THE UNIVERSITY OF BRITISH COLUMBIA

IoT Monitoring of Aquaponic and Hydroponic Food Production

Design

UBC Electrical and Computer Engineering Capstone 121

Carson Berry
Lynes Chan
Mason Duan
Jayden Leong
Hannah Xu

for

Nelly Leo, UBC SEEDS Sustainability Program
Bernhard Nimmervoll, UBC Mechanical Engineering

February 22, 2021

Revision History

Date	Author	Sections	Change
2020-10-05	JL	1.0 - 2.0	Creation of document skeleton
2020-10-30	JL	1.0 - 5.0	Added system overview and skeleton of subsystem sections
2020-11-29	LC	3.0 - 4.0	Added the design decisions made for the sensors and the control devices
2021-02-19	CB	3.0 - 4.0	Added details about power architecture, sensors, mounting, hardware, heat management
2021-02-20	JL	1.0 - 2.0	Updated system overview to add brief overview of hydroponics and aquaponics, new system diagram and updated purpose. Updated IoT section with details about the cloud implementation, along with other minor edits to microcomputer section.
2021-02-20	MD	6.0	Updated software section to include use case digagram, screen explanations, and how the mobile application interacts with the back end.

Contents

1 System Overview **4**

1.1 Hydroponics and Aquaponics 4

1.2 Project Purpose 5

1.3 High-Level System Design 5

2 Internet of Things (IoT) **6**

2.1 Microcomputer 7

2.2 Cloud 9

2.2.1 Cloud Communication 11

2.2.2 Cloud Storage 12

3 Power **13**

4 Sensors **15**

4.1 Analog to Digital Conversion 15

4.2 Temperature 16

4.3 pH 17

4.4 Water Level 17

5 Control Systems **17**

5.1 pH Adjuster 18

6 Software Application **18**

6.1 Technology 18

6.2 Implementation 19

6.3 Use Case 19
6.4 Screens 20
6.4.1 Sign In and Sign Up Options Screen 20
6.4.2 Sign In Screen 21
6.4.3 Sign Up Screen 22
6.4.4 Home Screen 23
6.4.5 Notification Screen 23
6.4.6 User Information Screen 23
6.4.7 Add Aquaponics Screen 24
6.4.8 View and Control Specific System Screen 24

7 Assembly **25**

1 System Overview

Since our project is designed to work with both hydroponic and aquaponic systems, this section provides an brief introduction to both hydroponics and aquaponics. Afterward, the purpose of this project is given, and the high-level system design is described.

1.1 Hydroponics and Aquaponics

Hydroponics is a method of agriculture where plants are grown in a nutrient-rich aqueous solution. Nutrients in hydroponics are inputted by the grower into the aqueous solution to nourish plants. This contrasts traditional forms of agriculture, where plants are grown in soil.

Aquaponics is very similar to hydroponics. In aquaponics, plants are also grown in an aqueous solution, however plants recieve nutrients in a different manner. Instead of growers inputting nutrients, plants recieve nutrients from fish waste. To make this possible, fish are grown in separate tanks from the plants, with a filtration tank, and sump tank between them. Aquaponics recieves its name because of this, by being a mixture of aquaculture and hydroponics. A simplified diagram of aquaponics is show in Figure 1. Note that hydroponics systems look nearly the same, minus the fish tank.

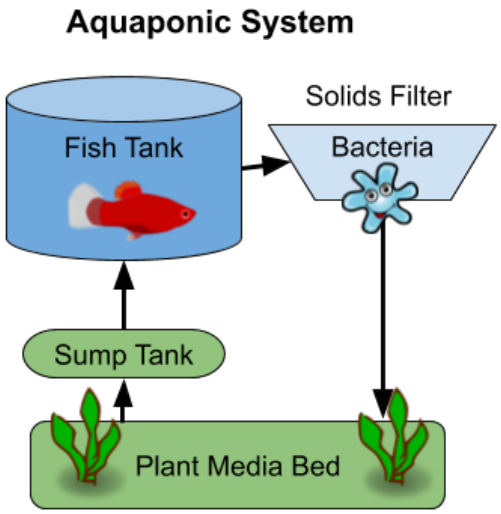

Figure 1: A diagram of a basic aquaponics system

1.2 Project Purpose

The goal of our project is to develop a system to monitor and control critical variables for hydroponic and aquaponic systems. In particular, our project focuses on monitoring water variables as they are most difficult to maintain at healthy levels in aquaponic and hydroponic systems. These variables include water pH, level, leakage, and temperature.

In terms of users, this system will be designed for small scale and hobbyist growers who maintain their aquaponic or hydroponic systems. As such, the system will utilize open-source and low cost technologies wherever possible to minimize barriers to usage and replication.

Please refer to the included *Requirements* document for further details about system requirements, background, and purpose.

1.3 High-Level System Design

Figure 2 below provides an overview of our design. As you can see, our design contains the following four subsystems: sensors, control, IoT, and software application.

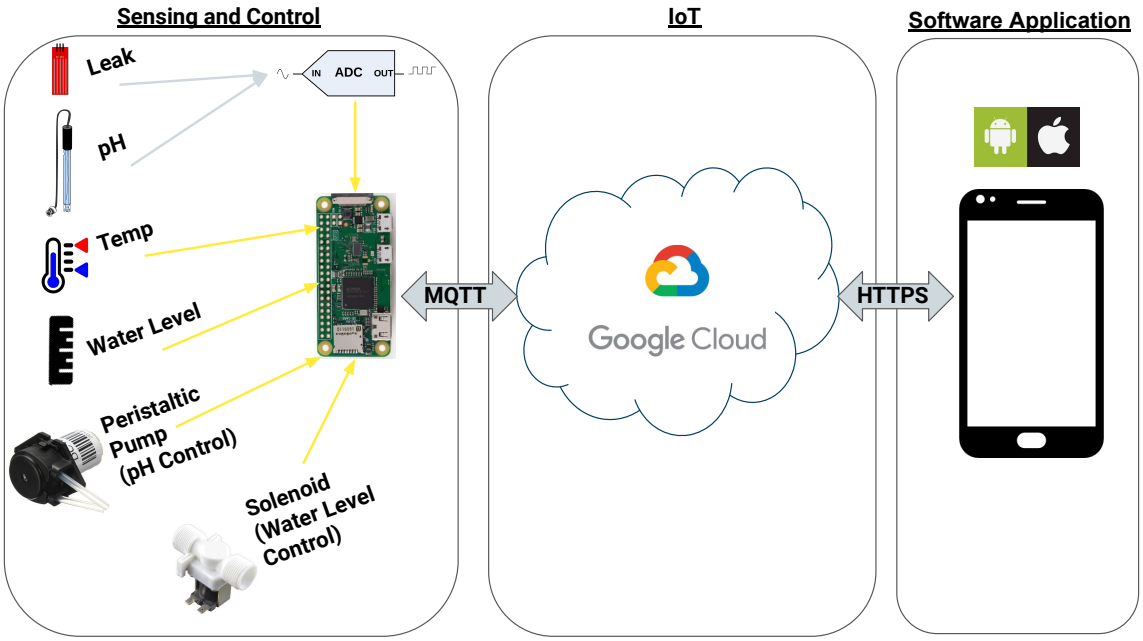

Figure 2: System Level Block Diagram

Starting from the left-hand side of Figure 2, we can see both the sensor and control subsystems. This consists of a microcomputer attached to multiple sensors and control devices. As previously mentioned, there are four key sensors in our project

shown: the water leakage, pH, temperature, and level sensors. These sensors allow our design to take measurements of the water of aquaponics or hydroponics systems. As such, the pH, temperature, and water level sensors are placed inside water tanks in an aquaponic or hydroponic system.

As far as the control systems are concerned, two devices are attached to the microcomputer: a peristaltic pump and a solenoid. These two devices allow our design to automatically adjust pH and water level respectively.

In the middle of Figure 2, we also see the IoT subsystem. This system's responsibility is to transfer data between a mobile software application and the sensing and control system. It also stores past sensor measurements for later analysis.

Finally, on the right hand side of Figure 2, we see the software application. This is where the user primarily interacts with their hydroponic or aquaponic system. This application displays past and current sensor measurements. It also allows the user to recieve alerts if urgent issues are detected by the sensing and control system, including issues like power failure and dangerous water pH levels.

In the following sections, the design of each of these subsystems will be explored in further detail.

2 Internet of Things (IoT)

The IoT subsystem recieves its name from the design decision to use internet communication to communicate with sensors and control devices installed on aquaponics or hydroponic systems. This enables our design to leverage advantages of internet communications. One key advantage is that the vast majority of our users have internet connected devices including laptops and mobile phones. Another key advantage is the ability for growers to monitor and control their aquaponics systems from anywhere they have internet connectivity.

The IoT subsystem, in our project, refers to all hardware and software needed to communicate between sensors and output devices and the software application. See Figure 3 for an overview of our IoT subsystem design.

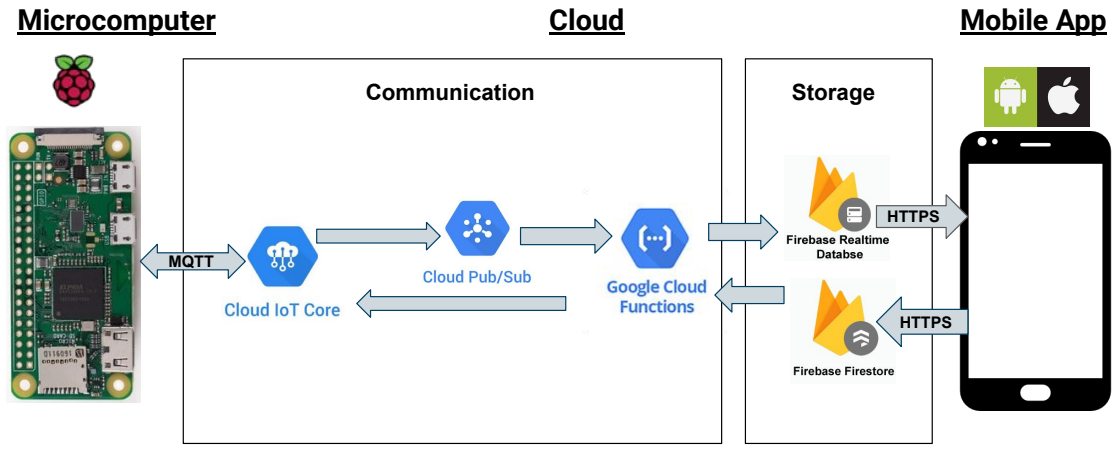

Figure 3: IoT system overview

As seen in Figure 3, our IoT subsystem has 3 major components: the microcomputer, cloud, and mobile devices. The microcomputer and cloud components will be discussed in detail within this section. In the Software Application section, the mobile app component will be discussed further.

2.1 Microcomputer

There are two key functions that the microcomputer serves in our design. The first responsibility is to receive data from sensors, and to control output devices. The second responsibility is to send sensor readings to users, and to recieve commands for output devices from users. For each of these key functions, our team has made chosen design solutions to help address these issues. Consult Table 1 for a description of the key functions and their associated technological strategies.

Key Function	Design Solution	Reason(s)
Communicate with sensors and output devices	Wired connection (e.g. SPI, IIC, GPIO), or wireless connection (e.g Bluetooth) depending on the sensor or control device	• Allows the flexibility to interface with many different sensors and output devices.
Communicate with user devices	WiFi internet network connection	• Low cost compared other alternatives like LTE • High accesibility as hobbyist and small-scale growers typically have a WiFi network setup within range of their system.

Table 1: Description of microcomputer key functions and design strategies

With these key functions in mind, our team selected a capable microcomputer. This microcomputer is the Raspberry Pi Zero W [1], shown in Figure 4.

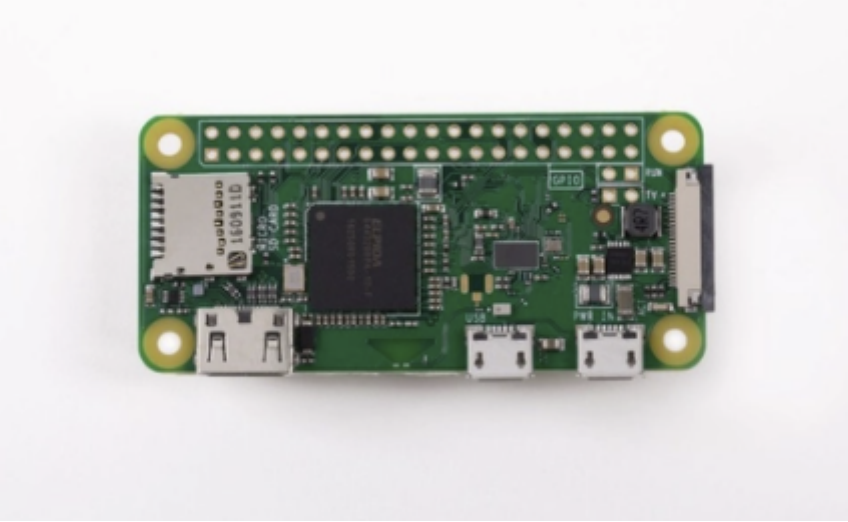

Figure 4: The Raspberry Pi Zero W microcomputer

The Raspberry Pi Zero W was chosen over other microcomputers that could perform the same key functions. This was done for the following reasons:

1. **Low cost:** costs around 40 CAD.
2. **Wireless networking:** supports both WiFi and Bluetooth.

3. **Powerful:** runs a Linux-based operating system called Raspberry Pi OS.
4. **Documentation and support:** Raspberry Pi is a popular platform with extensive tutorials, resources, and forums.
5. **Interfacing:** supports a variety of wired protocols including SPI, I2C. Also has 40 GPIO pins to interface with multiple devices.

Other notable microcomputers that were considered ESP32-based microcontrollers, and the Raspberry Pi 3B. However, the ESP32 was not chosen due to its more limited documentation and inability to run a linux-based operating system (making development slower). The Raspberry Pi 3B was not selected because of its similar functionality to the Raspberry Pi Zero W, but its higher cost of about 80 CAD.

Nevertheless, a final benefit of chosing a Raspberry Pi Zero W is that its software is compatible with other Raspberry Pi models, like the 3B. This gives design flexibility to optionally use more powerful or improved Raspberry Pi models other than the Raspberry Pi Zero W.

2.2 Cloud

The Cloud component of our IoT subsystem sits in between the microcomputer. Cloud in our project refers to cloud computing. This constitutes all the internet connected servers and associated IT infrastructure needed to communicate between the microcomputer (MCU) and the user's mobile devices. As such, the cloud is responsible for the following:

- Relaying information between the MCU and user's mobile devices
- Storing past sensor measurements
- Providing a software interface to the user's mobile devices so they can access past sensor data and communicate with the MCU.
- Scaling to allow monitoring and control of multiple aquaponics or hydroponics systems from a single mobile device.

As such, the cloud contributes to meeting the functional requirements F_1, F_2, F_3, and F_6. Please refer to the *Requirements* document for details about these requirements.

As far as design solutions are concerned, we considered 2 key alternatives based on the above specifications. The first was using IT infrastructure as a cloud proxy to communicate between the microcomputer and the mobile application. The second choice was to communicate directly with the MCU, without any infrastructure in between. These options are summarized in Figure 5.

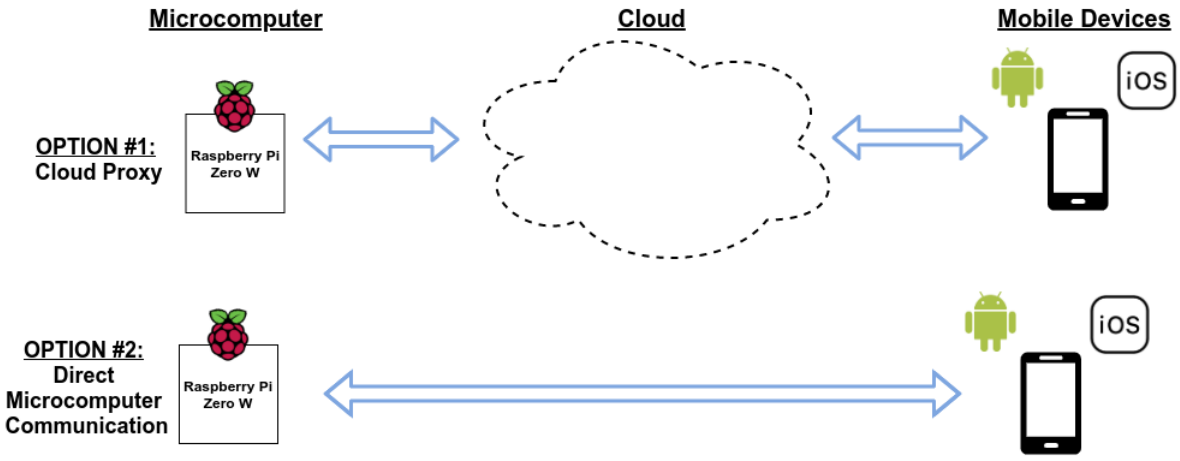

Figure 5: Two MCU to mobile application communication options.

To decide between these two main options, we weighed their strengths and weaknesses relative to their key functions. This comparison is summarized in Table 2.

Function	**Cloud Proxy**	**Direct MCU communication**
Relaying information between the MCU and user's mobile devices	**Advantage:** User devices can communicate the MCU anywhere with internet connectivity.	**Advantage:** No cost for proxy service. **Disadvantage:** User devices must be on the same local network as the microcomputer.
Storing past sensor measurements	**Advantage:** Cloud proxy services can automatically provision data storage to meet unlimited demand inexpensively.	**Disadvantage:** Microcomputers have very limited storage capacity, generally much less than 16GB. Buying extra physical storage is expensive.
Providing a software interface to the user's mobile devices	**Advantage:** Cloud proxy services can offer APIs and pre-built tools to connect with mobile devices.	**Advantage:** communicating directly with an MCU can reduce software interface complexity and maintenance.
Scaling to allow monitoring and control of multiple aquaponics or hydroponics systems.	**Advantage:** Cloud proxy services offer automatic provisioning of resources based on demand	**Disadvantage:** limited to monitoring only systems on the same local network.

Table 2: Comparison of MCU to mobile application communication options.

As seen in Table 2, the advantages of the cloud proxy option outweigh the direct MCU communication option. Most notably, the cloud proxy option offers more flexibility as it is not limited to being on the same private network as the MCU. It also is more capable of storing large amounts of past sensor data.

After choosing to use the cloud proxy option, we had a choice between 3rd party service providers, or managing our own cloud computing resources. We ultimately chose to use 3rd party IoT service providers over managing our own cloud computing resources for the following reasons:

1. **Low cost:** The estimated cost is less than 5 CAD/month.
2. **High implementation speed:** Service providers manage cloud computing resources, instead of developers needed to create and manage these from scratch.
3. **Easy maintainance:** Service providers typically provide web consoles for easy maintainability of resources.
4. **Automatic scalability:** Service providers automatically scale resources to match the amount of usage. For instance, more memory in a database is automatically provisioned.

Some popular 3rd party IoT service providers include Microsoft Azure, Google Cloud IoT, and Amazon Web Services (AWS) IoT. These service providers offer very similar services at comparable prices. So, our team decided to develop using Google Cloud IoT because it offered a better software interface to our mobile application.

This software interface will be discussed in further detail in the following sections. In addition, the architecture of our cloud system will be described in terms of two aspects: cloud communication and cloud storage.

2.2.1 Cloud Communication

Referring back to Figure 3, we see that the cloud communication component transfers data between the MCU and cloud storage. These components therefore help satisfy the functional requirements F_1, F_2, and F_3, which require communication between the mobile application and the sensing and control system.

From Figure 3, we also see that the cloud commmunication part is made of 3 key components which are Google Cloud services. These are Google IoT Core, Google Cloud Pub/Sub, and Google Cloud Functions. See Table 3 below to see the role of each of these services in our design.

Google Cloud Service	Role
IoT Core	• Provides an API for MCUs to communicate with Google Cloud using the MQTT protocol. • Provides MCU authentication and data encryption using public/private RSA keys
Pub/Sub	• Automatically recieves incoming messages from Google Cloud IoT core. • Allows Google Cloud Functions to respond to each message that is published by an MCU.
Functions	• Provides the ability to write functions that run on Google managed servers. • Writes MCU messages (with sensor data) in databases. These messages can then be retrieved by the mobile application. • Sends commands to MCUs through Google IoT Core.

Table 3: Roles of Google Cloud communication services used.

Lastly, note that the communication protocol used between the MCU and Google Cloud IoT Core is MQTT. This was chosen over other possible protocols like HTTPs as it uses lower bandwidth and has less software overhead. These two factors decrease the power consumption and data usage of our design.

2.2.2 Cloud Storage

Referring back to Figure 3, we see that the cloud storage components saves messages from both the MCU and the mobile application.

The key requirement satisfied by this section is F_2, to be able to store past sensor measurements for analysis. However, like the previous section, the satisfy the functional requirements F_1, and F_3, which require communication between the mobile application and the sensing and control system.

Table 4 also shows another two Google services that are used in our design: Google Firebase Realtime Database, and Google Firestore.

Google Service	**Role**
Firebase Realtime Database	• Saves most recent sensor measurements in a NoSQL format.
Firestore	• Stores past sensor measurements in a NoSQL format. • Stores MCU configurations.

Table 4: Roles of Google data storage services used.

Note that in Table 4, Firebase Realtime Database was chosen to save the most recent sensor measurements, and not Firestore - even though they are both NoSQL databases. This is because like the name suggests, Firebase Realtime Database is optimized to sync data with all clients each time it is updated. This feature is useful for our mobile application, as recent updates need to be displayed as soon as possible.

Otherwise, Google Firestore is used for storage of past sensor measurements. This is because Google Firestore charges a cheaper rate for storage, making it more ideal for storing large amounts of sensor measurement data.

Finally, Google Firestore also stores MCU configurations. When these configurations are updated, Google Cloud Functions automatically updates a device configuration. This is used to send commands to the sensing and control system from the mobile application. This design was chosen as it allows the mobile application to both read the current MCU configuration, and change it. That way, the mobile application can make configuration updates based on a MCU's current status. For instance, if the pH level of a given MCU configuration is below the target value, it could be changed to a higher value by updating that MCU's configuration.

3 Power

Our system is comprised of various electric components, including the Raspberry Pi microcomputer, sensors, and output devices. Power for the system will be available from two sources, a main, and a backup. The main power source is 12V DC. This is supplied by a commercially available power supply that rectifies and steps-down voltage coming from a 120V wall outlet. This will provide power to the sensor hub

during normal operation. That is, when the device is plugged in, and the line-voltage is available.

The backup power will be provided by a battery pack with 2 AA batteries, with voltage boosted to 5 V from a DC-DC boost converter. The arrangement of these two inputs can be seen in figure 6.

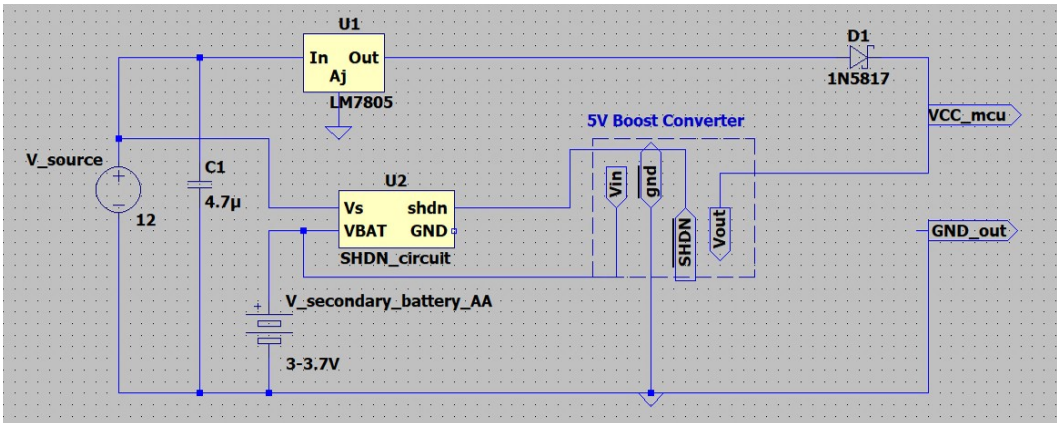

Figure 6: Uninterruptible power supply architecture.

The LM7805 voltage regulators are responsible for delivering all of the current used by the 5V devices during normal operation. It has been observed that all of the 5V devices consume around 200mA during normal operation, and the voltage regulators are rated to 1A. To increase the safety margin associated with heat-burnout, a 15g aluminum TO-220 heat sink was added to the LM7805 chip. Two LM7805 regulators are used in parallel to ensure adequate power is always available. This was also added to the Schottky diode at the output of the LM7805 chip.

To conserve the battery-lifespan, the boost converter is turned off automatically when the 12V line voltage is present. The circuit that accomplishes this can be seen in figure 7 - a simple BJT switch application. In the event of a power failure, the system will be able to run for 8 hours on backup power. This functionality is intended as a way to keep monitoring the condition of the aquaponic or hydroponic systems in the event of a power failure.

For boosting 2 AA batteries up to 5 V, the Pololu 5V Step-Up Voltage Regulator U1V11F5 is a high quality, low cost, easily accessible option that can be implemented by hobbyists with minimal difficulty. It can take anywhere between 0.5 and 5 V input and will automatically boost it to 5.02 V output.

The output voltage will be connected via Schottky diode to the line voltage in, before being connected to the Raspberry Pi. This will effectively OR the voltage source seen by the Raspberry Pi. Several other circuit topologies were simulated and tested, and the Schottky diode topology represents the best functionality in both normal operation and backup power mode.

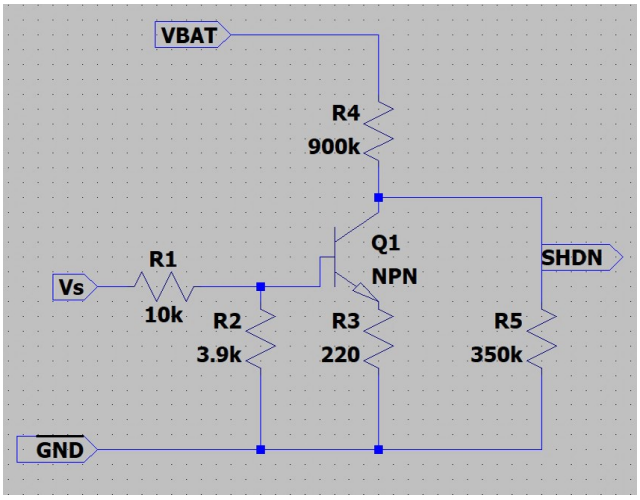

Figure 7: Boost converter shutdown circuit. This circuit disables the AA batteries when 12V is present.

This system is intended to be used with a LiPo battery pack as a backup battery source, however due to safety considerations, the implementation of this will be limited to a recommendation and design only. If one is using LiPo batteries to provide backup power, please familiarize yourself with the appropriate safety procedures associated with using LiPo technology.

For keeping a LiPo battery fully charged while plugged in, and boosting the voltage a 3.7V LiPo cell to 5V when using, the PowerBoost 1000 Charger by Adafruit is our recommended breakout board. It supports load-sharing which means the board can supply power to the hub at the same time as charge the battery. It accomplishes this with 2 special ASICs. The input line voltage can be directly input into the PowerBoost 1000, which will automatically manage OR-ing the power for the Raspberry Pi.

4 Sensors

Our design is centered around 4 types of sensors: Temperature, pH, water-leak, and water level. These sensors focus on the the status of the water in the system because this will effectively address the needs of fish, plants, and microbes, as water is the media through which these species interact.

4.1 Analog to Digital Conversion

The Raspberry Pi does not have an analog to digital converter (ADC) built in. Several of the sensors that are being used, and that future designers could use will require the use of an ADC, so we added one to our project. The ADS1115 is a 16 bit,

4 channel, programmable gain, I2C ready ADC, which can operate between 3.3 and 5 volts. These specifications meet the needs of this project by maximizing resolution, minimizing pin use, minimizing power consumption, and having sufficient channels.

Currently three of the ADC channels are being used, leaving one free for new features. The current channels in use are:

- Leak Sensors
- pH Sensor
- Battery voltage

The pins of the Raspberry Pi are designed to operate at 3.3 V, however have large enough pull-up and pull-down resistors that they can handle 5V as well. To maximize longevity, and minimize chance of pin burnout, the ADS1115 ADC is operated at 3.3V.

By operating the ADC at 3.3V, the maximum voltage it can read from one of its input channels is now 3.3V also. As discussed below, for the pH sensor, this means that a voltage divider from 5V (max) to 3.3V (max) is necessary to prevent the ADC from being damaged by the pH sensor chip which is powered at 5V.

4.2 Temperature

We considered three types of temperature sensor. We chose DS18B21 ($10) temperature sensor over the DS18B20($5) and I2C BME280($20) temperature sensor. This decision was made because:

- The DS18B21 provides temperature values every second and allows the Implementation of a system that can display values in real time. Requirement: F_1
- The DS18B21 is waterproof and the other alternatives are not. Choosing this sensor leads to a system that is water resistant. Requirement: NF_4
- The DS18B21 is in the mid price range and contributes to an affordable system. Requirement: NF_1

The temperature sensor communicates over 1-wire interface with the Raspberry Pi. The Raspberry pi operating system has convenient protocols for 1-wire interfaces that automatically receive the data from these devices and store them in a given location in memory. The script that reads the temperature, quite simply, checks this location in the memory for the temperature, and performs some simple cleanup.

4.3 pH

We considered two types of pH sensors among many and we chose the PH-4502C($39) over the LeoEc($60) because:

- The PH-4502C is a low power sensor (less than 0.5W) and will therefore provide better lifespan in the event of a power failure. Requirement: F_5
- The PH-4502C provides pH values with a delay of 0.5s. Requirement: F_1
- The PH-4502C is lower priced compared to the LeoEc and middle priced compared to other pH sensors that don't meet our requirements. This choice contributes to an affordable system. Requirement: NF_1

The PH-4502C is intended to be used by mapping pH 7 to 2.5V, 14 to 0V, and pH 0 to 5V. These values rarely seem to be present in reality, necessitating significant calibration. To prevent a 5V pH input (unlikely) from damaging the ADC, a voltage divider (R1 = 4.7k, R2 = 10k) is implemented to reduce the max voltage to 3.3V at the ADC.

4.4 Water Level

There are different ways to measure the water level. We chose the Binary Water Level Detector, SEN0204-ND ($13). The other alternatives are the Ultrasonic Sensor, HC-SR04 ($4) and the E-tape Liquid Level Sensor ($60). We made this decision because:

- The SEN0204-ND provides the minimum data to maintain a desired water-level F_1
- The SEN0204-ND is small and can scale very well to differently sized systems. This makes the sensor easier to install on many more types of aquaponics systems compared to alternatives. NF_5
- The SEN0204-ND is low cost. The HC-SR04 and other Ultrasonic Sensors have a very short lifespan and break very easily, especially in environments where water is involved. So in the long term, the SEN0204-ND contributes the most to an affordable system as it is IP67 rated. NF_1, NF_4

5 Control Systems

We considered 3 types of output devices: Air pump to oxygenate the water, Water pump to control the water levels, and the pH adjuster to control the pH levels

5.1 pH Adjuster

There are two ways to change and maintain the pH levels of the system: with a liquid solution or a solid object usually peat moss or driftwood. We have chosen to use the liquid solution and have decided to use a Peristaltic Pump ($12) to control this liquid solution due to the following reasons:

- The solids can produce discolouration in the water. This will not only make physical observation harder, a feature we might implement is a waste detecting system, and discolouration affects that. The liquid solution will not interfere with the appearance of the system and provide accurate values in real time. (functional requirement)
- The liquid solution is a lot easier to measure. The volume of a liquid is much easier to measure and transport compared to something solid. This results in a much easier installation. (non-functional requirement)
- The Peristaltic Pump is cheaper and easier to design compared to something that needs to transport a solid. This leads to an affordable system (non-functional requirement)
- There are many types of liquid solutions that can be used while the solids needed are very specific and usually hard to find. This allows users to customize more easily compared to the solids. (non-functional requirement)

6 Software Application

To implement the software application to display the aquaponic's data to the user, we considered various aspects, like the technology we were going to use, how we were going to implement said technology, and the trade-offs we made for each decision.

6.1 Technology

The two main technologies we considered to implement the software application were web and mobile. We ultimately choose to use a mobile based approach to implement our software application than a web based approach. The main factor that that determined this decision was ease of use. In a mobile based approach users would be able to quickly access the application, as user preferences are saved and tied in with the mobile application. This means that the user doesn't have to re-authenticate time after time when trying to access their aquaponics data. Another big factor we considered when choosing a software approach was accessibility. Almost everyone has their mobile devices on them at any given time during the day, which would provide greater accessibility to the application. A final consideration

that we explored was push notifications. With a mobile based approach, we are able to send push notifications to users when extremely urgent matter arises, whereas in a web based approach this isn't even an option.

6.2 Implementation

We considered many different options we could take to create a mobile application. We debated on whether or not we would want to create a native application or build one with a cross platform language. Given the time and nature of this project, we ultimately decided on a cross platform approach. With a cross platform approach, we would be able to develop for both Android and IOS with half the effort and half the code base. One of the draw backs however is that we might lose niche native functionality specific to either Android or IOS. Since our software application is simple, we believe that the niche native service loss wouldn't be detrimental to the project. We choose Flutter as the our cross-platform language to implement this mobile application over other cross platform languages such as React Nastive, or Xamarin since Flutter allows for quick implementation of native feeling applications.

6.3 Use Case

Upon entering the application, the application checks if the user has previously been authenticated. Here users can authenticate themselves by either signing up or signing in. Upon successful sign in or creation of an account, the user would be taken to the application's home screen. The home screen displays a list of the current aquaponic systems that the user has registered with the application as well as an indication of their status. From the home screen, the user is able to navigate to the notification screen, navigate to the user information screen, add additional aquaponic systems, and view and control a given auquaponic system. The notification screen is where all notifications are stored and meets our functional requirement F_3. The user profile screen is where the user is able to logout or delete their account if they so choose. The screen to view and control a given aquaponic system meets our functional requirement F_1 and F_4. An overview of how the user can interact with the software application is highlighted and illustrated in the use case diagram below in figure 8.

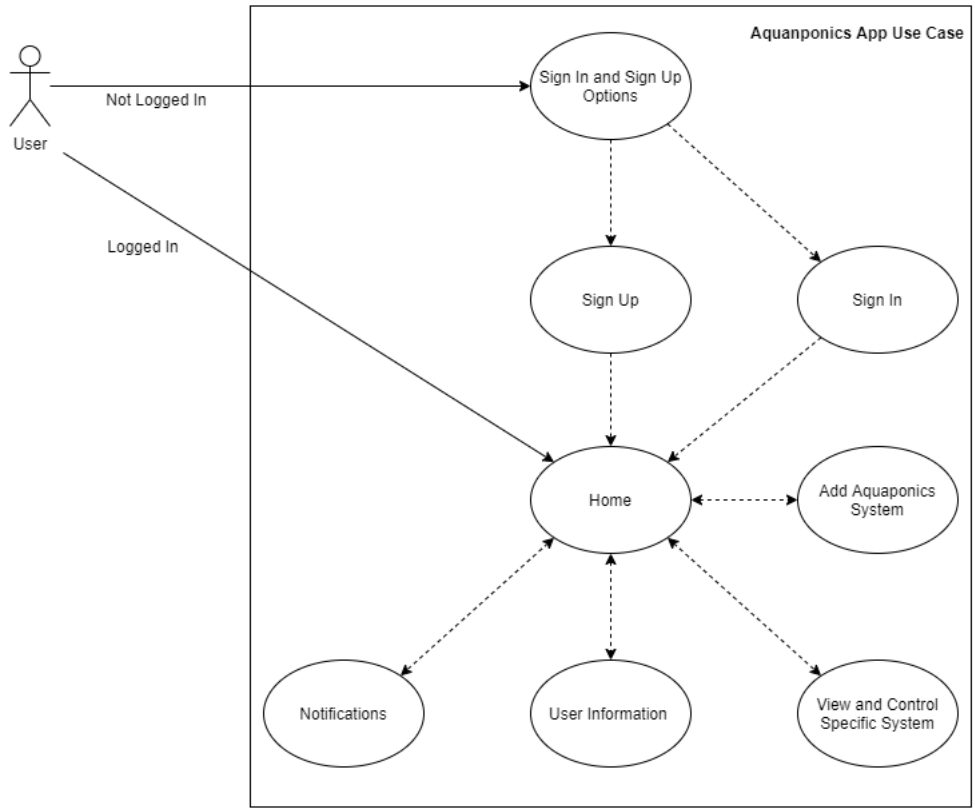

Figure 8: Use Case Diagram

6.4 Screens

Below are the different screens in the software application and an explanation of their intended user interactions as well as how they interact with the overall IOT infrastructure.

6.4.1 Sign In and Sign Up Options Screen

The sign in and sign up options screen will be presented to users who aren't previously authenticated on the application. The sign in and sign up options screen will give the user the option to either Sign Up an account or login to an existing account. The user can Sign Up with the application by providing an email and password or by Signing in using a valid Google account. An image of the application's sign in and sign up options screen can be seen below in figure 9.

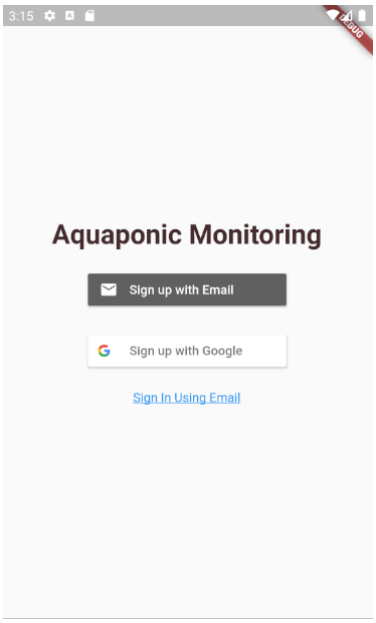

Figure 9: Sign In and Sign Up Options Screen

6.4.2 Sign In Screen

The sign in screen is presented when the user taps on the "Sign in using Email" option in the sign in and sign up options screen. This page contains two input fields for the email address and password. Unsuccessful authentication submissions won't let the user into the application whereas successful authentication submission will direct users to the application's home screen. Each login attempt will trigger a request to Firebase to confirm the users credentials and to get the user's unique User ID. The email input box verifies that it's a valid email address by checking if the '@' character is present within the email submission. Similarly password input box verifies that the password is sufficient by the password length is at least 6 characters. An image of the sign in screen can be seen below in figure 10.

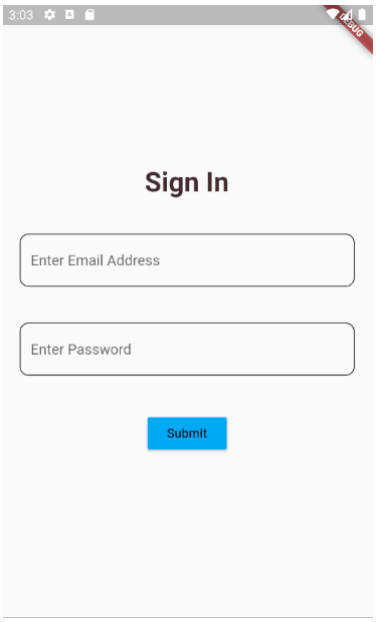

Figure 10: Sign In Screen

6.4.3 Sign Up Screen

The sign up screen is presented when the user taps on the "Sign up with Email" option in the sign in and sign up options screen. This page contains two input fields for the email address and password. Unsuccessful sign up submissions won't let the user into the application whereas successful sign up submission will direct users to the application's home screen. Each login attempt will trigger a request to Firebase to create the user account by using the submitted user credentials and to get the user's unique User ID. The email input box verifies that it's a valid email address by checking if the '@' character is present within the email submission. Similarly password input box verifies that the password is sufficient by the password length is at least 6 characters. An image of the sign up screen can be seen below in figure 11.

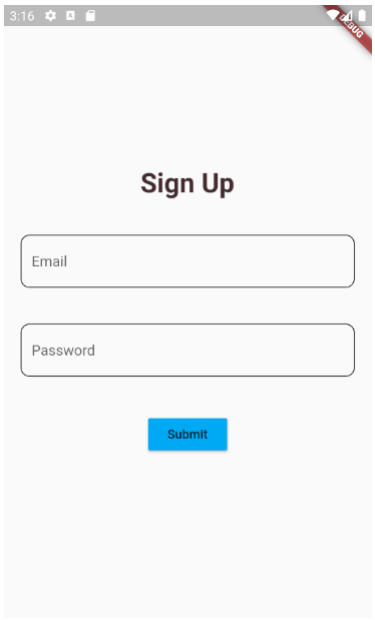

Figure 11: Sign Up Screen

6.4.4 Home Screen

The home screen is presented when the user successfully authenticates with the application. The home screen shows a list of the user's existing aquaponics systems that are registered with the application as well as a quick icon indicating the overall heath of the aquaponics system. Upon tapping a given aquaponic system from the list, the user will be taken to the view and control screen of that specific system. The home page as a bottom navigation bar with icons that allow for navigation to the home screen, notification screen, and user profile screen. The bottom navigation bar also contains a floating button that allows user to add additional aquaponics systems to the application.

6.4.5 Notification Screen

The notification screen is presented when the user clicks the bell icon in the bottom navigation bar. The home screen shows a list of the user's pending notifications that usually pertains to the health of their aquaponic systems. The notification screen can be accessed as well when the user taps on a notification that is sent to them.

6.4.6 User Information Screen

The notification screen is presented when the user clicks on the user icon in the bottom navigation bar. The user screen shows the user information and allows for

the user to log out of the application or delete their account.

6.4.7 Add Aquaponics Screen

The add aquaponics screen is presented when the user clicks the profile icon in the bottom navigation bar. The add aquaponics screen contains input fields that pertains to the aquqponic system they are trying to add. Upon successful submission of results, the aquaponic system will be created in the Firebase back-end and linked with the users User ID.

6.4.8 View and Control Specific System Screen

The view and control specific system screen is presented when the user taps on a specific aquaponic system in the home screen. This screen contains real time accurate data of the different measurements taken by the sensors of the aquaponic system as well as buttons to request value readings. The color of the buttons represents whether or not the specific data is within an acceptable range; green means that the data is within an acceptable range and read means that the data is within an unacceptable range. This screen also contain buttons to control the different actuators that affect the system such as the water pump. An image of the view and control specific system screen can be seen below in figure 12.

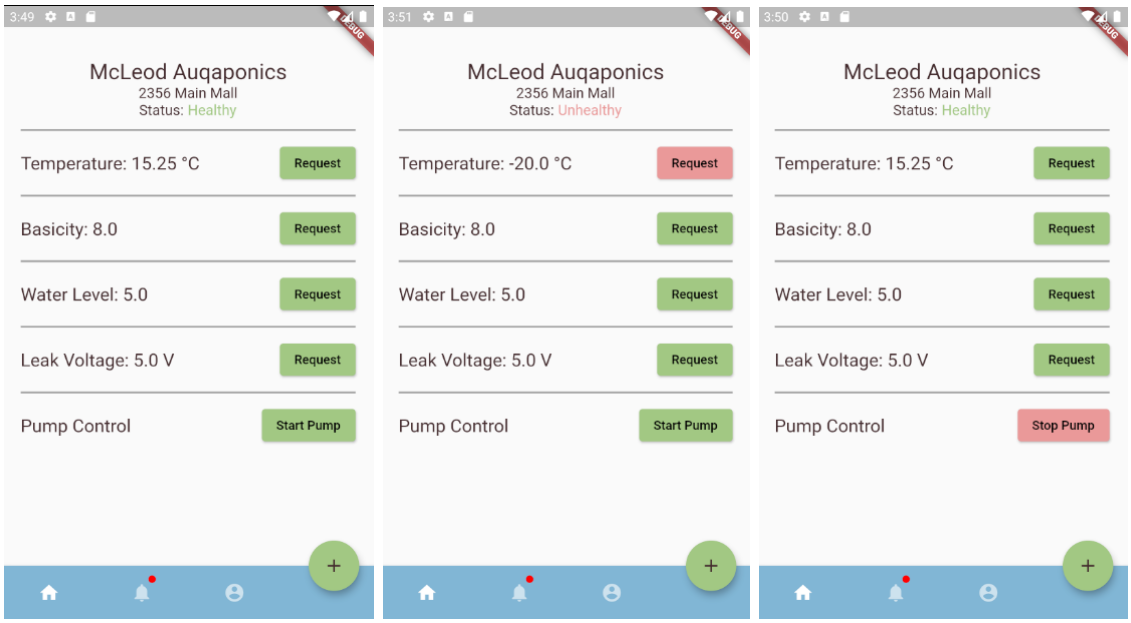

Figure 12: View and Control Specific System Screen

7 Assembly

In consideration of the construction of the system, the NF_4, NF_1 requirements were chiefly considered. That is, that the design is waterproof and low-cost.

All components were roughly modelled in Solidworks 2019 to gauge how large the enclosure needed to be. Models of all necessary components can be seen in figure 13 and 14.

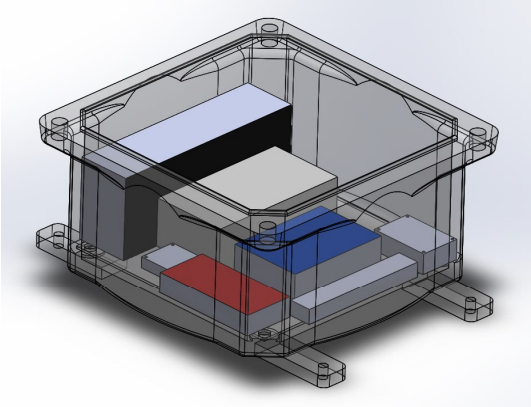

Figure 13: Enclosure elements inside enclosure: Power Supply, Breadboard, Relay Block, Terminal Block, Raspberry Pi, Water-level board, pH sensor board.

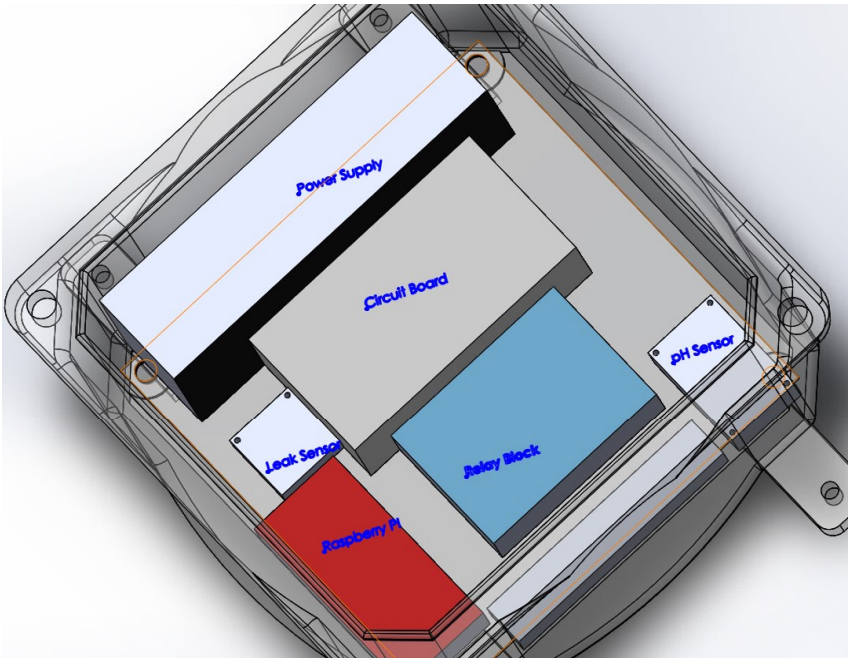

Figure 14: Labelled Enclosure elements: Power Supply, Breadboard, Relay Block, Terminal Block, Raspberry Pi, Water-level board, pH sensor board.

There are 5 independent wire ports that must penetrate the enclosure in a waterproof manner. To accomplish this, waterproof nylon cable glands were used. An example

of one of these can be seen in figure 15. These devices have a rubber seal which is compressed around a wire inside it, forming a seal.

Figure 15: Waterproof Nylon Cable Gland example.

For ease of maintenance and durability, all of the electronic components are mounted to a board that will press into the corners of the enclosure securely. For our demonstration, this board is made from recycled 1/2" plywood.

References

[1] Raspberry Pi Zero W.
https://www.raspberrypi.org/products/raspberry-pi-zero-w/?resellerType=home