Pocket Ponics - A Semi-automated Microscale Greenhouse Leveraging Hydroponic Technology

Catherine Abbruzzese, Matthew Bonsignore, Alexandra Cusell, Graham Hill, Elli Howard, Rohan Patel

University of Central Florida, Department of Computer Science and Department of Engineering, Orlando, Florida, 32816, U.S.A.

Abstract **- Pocket Ponics is a semi-automated hydroponics system that allows users with no technical or agricultural knowledge to grow plants in the comfort of their own home. It consists of three parts: the greenhouse sensor grid, which holds the plants and contains the pumps, lights, and sensors necessary to interact with them, the backend, which stores and provides data, and the frontend, where users can monitor and interact with their greenhouse in an easy-to-understand UI.**

Index Terms **- Agricultural engineering, Food technology, Greenhouses, Home automation, Internet of Things, Open source hardware, Open source software,**

I. INTRODUCTION

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 [1]. Indeed, eradicating hunger was the first of the eight Millennium Development Goals [2] that the United Nations hoped to meet by 2015, and is the second of the seventeen current Sustainable Development Goals [3], 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 [4]. Members of such households have a higher rate of health issues due to poor nutrition and have reduced lifespan expectancies.

To help solve this problem, Pocket 'Ponics will automate the use of small-scale hydroponics. To do this, we have devised a three-part system: a greenhouse and sensor grid system that will monitor the plants and adjust water, nutrient, and light levels to the optimum values for each plant species; a backend that will collect data from the greenhouses store the data in a database, and process the data before surfacing it in an API; and a frontend that will retrieve data from the backend and present it in an easy-to-understand interface for the users.

II. GREENHOUSE AND SENSOR GRID

A. System Components

This section briefly describes then expands upon the individual components and system modules that were designed or purchased for integration. All of these modules are needed to bring the greenhouse system up and running to its full capabilities.

B. Microcontroller

We are using the ATMega2560 as our Microcontroller Unit (MCU) because it has multiple UART, SPI, and I2C connections which can support serial connections when paired with the smaller MCU the ATMega16-aur included on the development board, which is used in our system for a simpler communication method. The system also has a considerable amount of flash and non-volatile memory that, along with a clock speed of 16MHz, allows the system to respond quickly to requests over the serial connection. The system has a total of 86 supported GPIO pins with 16 of them being analog pins.

Per the specification provided from arduino and by ATMega the operating voltage for the board was at a lower bound of 5 volts with a recommended maximum up to 12 volts, with over volting up to 20 volts supported but not recommended. We tested the board at different operating voltages ranging from 6 up to 13 volts. With our power supply being designed to output 12 volts we settled on this as a direct line into our MCU from the power supply to lower the amount of extra needed hardware and calibration, paired with the fact that the extra power will prevent the MCU shutting off due to power draw.

C. System on Chip

Due to the complexity of our code and system we decided to implement a secondary system that was a System on Chip to run in tandem with the microcontroller. For this we chose a Raspberry Pi 3B, with a Quad Core 1.2GHz Broadcom CPU on board it was possible to run a full linux operating system. The system has 1GB of onboard RAM, an SD slot for storage, supports wifi, and has an HDMI port for a visual on the desktop environment that allows for live debugging. With the filesystem from the operating system we can also store debug files if there were to be any issues or bugs.

The Raspberry Pi 3 Model B is the brain of the entire greenhouse system. The Arduino may be responsible for directly making changes and receiving inputs from the system, but the System on Chip is what determines what actions to make and sends out the proper commands and information depending on those inputs. Based on the lack of analog pins and the limited amount of digital pins available on the System on Chip, it would have been impossible to collect and process all of the sensors and other components that the greenhouse and hydroponics system needs to function on an hourly basis. But without the communication properties that it has, along with it's faster clock speed and additional RAM, the system would not run properly.

D. Electrical Conductivity Sensor

The nutrient concentration present in each tiers water supply is determined upon the pH and total dissolved solids present. The water of a tier could be well within an acceptable range but not in high enough concentration to actually benefit the plants, so keeping an accurate measure of this value is critical for the well being of the plants in the system. To measure the total dissolved solids we chose to implement an electrical conductivity or total dissolved solids sensor into the system. This sensor measures the Total Dissolved Solids (TDS), in ppm, or parts per million. Most plants require a concentration of nutrient well within this sensor's measurable range. Because of this the TDS sensors tell us when the plants are in their ideal range,or if more nutrients are needed.

The DFRobot analog TDS sensor is the optimal choice for ease of integration and price. The sensor operates by having an input voltage ranging from 3.3 to 5.5 volts inputs into the signal transmitter board which allows the board to take a reading of electrical potential across the two prongs of the probe. This reading can be any value between 0 to 2.3 volts depending on the TDS and the input voltage. Readings are sent to the MCU at all times so any readings must be collected as a group and averaged to ensure that the reading is "correct" before using it for nutrient calculations.

Before full integration into the system for use we ran a number of tests on the sensors for calibration. Using a 770ppm test solution with an input voltage of 5 volts, ten measurements, using their average as the final reading. All four sensors performed well within a ±5% tolerance. Our database is precise to the tenths allowing slight imprecision.

The only issue with integration of this sensor was the worry of signal degradation on the uppermost tiers due wire length. We repeated the previous test with longer leads, testing with a 26 AWG wire with a length of 1.4 meters and an expected wire resistance of approximately 0.162 ohms. Post testing we found that a wire of this length at these voltages and current levels preserved the signal integrity with little distortion or degradation.

E. pH

The amount of pH concentration of a solution is very important to the nutrient intake of plants. For this reason we opted for the Gravity analog pH sensor, with ease of integration and for a reasonable price compared to other pH sensors available this was the best choice for our system. A pH sensor is needed because the nutrient level is inversely proportional to the concentration of pH in the solution, meaning that the lower the pH concentration is, the more a plant can absorb nutrients.

The pH sensor can run off of an input voltage range of 3.3V to 5V. This works optimally with the 5 volt step down buck converter that we used for the Electrical Conductivity sensor. Similarly, the power and ground lines of the pH sensor will be connected to the power buck, and the signal pin is then connected directly to the Arduino's analog pins. The pH sensor pins are mapped to the A1, A3, A5, and A7 analog pins. The Arduino collects the reading from the pH sensor several times before averaging and converting it to a normalized pH range from 0 to 14.

Since the pH sensors that we used for this project did not come pre calibrated, some manual adjustments need to be made before they are used. This is done by collecting a few readings from testing solutions of differing pH values. After collecting these readings, they need to be linearly interpolated to find the function that can convert the reading of the pH sensor to the correct pH value. This must be done to every pH sensor to ensure they read in the correct values.

F. Water Level

Each tier will have its own water level sensor pair, along with water level sensors inside the reservoir tanks. Due to the fact that each plant requires different amounts of water 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 is needed for growth. They are also used as a precaution to prevent the tier from overfilling.

A pair of water level sensors in the reservoir tank used for containing the system's water supply for the pumps is also needed for the system to notify users that the reservoirs are low and will run out of pumpable water.The sensor is triggered when the bobber raises off the base, allowing a signal to pass through. When the water level sensor reaches this height, a signal will be passed. 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.

The Anndason float switch water level sensor proved to be useful and easy to use in our project. The water level sensor functions by applying a voltage to the one wire on the sensor with the other connected at a digital pin on the MCU. When the bobber on the sensor reaches its maximum, the current is allowed to pass through. As stated previously a voltage is applied to one wire with the other ground with a resistor and also placed into the arduino to be read as a digital high or low depending on the bobber's position.

Every tier has a pair of water level sensors, such that they can be used in tandem to determine if the water level is below, above, or inside a given range. This is achieved by inverting one sensor of each pair so that it will send a signal when the water level is too low, while the other will when the water level is too high. Depending on where the sensors are mounted, the range in between the two sensors can be modified at will.

G. LEDs

The LED system is in place to provide the plants with light when they are being stored in the greenhouse, indoors. For this we utilized the AveyLum Plant Grow Lights operating at 12 volts at 2 amps with 24 watts per a strip per their specification. These lights were chosen due to the specific wavelengths emitted by the red and blue LEDs on the strip proving them to be optimal for plant growth when the plants are restricted to an interior setting. The stips came with clear markings for cutting strips to specific lengths with large copper pads for soldering and mounting wire connections and other strips together.

The LED lighting strips are used to supply the growing plants with sufficient light to thrive on. 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 of the strip. The quantity of the LEDs needs to be around 60 LEDs per meter to provide enough light for proper growth, with a ratio of one red to four blue LEDs, these LEDs have respective wavelengths being 660 nanometers for the red LEDs, while the blue LEDs provide 455 nanometers.

H. Pumps

The greenhouse wouldn't be able to maintain its semi-autonomy without a water and nutrient management system in place. For this we chose a pump that had enough power to push water and nutrients up to the top tier and with enough suction to pull water out of the reservoirs. We chose to operate the pump at 12 volts at 0.5 amps per the specification, which the maximum allowed operation voltage. The pump also came with two terminals for ground and live with one marked for the positive terminal to make wire connections easy to distinguish.

The water pumps that will be used in this project for 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. This pump is relatively small for its dimensions (95mm x 47mm x 36mm), with a mount system allowing it to be easily placed and mounted securely. With the use of a relay for power control this pump is easy to control through the MCU.

I. Power Supply

To provide power to our greenhouse we need a power system that is variable and has enough wattage to run multiple pumps and the LEDs at all times. For this reason we opted to use the pre-built system, the Ledmo switching transformer AC 120V to DC 12V 20A power supply. This system comes pre-built with 3 ports for AC inputs (Live, Neutral, and Ground), with the output ports being three ground ports and three 12 volt DC ports.All ports come with a screw mount and a plastic cover.

Due to a high power demand for our system we initially opted for a power system that could convert an AC signal to DC. These components are widely used in almost any system that has a high power demand that requires a constant power signal. This component is paired with the relays, step down bucks, and micro USB converter to power the entirety of the greenhouse.

Before full integration into the system we opted to stress test the system under different loads. Our system is designed to limit the amount of pumps running at the same time, to reduce power draw. A full test of components under these conditions caused a small amount of heat build up, but otherwise the power supply was stable for extended periods of time.

J. System Concept

This section's intent is to describe how the Pocket Ponics system is designed to run as a completed system. Once the setup of the greenhouse in the app is complete Fig. 1 details the behavior of the hardware once it has become autonomous.

After the initial setup detailed above the hardware and its code starts to run independently of the rest of the overall system (i.e. the app and API) with its only interaction being the transfer of Tier readings to the API for record keeping purposes. At the start of every hour the SoC will poll the MCU to request tier readings from the greenhouse. The greenhouse in turn will report the nutrient, water levels, and light levels for each tier along with the reservoir levels to the SoC to check if they are within the appropriate ranges. The SoC will report any pulled data to the API backend and then proceed to check if the provided information is within the desired ranges. If all information is within the correct ranges then the system will sleep for one hour before requesting more data from the MCU. If any data is outside of the desired range then the SoC will send a signal to make an adjustment pass control to the MCU to make adjustments to the tiers or if one of the reservoirs is low it will send a push notification to the app for a user to refill them.

When control is passed back to the MCU for adjustments the System on Chip sends a few strings to the MCU indicating which tiers need to be adjusted and in what way. The three adjustments to the system are if the water level is too low, if the nutrient level isn't in a high enough concentration, and if the light level for that tier is incorrect for the current time. The MCU uses these string(s) to determine what adjustments are needed on which level. If the water level is below the desired range the MCU will send a digital high signal to a relay to activate the pump for that tier. The digital high will stay on for 1 second before deactivating the pump. If the nutrients concentration is too low and the water level isn't at the maximum then the MCU shall send a signal similarly to the water pump to the nutrient pump relay and will deliver a set amount of nutrients to the desired tier. Lastly is the light levels or rather if the grow LEDs should be active. If the LEDs are active outside of the tiers designated hours the MCU will stop sending a digital high causing the power relay for that level to close, shutting off the lights. On the other hand if the LEDs are off within the desired hours the MCU shall send a digital high to the tiers power relay to open the relay and turn on the LEDs. Once adjustments are made to the greenhouse the MCU will proceed to sleep until the next request from the SoC to provide tier data.

K. Hardware Code

This section pertains to all of the software that was used on both the Microcontroller Unit and System on Chip for the Pocket Ponics system. The objective that the software for the system is designed to accomplish is on a fixed hourly basis, check every tier of the greenhouse, and make the appropriate actions to maintain the given tier.

L. Microcontroller Unit

The MCU's software accomplishes its objective by the use of several functions. There are several functions used for communication with the SOC, a few for getting a reading from a sensor for a given tier, one to perform an action to a given tier, and finally one for a main loop. The main loop calls the two communication functions. One is used as a trigger for when communication begins between the two devices. The other is used to receive data from the SOC, perform an action, and send data back. Within the function that receives the data, depending on what 'tag' the data has, it will either call the function that will perform an action on the greenhouse, or gather readings from the greenhouse and send it back to the SOC. Within the function that performs an action, it receives the tier number, type of component, and an amount. This will either turn on and off a water pump, or nutrient pump for one second, or toggle an LED strip array either on or off. The function that gathers readings form the greenhouse will call functions to get a reading for all of the sensors of a specified tier, and it will get the readings from the two reservoir tanks. Once all of the data has been gathered, it will then be concatenate into a string and send it back to the System on Chip

M. System on Chip

The general flow of the software for the System on Chip (SOC) is that it begins with the main loop that is constantly looping. In this loop it will, on every hour, will call a function to check the levels of every tier, and send the API the data collected from the greenhouse. In the function that check the levels of a specified tier, it gets the readings for the tier from the MCU, get data about the vegetation on the tier from the API, and call a function that will either turn off or on the LED strip array for the tier based off of data from the API, and check to see if a water or nutrient pump needs to be activated. Within the function that controls the pumps, it will decide whether or not to activate the pumps based on the readings on the water level and nutrient levels of the tier. If the tier's water level isn't at max it will either activate the nutrient pump if the levels are too low, or the water pump if the nutrient concentration is too high or the water level is too low. The main function will then wait an hour, and loop again.

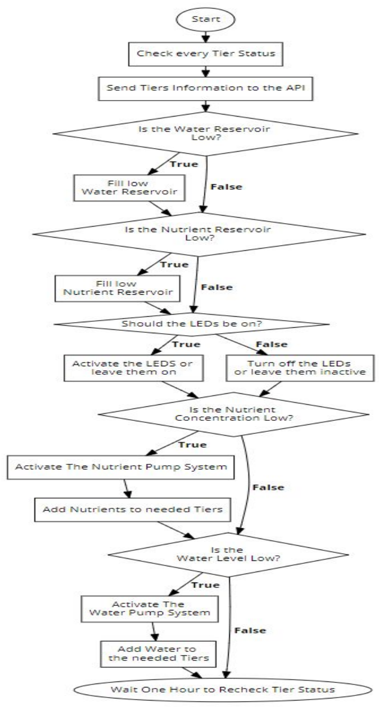

Start
Check every Tier Status
Send Tiers Information to the API
Is the Water Reservoir Low?
True
False
Fill low Water Reservoir
Is the Nutrient Reservoir Low?
True
False
Fill low Nutrient Reservoir
Should the LEDs be on?
True
False
Activate the LEDS or leave them on
Turn off the LEDs or leave them inactive
Is the Nutrient Concentration Low?
True
False
Activate The Nutrient Pump System
Add Nutrients to needed Tiers
Is the Water Level Low?
True
False
Activate The Water Pump System
Add Water to the needed Tiers
Wait One Hour to Recheck Tier Status

Fig. 1 Flow Chart of system Hardware control code

III. BACKEND

The backend component for the Pocket 'Ponics system contains two parts: The Express server containing the API routes and the MySQL Database containing user and greenhouse data. The API routes are used by the sensor grid to post data to the MySQL database and used by the mobile app to retrieve greenhouse data.

A. Technologies Used

The backend is built using Node.js and the Express.js framework. The API endpoints are located on the Express.js server and accessible to the sensor grid and the mobile app. The sensor grid and mobile app use the endpoints to interact with the MySQL database. Using API endpoints allows our greenhouses and mobile app users to communicate easily with the backend. We chose a MySQL database because of the relationships between our data. Users own greenhouses, where each greenhouse has tiers, and those tiers contain plants and sensor readings for EC, pH and water level. Our backend is hosted using the Amazon Web Services (AWS) platform because a faculty member advised us to become familiar with the technology through our project. Additionally, the AWS platform is held as an industry standard. We use an EC2 instance to host the API endpoints, an RDS instance to contain the MySQL database and an S3 bucket for storing image assets used by the mobile app. To interact with the database, we use the MySQL npm package to create a pool of connections used to perform queries and transactions on the database.

B. Backend Components

The API endpoints are grouped into four different routes: authentication, mobile app, sensor grid, and admin portal endpoints. Each route contains endpoints used for a specific purpose or used by a specific component in the project. The authentication route contains the endpoints used by the mobile app and the admin portal to authenticate users on login. The endpoints are responsible for receiving a username and password, verifying that those are valid credentials, and issuing a temporary token to the user allowing them to access other endpoints. The temporary token is valid for 30 minutes, and after its expiration, the user will need to provide their username and password to receive a new token. We use this approach to prevent the verification of credentials every time the user makes a request to the backend, which can occur many times in a short period of time. The authentication route also provides endpoints for user registration, an endpoint for the user to reset their password or change their password using their existing password. endpoints belonging to the mobile app route are used by the app to retrieve a user's list of greenhouses, as well as perform create, read, update and delete operations on a user's greenhouses and the tiers belonging to each greenhouse. The mobile app can retrieve a single greenhouse's data, as well as data for each tier in the greenhouse. Additionally, the mobile app can retrieve individual sensor readings for a given tier. The backend also stores the historical data for the greenhouse's water reservoir and nutrient reservoir levels. The mobile app can use an endpoint to retrieve the historical data for a greenhouse to display to the user. The mobile app route allows the user to create a new greenhouse and register the sensor grid to the greenhouse. Another endpoint allows the mobile app to add a new device to receive notifications. Finally, the mobile app route contains an endpoint for image classification, used when the user harvests plants from the greenhouse. On the harvest tier screen, after the user clicks the Harvest button they will be taken to a screen which allows them to take a picture of the plant they are going to harvest. The image will be passed to the backend and classified as a ripe or unripe vegetable. This gives the user a reassurance that the plant they are about to harvest is ripe and ready to harvest. The sensor grid route contains endpoints used by the sensor grid to post the different water levels, pH and electrical conductivity (EC) readings for each tier as well as general greenhouse data. The sensor grid can also use the endpoints to retrieve ideal values for each tier's pH and EC as well as the light schedule required for each tier. The endpoints contained in the admin portal route allow the administrator portal to perform operations on the plant_ideal table in the database. This table is responsible for storing the ideal values for each different type of plant the platform supports. This includes ideal ranges for pH, electrical conductivity and the amount of light each plant needs. The endpoints under the admin portal route allow the administrator to add new plants to the list of supported plants, view and update existing plants and delete a plant from the table.

Another function of the backend is to provide users with notifications on their devices for different harvesting and transplanting steps. Each morning, the backend retrieves the greenhouses which have seedlings that are ready to transplant or tiers that are ready to harvest. For each user, a list of devices is retrieved from the database and appropriate notifications are sent to the user. Additionally, the backend is responsible for sending notifications to alert the user to add more water to the greenhouse's water reservoir or add more nutrients to the nutrient reservoir. The backend will also notify the user if the power source is interrupted.

The MySQL database contains eight tables: active_sessions, devices, greenhouse, historical_data, plant_ideal, sensor_grid, tiers, and user. The active_sessions table stores the tokens that the backend has issued to users. An event scheduled to run every 24 hours removes the tokens that have expired. The devices table stores the Expo token for each user, required to send notifications when the tiers are ready to harvest when seedlings are ready to be transplanted or water and/or nutrient reservoirs need to be refilled. The greenhouse table stores general information about the greenhouse including its name, seedling_time and water, and nutrient reservoir levels. Each greenhouse has a unique identifier used globally in the database tables. The historical_data table stores the water and nutrient reservoir levels for each greenhouse for each hour, to provide the user with historical data on water and nutrient reservoir levels for each greenhouse. The plant_ideal table stores the plants our platform supports and their names, ideal pH and EC values and the amount of light they require to grow. The table also stores the number of plants that each tier can contain for that type of plant, ideal temperature range and the amount of time the plants require to become ripe. Additionally, S3 bucket URLs are stored and used by the mobile app to retrieve the plant's appropriate image assets. The sensor_grid table stores the sensor grid registrations for each greenhouse. Each registration record contains a serial number, as well as a password hash, user_id, and greenhouse_id. The tiers table contains the pH, EC and water level values for each tier in the greenhouse. Additionally, the plant growing in the tier, as well as the greenhouse_id associated with the tier, is stored. The tiers table also stores the date the tier's plant will be ready for harvest and the time the light schedule should start. The user table contains user information. Each user has a user_id, a unique identifier used globally in the database tables. Additionally, the user's email and password hash are stored. The user's email and password is used upon login, and the user_id is used internally to identify a user and their data. A flag stores a 0/1 value indicating if the user is an administrator. The password hash is stored in the case of a security breach. If the database contents are retrieved by an unauthorized party, the password hash can be stolen but not used by an attacker to log into a user's account or perform operations on a user's behalf.

C. *Image Classification*

The backend performs image classification on images of vegetables growing in the greenhouse using a convolutional neural network. Our team used a pre-trained, modified Inception V3 neural network to perform classification on nine classes: ripe-tomato, ripe-greenbeans, ripe-turnip, ripe-spinach, unripe-spinach, unripe-tomato, unripe-greenbeans, unripe-turnip and not-vegetable. The neural network is able to perform image classification with a 91 percent accuracy and includes anomaly detection to prevent the neural network from falsely classifying images that are not vegetables as ripe or unripe vegetables. The training was performed using a dataset our team built using a Google image scraper and approximately 1000 images of vegetables and common objects found around the house and outside the home that could be falsely classified as vegetables. To prevent overfitting, our neural network utilizes dropout, early stopping and checkpointing. Early stopping prevents the neural network from training when the validation accuracy is becoming stagnant while training accuracy remains increasing. Checkpointing saves the model's best-performing weights, even if they are before the end of the training process. Dropout temporarily disables a certain number of random nodes in the layer during training to prevent overfitting.

IV. FRONTEND

The frontend of the Pocket 'Ponics system is comprised of two pieces: the mobile application that allows users to monitor their greenhouses and walks them through any interactions with the plants, and the administrative web portal that allows system admins to add, modify, and delete types of plants that can be grown in the greenhouse.

A. Technologies Used

To build the app, we used the React Native framework, which allowed us to build the app independent of any mobile operating system. Similarly, we leveraged the Expo CLI framework to allow us to use native technologies such as the camera and notification system without having to worry about the specific details of whichever native operating system and hardware the app might be running on. Together these technologies abstracted out the details of various phones, allowing our app experience to be consistent across whatever devices the users may choose.

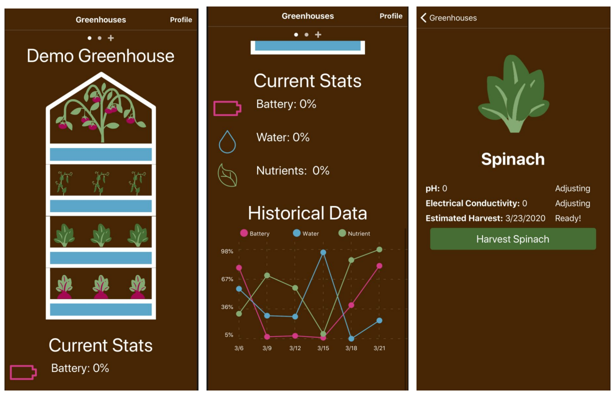
For the admin portal, we harnessed React to allow for modular, adjustable components that matched the look and feel of the app. We also harnessed HTML5 and Javascript cookies to ensure that all users who access the admin portal are properly authenticated, so that unauthorized users cannot modify details about the plants that are so important to the system.

B. App Design

The app is designed to be simple for users to understand, with three major sections: the login section, the greenhouse monitoring section, and the greenhouse registration section.

When a user opens the app for the first time, they will be directed to the login section. There, they can create a new Pocket 'Ponics account, log in with an existing account, or reset a forgotten password on their account. Additionally, advanced users can click on a hidden button to open up the list of servers, where users that prefer to set up a private backend instead of the default AWS backend can enter the host and port of their private server, which the app with then use for credentials and data. If they log in or create a new account, their credentials are stored and verified with the backend, and then, assuming they were verified, the app requests information from the backend about their greenhouses and then they are passed through to the greenhouse monitoring section.

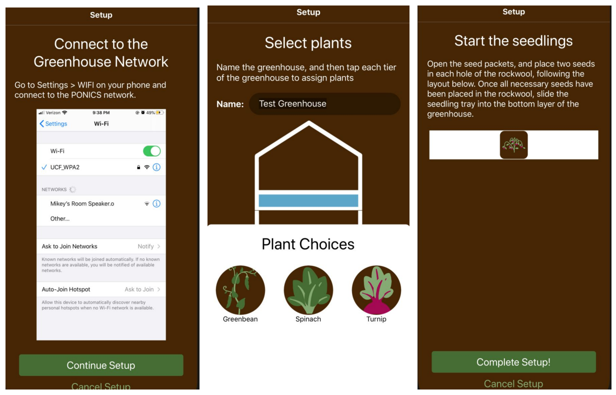
In the greenhouse monitoring section, users will see a simplified representation of each greenhouse that they have registered. They can swipe left and right (as indicated by the dot indicators on the top of the screen) to change between different greenhouses and can swipe up on any greenhouse to see the current water, nutrient, and battery levels for that greenhouse, as well as a chart of historical data for the water, nutrient and battery levels. Users can also click on each tier of the greenhouse to gain detailed information on the current electrical conductivity and pH levels for that tier. If the tier is ready to harvest, users will be presented with a harvest button that will open up the in-app camera, and instruct them to take a picture of the plant they wish to harvest. The app then contacts the backend, where machine learning determines the type and ripeness of the plant, which is then conveyed to the user. Users can then proceed to the harvest screen, where detailed instructions explain how to properly harvest each plant. After harvest is complete, the user is returned to the greenhouse overview. In addition to the tier view, users can also access the profile screen by clicking on the top right corner of the greenhouse overview. From the profile screen, users can check which account they are logged into, and they can view the total number of greenhouses currently registered to that account. They can also change their password or log out; either action will return the user to the main login screen.



pH: 0 Adjusting
Electrical Conductivity: 0 Adjusting
Estimated Harvest: 3/23/2020 Ready!

Figure 2. A subset of greenhouse monitoring screens. From left to right: greenhouse overview, greenhouse statistics and historical data, and tier detail view

To access the greenhouse registration, users swipe all the way to the right in their list of greenhouses. (New users are immediately presented with this screen, as they have no existing greenhouses to monitor.) There, they are walked through the process of setting up a greenhouse, step-by-step. They are instructed to plug a new greenhouse in and wait for the lights to flash purple, indicating that the greenhouse is ready. Then the user is instructed to connect with the LAN network that the greenhouse has established, with clear visual instructions on how to do so. Once they have connected to the Greenhouse LAN, the greenhouse passes a list of networks that it can access, and the user selects a WiFi and enters any necessary password. Then users are asked to name the greenhouse and select which plants are growing on each tier. Once the greenhouse is named and the tiers are selected, users are passed through a series of instructional screens that inform them how to fill the water and nutrients, and how to prepare the rockwool to plant in. Finally, users are instructed on how many of each type of seed to plant, and once they verify that they have planted all the seeds, the app sends the final information about the greenhouse to the backend and returns to the greenhouse overview screen, which now displays the newly registered greenhouse in addition to any previously registered greenhouses.


Setup

Connect to the Greenhouse Network

Go to Settings > WIFI on your phone and connect to the PONICS network.

Known networks will be joined automatically. If no known networks are available, you will be notified of available networks.

Allow this device to automatically discover nearby personal hotspots when no Wi-Fi network is available.

Setup

Select plants

Name the greenhouse, and then tap each tier of the greenhouse to assign plants

Name: Test Greenhouse

Setup

Start the seedlings

Open the seed packets, and place two seeds in each hole of the rockwool, following the layout below. Once all necessary seeds have been placed in the rockwool, slide the seedling tray into the bottom layer of the greenhouse.

Figure 3. A subset of greenhouse registration screens. From left to right: greenhouse LAN connection, greenhouse naming and tier selection, and final seedlings list

In addition to the internal app flow, users may also receive notifications from the mobile app that indicate low water, low nutrients, or times to plant and harvest. When the app is not open, these will show up in the notifications panel created by the phone's operating system and can be clicked to open the app to the relevant greenhouse or tier. When the app is open, the notifications will show up as a green pop-up at the top of the screen, and clicking on this popup will also take the user to the relevant screen.

C. *Administrative Portal Design*

The administrative portal is attached to the public Pocket 'Ponics website as a hidden page. Administrative staff will need the URL of the page to access it, which helps prevent general visitors to the website from accessing it, potentially with the mistaken belief that it will allow them to view their greenhouses. Once at the hidden page, users will need to log in with an admin-level account, whose credentials are verified with the backend in a similar manner to the mobile app verification. Administrators that successfully log in are taken to a list of all the different types of plants that the system can currently grow. There, administrators can click on any of the plants to modify their name, maximum and minimum pH values, maximum or minimum EC values, how long it takes the plant to grow, how many hours of light the plant requires to grow, what the instructions are for harvesting the plant, and the different images that the app displays of the plant. Administrators can also add new plants into the system, or delete plants from the system that should no longer be grown.

V. CONCLUSION

With these three systems working together, it is our hope that Pocket 'Ponics can help to alleviate food deserts across the US, particularly in urban areas. While we have made every effort to ensure that Pocket 'Ponics is easy to set up and use, we also feel that open-source software and hardware go a long way towards improving the affordability of a system. As such, we have made all components of Pocket 'Ponics open-source, so that anyone who wants to set up their own Pocket 'Ponics greenhouse, app, or server can do so. All of the code for the software (sensor grid, frontend, and backend) is available to the public on Github, along with instructions on how to set up each part. By making it open source, we hope to encourage future developers to create and improve upon their own Pocket 'Ponics systems, and thereby contribute to the continuing success of the project.

VI. ACKNOWLEDGEMENTS

We would like to thank Dr. Heinrich and Dr. Leiniker for their guidance on the computer science aspects of this project, and Dr. Richie and Dr. Wei for their guidance on the computer and electrical engineering portions of this project.

VII. REFERENCES

[1] "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.

[2] "United Nations Millennium Development Goals." United Nations, United Nations, www.un.org/millenniumgoals/. Retrieved 13 Sept 2019

[3] "United Nations Sustainable Development." United Nations, United Nations, 2015, www.un.org/sustainabledevelopment/. Retrieved 13 Sept 2019

[4] "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