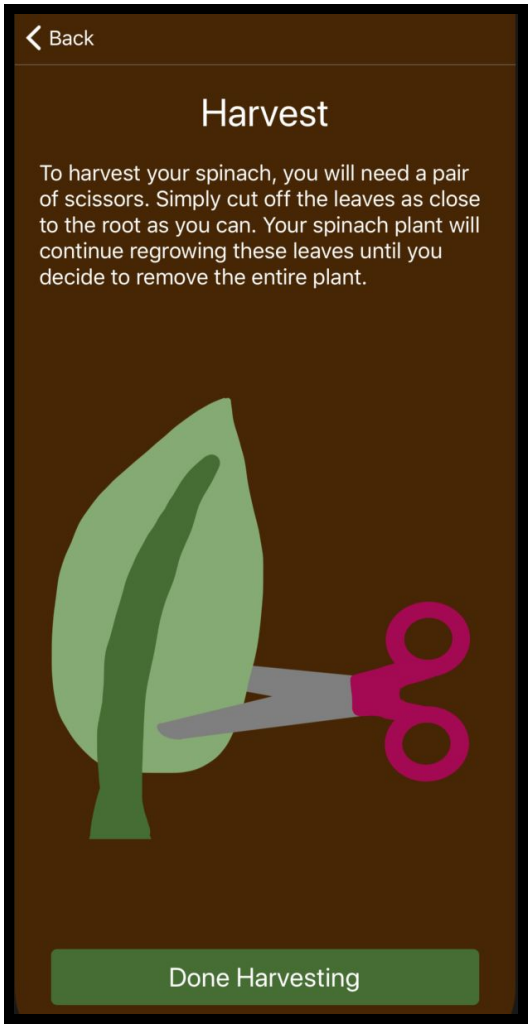


Harvest

Once the user's plant has ripened and they have selected the Harvest button on the previous screen, they are directed to the page shown in Figure 9.8.12. There are two main purposes that this screen serves:

1. To ensure that the user meant to press the harvest button, and didn't select it by mistake. This would cause huge issues if the plant was automatically removed from the greenhouse mistakenly because that information cannot be restored. Hence the cancel button at the bottom of the screen.
2. As some of our plant options require a more strenuous harvesting process, we want to be fully confident that our users have all the right information before removing their plants from the greenhouse.

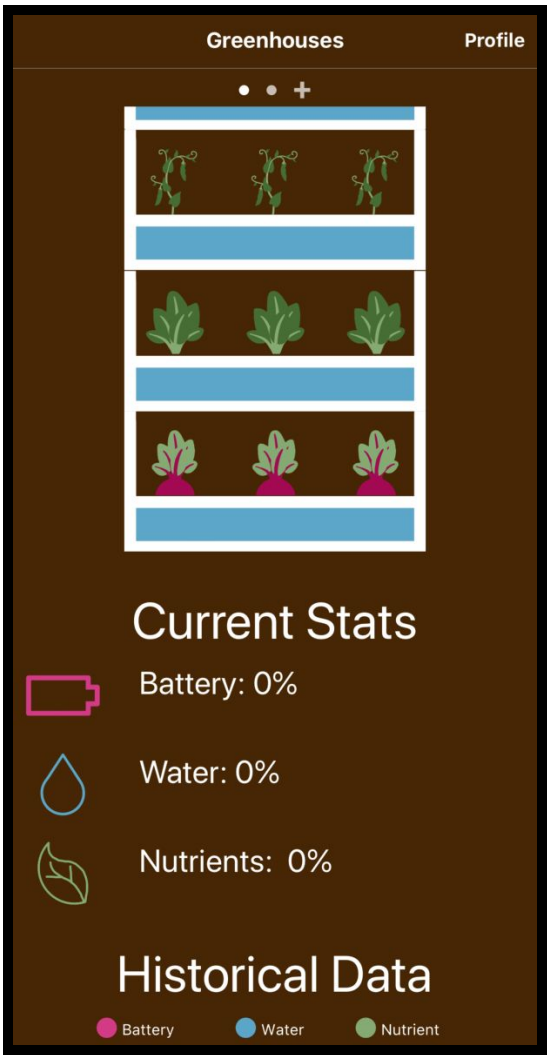
Figure 9.8.12: Harvest Screen



Ponic Stats

The Ponic Stats page (Figure 9.8.13) allows users to easily and accurately see the levels of all their greenhouse resources in one easily accessible place. As displayed in the image below, the resource levels that are provided are battery level, water level and nutrient levels.

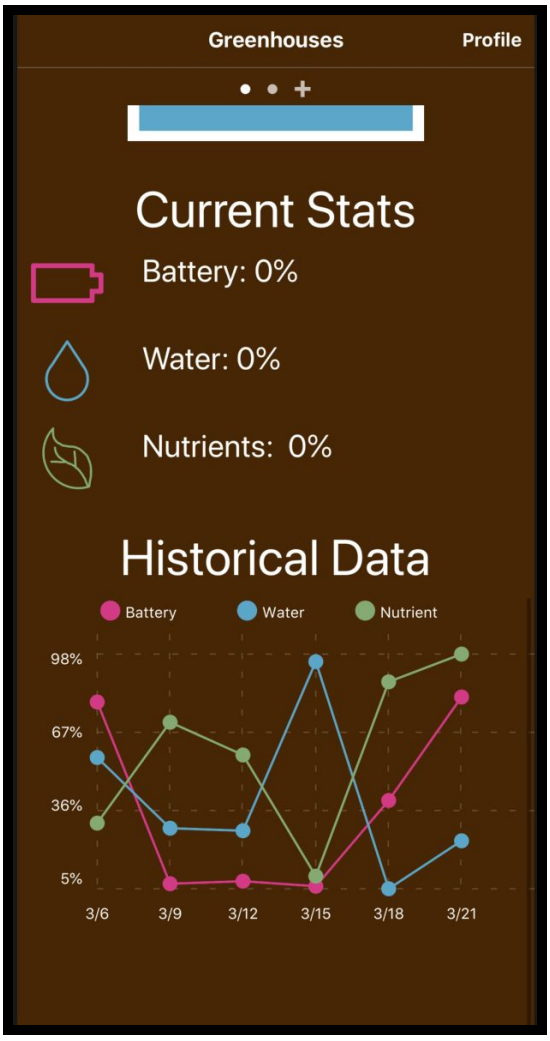
Figure 9.8.13: Ponic Stats Screen



Historical Data

The screen shown in Figure 9.8.14 provides an easily accessible and understandable overview of your greenhouse water and nutrients levels over the past 30 days. We have included this feature so that users are able to gauge how often they need to replenish these resources. Measurements are collected automatically every 72 hours. The nutrient levels are represented by the green line, the battery levels are represented by the pink line and the water levels are represented by the blue line. The x-axis represents time (on a three-day scale) and the y-axis represents percentage remaining.

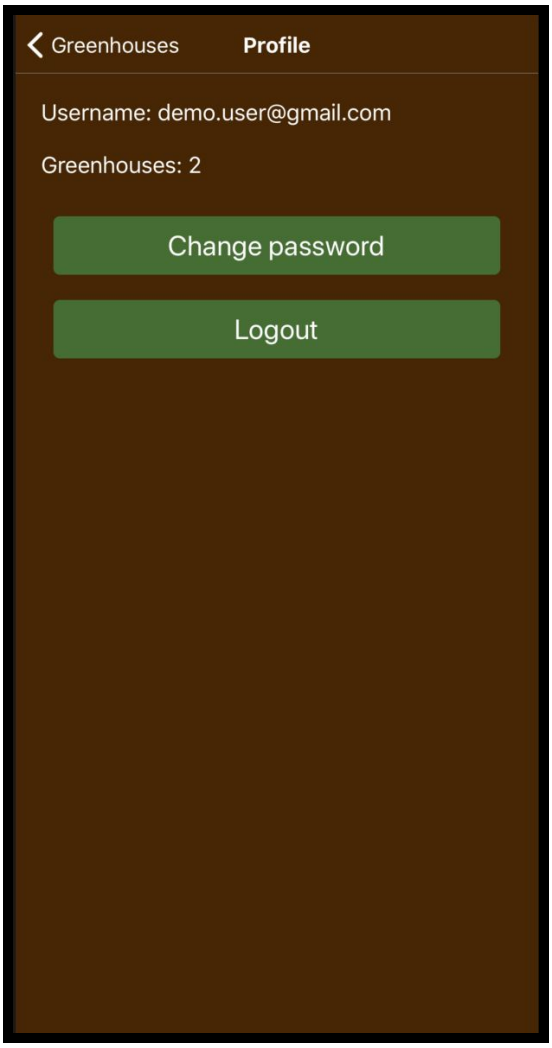
Figure 9.8.14: Historical Data Screen



User Profile

If the user icon in the upper right hand corner of the screen is selected at any point, the page that is displayed is that shown in Figure 9.8.15. This page displays user information such as the number of greenhouses currently registered to the user and the user's username. Our user profile page also allows users to change their password or logout of the app. The back arrow at the top left hand side of the screen allows users to return to whatever page they were on before they selected the user icon.

Figure 9.8.15: User Profile Screen



9.9 - Pocket 'Ponics Website

The Pocket 'Ponics GitHub was initially established on April 8 in anticipation of the upcoming project. On November 16, version 1 of our Pocket 'Ponics website was designed and created by our dedicated Project Leader, Elli Howard. Elli was also able to secure the PocketPonics.com domain name, and that is where version 1 of our website can currently be found. We plan to roll out version 2 of our website onto the pocketponics.com domain name as soon as possible. For the time being, we are hosting it locally.

Our initial vision in creating PocketPonics.com was solely to serve as an informational landing page for potential customers looking to gain insight into our company and/or our mission. However, as time went on and we received more feedback on our product, we decided to change that. Version 2 of our website still contains all the same information for the general public, but with the added bonus of an admin portal feature.

I will continue the rest of this section speaking as if version 2 of the website was currently being hosted on pocketponics.com. When someone navigates to PocketPonics.com, the first thing they will see is our homepage (Figure 9.9.1). On this screen, we feature several different subsections meant to educate potential customers on what it is we do. Therefore, it's only fitting that our very first section is titled: 'What is Pocket 'Ponics'. In this section of the homescreen we have a brief description of our product and what makes us unique. The next section is titled 'Why we built Pocket 'Ponics', where we give the viewer a very general reasoning as to what our motivation was in building this solution. 'What can you grow with Pocket 'Ponics' is the very next subsection on the homepage of our website. After speaking with several test subjects, we found that this is one of the first questions people usually wonder after establishing what our product is, so we decided it'd be in our best interest to feature this information on the home screen of our website.

Some other aspects of our website include an 'About' page and a 'Contact' page (Figure 9.9.2 and Figure 9.9.3). The 'About' page is broken down into six subsections, one featuring each of our designers. Each of these six subsections contains a brief description of our designers, including their role in the project, what makes them so passionate about Pocket 'Ponics, etc.

We plan on integrating a customer review page, where prospective users can see how our current customers are liking our product. Another idea we have is to incorporate more of a 'community' essence into our product by adding a blog feature onto the website. This way, community members can be as involved or uninvolved as they'd like to be, and it'd be a great way for us to gauge customer satisfaction and desired product updates. All in all, building a community of Pocket 'Ponics customers has many benefits and virtually no drawbacks. We

thought a website would be the perfect way to foster this relationship among users, and we're eager to see where it will take us as a company.

Another feature we're optimistic about integrating into our website design is a Question & Answer page. While the previously mentioned blog feature will provide users a great system for receiving suggestions and tips from fellow Pocket 'Ponics users, the Question & Answer page will be our way providing similar information from a company standpoint. We feel both of these avenues are beneficial to our product because it is important that our customers feel as though purchasing a Pocket 'Ponics system is more than just growing plants, it's joining a community of activists. We want Pocket 'Ponics to immediately be associated with comradery any time consumers hear our name.

In order to access the admin portal aspect of our website, the user must navigate to pocketponics.com/adminlogin (Figure 9.9.4) and enter the correct credentials to gain access. If the user enters incorrect credentials, a popup window will be displayed on the screen that reads "incorrect username or password". Also, if an unauthenticated user tries to access any private pages, such as pocketponics.com/adminhome, the user will be redirected to this page.

Once an administrator has successfully logged into the admin portal, they will be redirected to pocketponics.com/adminhome (Figure 9.9.5). On this page, administrators will see a list of every plant that is currently in the database. Administrators have the ability to add new plants to the database by clicking the "add" button at the bottom of the plant list. They can edit or delete any of plants currently in the database by clicking on the plant's name.

When an administrator selects the "add" button, they are redirected to pocketponics.com/admin (Figure 9.9.6). As you can see, this page houses a blank form with all of the information needed to add a new plant to the database. The admin simply fills in all of the fields and presses submit. They are then redirected back to the admin home screen and the plant list is updated with their newly added plant.

From the admin home page, if an administrator chooses to select a plant, they are redirected to pocketponics.com/adminplant (Figure 9.9.7). From this page, administrators are able to update any of the plant's values, delete the plant, or press cancel and return to the admin home page.

Figure 9.9.1: Website Homepage

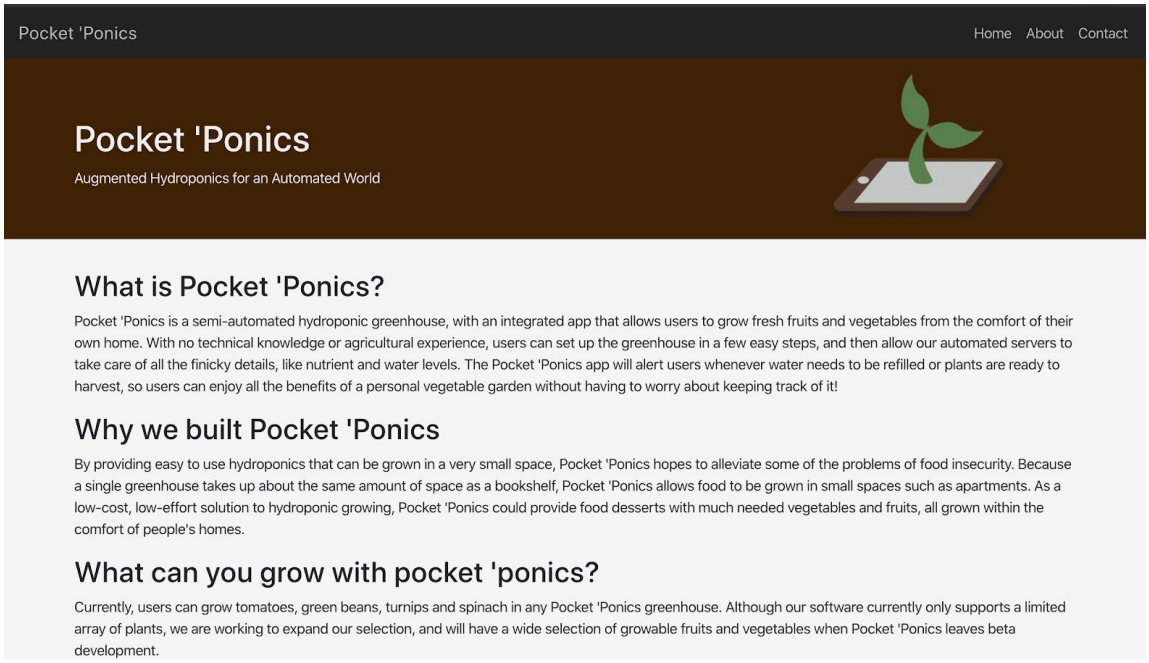


Figure 9.9.2: About Page

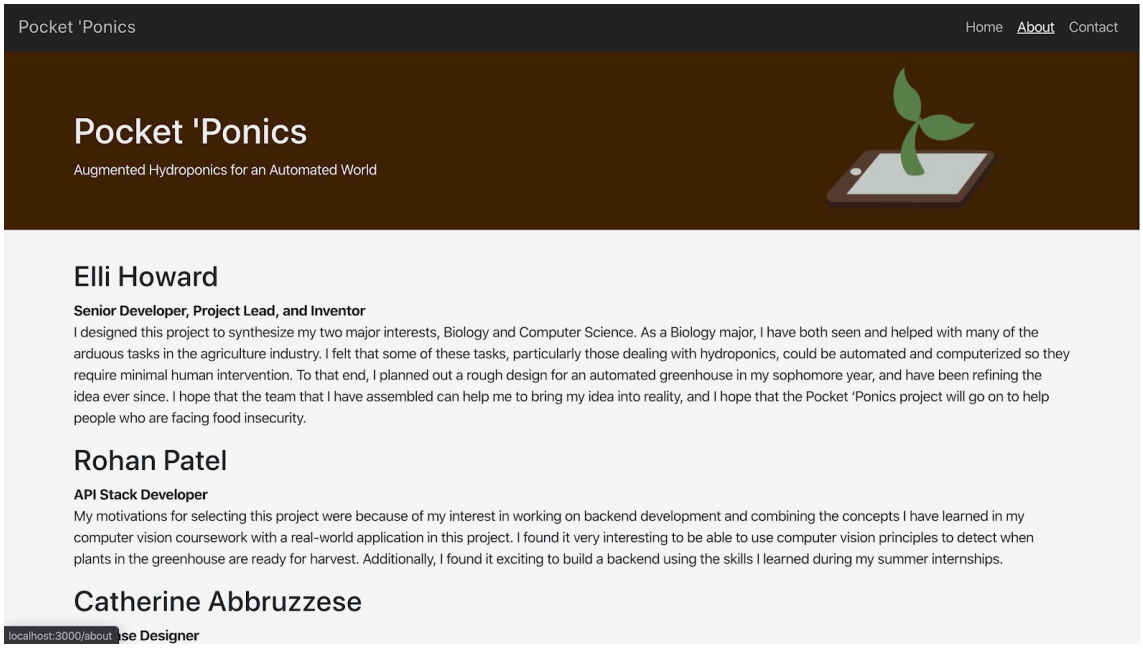


Figure 9.9.3: Contact Page

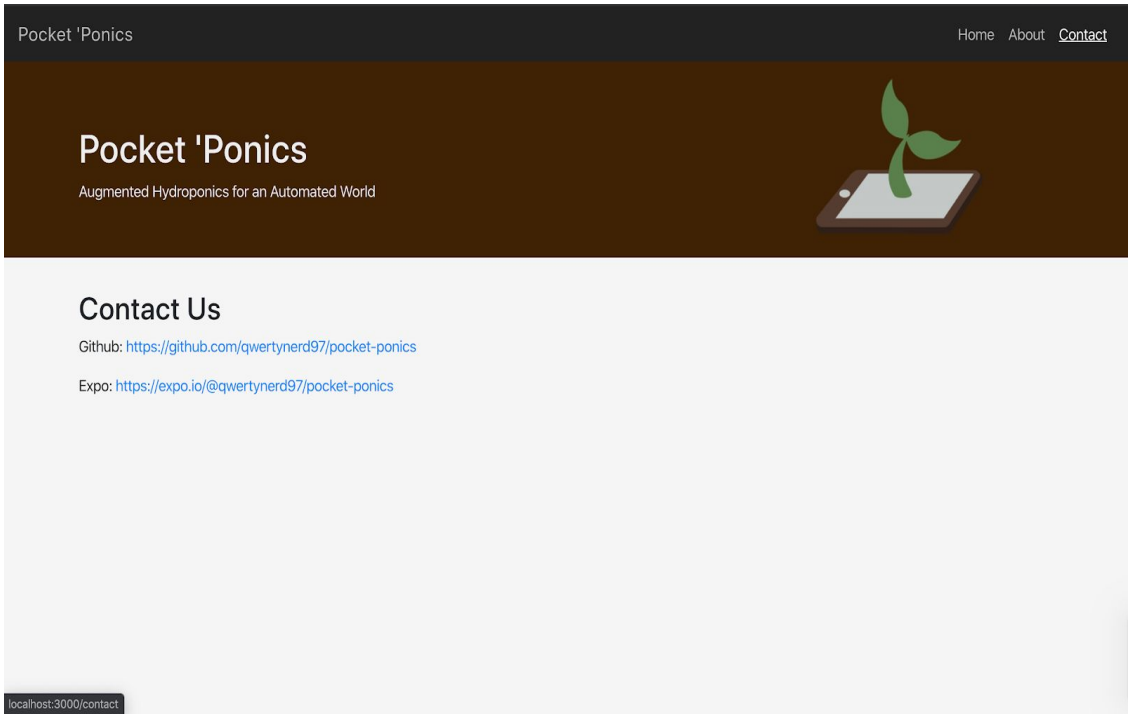
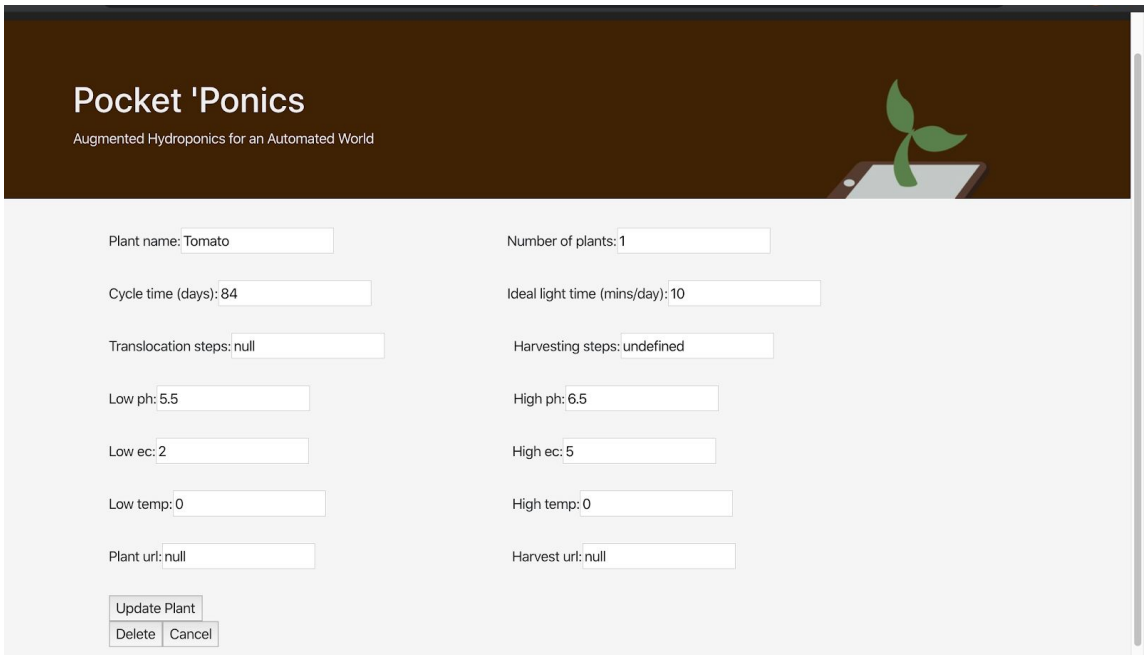


Figure 9.9.6: Admin Plant Page



9.10 - Unit Tests

Unit tests for the front end will be written in Jest, using React-Test-Renderer to help test the React Native elements. Jest allows programmers to mock out any modules that the code is dependent upon, which helps increase the granularity of the unit tests, and ensure that all code is working exactly as intended. Jest also allows tests to be written as snapshots, which means that tests can be quickly written up, and their output saved to compare to the next run. This is particularly helpful with frontend testing, as it allows for quick identification of small changes in the UI.

When writing unit tests with Jest, there is an additional option to check coverage. This keeps a record of which lines of code have been run by the unit tests, and what percentage of the entire module, package, and project has been run. Although 100% coverage does not guarantee that the code is bug-free, high coverage percentages can help reduce the number of bugs that make it to production. As such, the front end will have 100% function, branch, and statement coverage.

Jest unit tests will be run in several different places throughout the workflow of frontend development. First, they will be run as a pre-commit hook, ensuring that any commits will pass all the tests and maintain 100% coverage. Second, they will be run as a continuous integration on Github pull requests, to make sure that no code is added to the development branch of the project without passing all of the tests. Finally, they will also be run before compiling any version of the app for production. By running the tests in three different places, it will minimize the bugs we introduce, and it will help to prevent regression as updates occur.

As far as the file structure goes, each code file in the app will have a corresponding jest test, named “filename.test.js”, that is placed in the same folder as the code file. Since Jest can easily collect all tests from files named *.test.js, there is no need to have a separate test file structure, and having the tests next to the code files will help to ensure that every file is covered independently of other files. Within the test files, any external files that the code depends on will be mocked out, and at least one test will be written for each method in the code file. Whenever possible, UI tests will be labeled with the UI component name that they are testing and will use snapshots, while functional tests will be labeled with the function name and will use input/output tests and method call tests.

9.11 - Integration Tests

Although the software is available that runs automated integration tests for React Native applications, none of the software that was tested both met the needs of this project and was easy to use. As such, it was determined that manual integration tests would be best for this project. Since manual tests can be prone to errors (usually caused by a change in how the tests are run), the following list describes the input and expected behavior for each of our manual integration tests.

The first suite of integration tests is the login test suite. To run the first test, open the app for the first time, and use the sign in prompt to log in. The backend function /auth/get_token should be called with GET, and should return an authentication token. As a result, the Signed In prompt should display, and screen should change to the greenhouse overview. To run the second test, close the app, clear the cache, and open the app again. Just like the first time, the backend function /auth/get_token should be called with GET, and should return an authentication token. As a result, the app should not prompt for sign in again; sign in should occur automatically with the saved password.

The next group of tests is the sign up test suite. To run the first test, open the app for the first time, and use the sign up prompt to create an account. The backend function /auth/create_user should be called with POST, and should return an authentication token. As a result, the account created prompt should display, and screen should change to greenhouse registration screen. To run the second test, log out, and then use the sign in prompt to log back in. The backend function /auth/get_token should be called with GET, and should return an authentication token. As a result, the Signed In prompt should display, and screen should change to the greenhouse overview.

The third group of tests is the greenhouse test suite. To run the first test, load the greenhouse overview screen. The backend function /mobileapp/greenhouses/ should be called with GET, and should return a list of greenhouses that are registered to the user. As a result, the first greenhouse registered to the user should be displayed, and the user should be able to swipe left and right to change between all of their greenhouses. To run the second test, swipe left or right in the greenhouse overview screen. No backend functions should be called, but the displayed greenhouse should change. To run the third test, tap on a dot in the greenhouse overview screen. No backend functions should be called, but the displayed greenhouse should change to the dot that was clicked on. To run the fourth test, swipe all the way to the right in the greenhouse overview screen. No backend function should be called, and the screen should change to the greenhouse registration screen. To run the fifth test, tap on the + button in the greenhouse screen. No backend function should be called, and the screen should change to the greenhouse registration screen. To run the sixth test, swipe up in the greenhouse screen. The backend function

/mobileapp/greenhouses/:id should be called with GET, and should return data for the greenhouse. As a result, battery and tank statistics for the greenhouse should be displayed. To run the seventh test, swipe up in the greenhouse screen when statistics are displayed. The backend function /mobileapp/greenhouses/:id/history should be called with GET, and should return the recent history of the greenhouse.

The fourth group of tests is the tier detail view test suite. To run the first test, click on a harvest notification outside the app. The backend function /mobileapp/tiers/:id should be called with GET, and should return information about the tier. As a result, the detail view for tier should be displayed, and the tier should be marked as ready for harvest. To run the second test, click on the replant button inside tier detail, and select a new plant. The backend function /mobileapp/tiers/:id should be called with PUT, and should return a success callback. As a result, the app will walk the user through setting up seedlings, and then the screen will change to the greenhouse overview, which will show seedlings on the bottom tier, and no plants on the altered tier. To run the third test, click on a move seedlings notification outside the app. No backend functions should be called, and the detail view for seedlings should be displayed, with the seedlings marked as ready to move. To run the fourth test, click on the move seedlings button inside tier detail. The backend function /mobileapp/tiers/:id should be called with PUT, and the backend should return information about the newly updated tier. As a result, the app will walk the user through moving the seedlings to their proper tier, and then change to greenhouse overview, which will show no seedlings on the bottom tier, and plants on the altered tiers.

The fifth group of tests is the user detail test suite. To run the first test, click on the user icon in the greenhouse screen. No backend function should be called, and the user detail screen should display, with the user name and the user's settings. To run the second test, click logout on the user detail view. No backend function should be called, and the user should be logged out and login screen should be displayed.

The sixth group of tests is the greenhouse Registration test suite. To run the first test, swipe all the way to the right in the greenhouse screen. No backend function should be called, and the screen should change to the greenhouse registration screen, and user should be prompted to plug in greenhouse and wait until lights pulse. To run the second integration test, click continue once greenhouse lights pulse. No backend function should be called, and the camera should open to the QR scanner. To run the third test, a non-greenhouse QR code should be scanned. No backend function should be called, and the app should display an error message indicating that an invalid QR code was scanned. To run the fourth test, a greenhouse QR code should be scanned. No backend function should be called, and the app should connect to the greenhouse network and a list of local wifis should be displayed. To run the fifth

test, a wifi that needs a password is selected from the list. No backend function should be called, and the app will prompt the user for a password, and validate it. To run the sixth test, a wifi that does not need a password is selected from the list. The backend function /mobileapp/greenhouses should be called with POST, and the backend should return confirmation that a new greenhouse was created. As a result, the app sends the wifi authentication to the greenhouse, and shows the plant selection screen. To run the seventh test, select a tier on the plant selection screen. The backend function /mobileapp/tiers/:tierId should be called with PUT, and the backend should return a success call. As a result, plant choices pop up and allow the user to select the plant for the tier. To run the eighth test, click continue on the tier setup screen. No backend function should be called, and the user is prompted to fill the water tank. To run the ninth test, click continue on the water tank screen. No backend function should be called, and the user is prompted to fill the nutrient tank. To run the tenth test, click continue on nutrient tank screen. No backend function should be called, and the user is prompted to set up seedlings. To run the eleventh test, click finish on the seedling screen. The backend function /mobileapp/greenhouses/ should be called with GET, and should return a list of greenhouses for the user. As a result, the user should be redirected to the greenhouse detail view, and the new greenhouse should be displayed. To run the twelfth test, click on cancel setup from anywhere in the registration flow. The backend function /mobileapp/greenhouses/ should be called with GET, and should return a list of greenhouses for the user. As a result, the user should be redirected to the greenhouse detail view, no new greenhouse should be displayed, and any setup that has been done should be cleared.

The seventh group of tests is the accessibility test suite. The accessibility tests are based on VoiceOver on iOS, but should have similar results on Android devices. To run the first test, begin moving a single finger over the top of the greenhouse overview screen. As a result, VoiceOver reads out greenhouse name, position in list, and a prompt to swipe left or right to switch greenhouses. To run the second test, begin moving a single finger over the greenhouse list when on the last greenhouse. As a result, VoiceOver reads out greenhouse name, position in list, and a prompt to swipe left to switch greenhouse and right to add a greenhouse. To run the third test, begin moving a single finger over the greenhouse overview. As a result, VoiceOver reads out the tier number, the plant in the tier, and the status of the tier. To run the fourth test, begin moving a single finger over bottom of screen in greenhouse. As a result, VoiceOver reads out a prompt to scroll up for more content. To run the fifth test, begin moving a single finger over icons in greenhouse stats. As a result, the icon name (battery, water level, or nutrient level) is read out, followed by the relative percentage (empty, low, falling, high, full). To run the sixth test, begin moving a single finger over data points in historical stats. As a result, individual data points are read out with the type context, including indication of whether the overall line is rising or falling at that point. .

10.0 - Plant Parameters

Although we hope that, in time, Pocket 'Ponics will be able to support a wide variety of plants, for the Senior Design project we have limited time and resources. As such, we will set up the system to handle 4 different types of plants: tomatoes (*Solanum lycopersicum*), spinach (*Spinacia oleracea*), turnips (*Brassica rapa*), and green beans (*Phaseolus vulgaris*). Each of these species have different nutrient and light needs, as well as slightly different optimal growing mediums between the species.

For tomatoes, we will place a single tomato plant at the top of the greenhouse in a soft collar and wire support. Due to time constraints, we will be using a pre-grown, immature tomato plant, as tomato plants can take almost 6 months to mature.

Conditions:

- 10 - 16 hours of light per day
- pH of 5.5 - 6.5
- Electrical conductivity of 2.0 - 5.0

For spinach, we will place 18 spinach plants in 1 inch rockwool plugs, organized in a 3x6 grid with 5-6 inches between each spinach plant. The spinach plants will be grown from heirloom seeds.

Conditions:

- 10 - 14 hours of light per day
- pH of 5.5 - 6.6
- Electrical conductivity of 1.8 - 2.3

For turnips, we will place 18 turnip plants in 2 inch plugs filled with perlite, organized in a 3x6 grid with 5-6 inches between each turnip plant. The turnip plants will be grown from heirloom seeds.

Conditions:

- 12 - 14 hours of light
- pH of 6.0 - 6.5
- Electrical conductivity of 1.8 - 2.4

For green beans, we will place 8 green bean plants in 1 inch rockwool plugs, organized in a 2x4 grid with 7-8 inches between each green bean plant. The green bean plants will be grown from heirloom seeds.

Conditions:

- 12 - 13 hours of light
- pH of 5.9 - 6.1
- Electrical conductivity of 2.0 - 4.0

11.0 - Prototype Construction

Once all software and system hardware testing is completed then construction of the full working prototype can be started. To prep for this all system components need to be gathered and the API and app software need to be running.

To start base construction use the hydroponics shelving as the base all other components will be attached or mounted to this unit. The first step will be to mount the plant water tanks onto the shelves, these will house all of the seedlings that will be put into system. Ensure stable mounting of the water tanks so that the sensors and pump systems can further be mounted later with no issue of them coming loose. Continue base mounting by attaching the water and nutrient tanks on the bottom tier of the platform.

With all liquid containers mounted the process of mounting the electronics is the next step. This starts with mounting the power system and its enclosure within the bottom tier next to the nutrient and water tanks. Power lines will come out the side of the enclosure and will be snaked up along the support structure of the hydroponics shelving and will be routed to the SOC and MCU. The power connections for the pumps will snake down along the bottom tier where the pumps will be located. Once all power lines are mounted properly the MCU and SOC components will be mounted between tiers 2 and 3. This location is optimal for the MCU and SOC due to the fact that the control signals being read and issued. from each tier will be relatively low voltage and current meaning that the signal will degrade if the connection is too long. The connections between the SOC and MCU will be attached now as well, making sure that the proper pin connections are being used on each end so data can be read and transferred between the devices properly.

With the mounting of the MCU and SOC completed, the sensors can now be mounted with a pH, EC, and two water level sensors being attached to each tier. The pH and EC sensor will be mounted to the sides of the water containers and will be within the range of acceptable water levels. The water level sensors will be mounted with one in the proper position and one being inverted for the low water level sensor. The low water level sensor will be placed about a third of the way from the bottom of the tank and the acceptable water level sensor will be placed a fourth of the way down from the top of the tank. Proceed to start running the connections from the sensor to the SOC it will be ran on the same side as the electrical connections and will run to the housing enclosure for the SOC and MCU. Connect all sensor traces to the proper pin assignments to which they will be read.

The next step of the construction process is to mount the devices controlled by the MCU, this would include the LEDs and the pumping systems. On each tier the LEDs will be mounted to the bottom of the tier above. With their power connections being routed to the same side as the power and SOC/MCU systems.

The LED power lines will then be connected with the potentiometer control signal being attached to the MCU system. For the pump systems the tubing and pumps will be connected from the water tank to the designated tiers water container. The nutrients will be connected in the same fashion as the water pumps. The electrical connections will be connected to the pumps along the bottom tier. The control relays will be located closer to the MCU system so that the signal to power the pumps doesn't degrade from a long trace.

After all electronics and systems have been mounted and connected properly the power system will be connected to a wall socket to start providing power to the system. At this point the system will be up and running functionally and can be accessed through the app interface.

For this part of setup connect to the system through the app interface and choose the proper network SSID and enter its password. This should bring the entire system online and will connect the system to the API. Completely fill the water and nutrient reservoirs with the appropriate liquids. Then fill the tanks with water appropriately and add the prompted amount of nutrients to the tanks as well. You will then be able to choose the plants for each tier of the greenhouse.

The greenhouse system should be entirely up and running at this point and will be able to operate without the input of a user anymore. For this reason the construction of the system is completed.

12.0 - Conclusion

We hope that Pocket Ponics can provide freshly grown fruits and vegetables to people who previously could not afford such things. The three major sections of our project - frontend, backend, and greenhouse - work together to ensure the best user experience possible, and to minimize the amount of knowledge that users need to operate the greenhouse. This ensures that users from every demographic can use Pocket Ponics.

We have also worked hard to ensure that Pocket Ponics is accessible to everyone. From ensuring that our app meets WCAG standards to building the greenhouse in a logical manner, we worked to maintain openness and accessibility across all parts of the project. We also made sure that all components of our project are available as an open source project on GitHub, allowing future developers to contribute to and improve our product.

As a team, we have grown a lot over the past two semesters. We have maintained regular communication and resolved differences in a peaceful manner, even when a large miscommunication threatened the cohesiveness of the group. Nevertheless, we managed to coordinate and resolve hurt feelings, and then continue on as a group once more.

We also dealt with external issues outside of senior design. About half-way through the second semester, COVID-19 caused shutdowns across the globe. This meant that many of the parts that we had planned to use for our project were no longer available, and our team members were also split up, unable to meet in-person due to social distancing measures. This slowed down some of our progress, and meant that we had to make component substitutions and reorganize some of our planning to cope with the fully online teamwork. Despite this chaos, we pulled through and made a complete, working project, coordinating our project online through Discord, Youtube, and Zoom.

Overall, the team managed to overcome all the roadblocks and challenges that we faced to build a well-functioning prototype of the Pocket Ponics system. Our system met all of the initial criteria we set for it, and even managed to achieve some of our stretch goals, such as Machine Learning for plant recognition and an administrative portal to help manage which plants the users should be able to grow. We think that the Pocket Ponics prototype is an excellent demonstration of the goals of the project, and hope to continue improving it in the future.

13.0 - Appendix

The appendix contains the bibliography for this document, including all of the sources that were researched when determining the context and broader impacts of the Pocket ‘Ponics project.

13.1 - Bibliography

[1] "Labor Force – By Occupation". The World Factbook. Central Intelligence Agency. Archived in the Wayback Machine from the original on 22 May 2014. Retrieved 13 Sept 2019.

[2] “FAO and Post 2015.” FAO, Food and Agriculture Organization of the United Nations, 2015, www.fao.org/resources/infographics/infographics-details/en/c/266124/. Retrieved 13 Sept 2019.

[3] “United Nations Millennium Development Goals.” United Nations, United Nations, www.un.org/millenniumgoals/. Retrieved 13 Sept 2019

[4] “United Nations Sustainable Development.” United Nations, United Nations, 2015, www.un.org/sustainabledevelopment/. Retrieved 13 Sept 2019

[5] ”Household Food Security in the United States in 2018”, ERR-270 Alisha Coleman-Jensen, Matthew P. Rabbitt, Christian A. Gregory, and Anita Singh. U.S. Department of Agriculture, Economic Research Service, 2019. Retrieved 13 Sept 2019