LeafIt!

An Autonomous Watering System

University of Central Florida

EEL 4915: Senior Design 2 Fall 2022

Final Document

Group 9

Kyle Eaton: Electrical Engineering

Adam Keller: Computer Engineering

Francis Olearcyzk: Electrical Engineering

Matthew Ongcapin: Computer Engineering

Table Of Contents

List of Figures

List of Tables

1.0 Executive Summary

The cultivation of plants is arguably one of the primary reasons humans have been able to grow and evolve from a tribal hunter-gatherer lifestyle, to growing their own foods. This technological advancement led to the creation of societies, population growth, and allowed humans to focus on things other than survival. Modern agricultural practices have advanced to the degree that the average person no longer needs to grow food to survive. The itch to grow and cultivate plants still remains for many people, and today anyone with nearly no experience can attempt to grow flowers, herbs, vegetables, and fruits in their own home. Although fairly straightforward, growing plants consistently is not always so easy. In today's society everyone loves to go on vacation or travel, however if you grow plants at your house this can be troublesome if you are gone for an extended period. You either must rely on a neighbor or friend to water your plants or hope that the rain waters them enough while you are gone. The practice of overwatering houseplants, destroying herb gardens, and believing you have a black thumb are over. It is also very easy to just forget to water your plants. Our project, the autonomous watering system: LeafIt! solves this issue. You no longer need to worry about your plants while traveling or at home.

According to recent statistics, seven in ten millennials refer to themselves as a plant parent. The average plant owner has killed seven houseplants, and the average household spends over $600 a year on gardening goods. With the recent COVID-19 pandemic, houseplant demand has grown by 18% due to the fact that more people were forced to work from home and or layed off from their jobs. Given that the plant and flower industry market size has reached over 15 billion dollars in 2020, it is no wonder that the majority of Americans have adopted the practice of owning houseplants. It has been proven that having houseplants improves productivity, reduces stress levels, and house plant owners take fewer sick days. Along with this practice comes the trial and error of killing several plants in the process. That does not have to be the case any more. With our new technology LeafIt! We will be able to suggest and automate better watering habits, allow the user to understand their plant better, and provide remote monitoring of soil levels and inform the user when their plant has been watered. Gone are the days of removing the dead plants from your windowsill.

The only disadvantages with owning houseplants are the worries of how you can keep your plants alive when you are busy with work, or on vacation, or you are unable to care for them. Of millennials, 48% of them are afraid of being able to keep their house plants alive. Being able to have a reliable and consistent monitor of your house plants will put your mind at ease. This could also be another use case for elderly or disabled people, where plant care might not be the easiest thing to do. Our project will water and monitor your plants for you, all you need to do is the initial setup and refill the water reservoir.

2.0 Project Description

This section contains discussion on the actual function of the project, a project block diagram demonstrating the team responsibilities in relation to the full system, a full list of goals, a house of quality diagram, and a list of engineering specifications for testing. The overall purpose of this section is to provide a general overview of this project in order to provide sufficient context for the reader.

2.1 Project Objectives

The project objectives will make clear the motivation for this project as well as the main goal. It will answer the reason why this project was chosen and why this project will be unique when compared to similar systems of the same category. The system scope will be derived from the ethos described below.

2.1.1 Motivation

The primary motivation for this project and idea is based around showcasing our skills, knowledge, and critical thinking gained while attending the University of Central Florida. This project has several parts that allow interdisciplinary work between computer engineering students, and electrical engineering students. Given the span of knowledge required this project will allow our group to work together as a team, to accomplish a goal of building, testing, refining, and producing a product that has real world application. This will also be a step in obtaining a bachelors degree in either computer or electrical engineering, based on ABET accreditation. Another benefit this project entails is allowing group members to have an informative and interesting story to tell to future employers, as much of the engineering job market wants to see what you can do rather than what you know about. The experience is invaluable, as it will grant insight to how projects are formed and executed as well as reveal common pitfalls in projects like these, which future employers will appreciate these types of experiences in their new hires.

2.1.2 Primary Objective

The primary objective of this project is to create a system to monitor/water your plants while you are away on vacation. The system will use sensors to gather the temperature, moisture levels, water levels, etc. Any of these sensors can trigger a notification to be sent to the user. For example, if the water reservoir is low a notification will be sent to the user to refill the water. To complement the user experience we plan to develop a phone app with an interface to use the watering system. The phone app will alert you when the water reservoir is low, a plant needs more sunlight, the moisture or temperature is off, etc. Two aspects of our project make it unique. Firstly, during our product research we noticed that most automatic watering systems were either a manually programmed pipe irrigation system or a framed hydroponics system controlled from the user's phone. **Our**

project is unique in that it is both a simple pipe irrigation system which can be applied to already existing gardens and a system that is controlled by the user's phone. Secondly, our product research informed us that many of these systems already on the market do not specify the amount of water based on the type of plant being watered. **Our system is unique in that the users can optimize the care of their plant based on three types of plants of their choosing (Tropical, Temperate, and Arid).** Each type of plant has specialized watering programming so that the plant does not get under or overwatered.

2.2 Discussion of the function of the project

The function of this project is to create a system to monitor and take care of plants autonomously. All the user needs to do is refill the water reservoir. We will make a phone app for easy user interface and ease of use for when you are away from your house. If a plant is still not doing well, even with the watering, the user can check the long term statistics of the plant's measured environment. From there, the user can see if the plant does not receive enough sunlight or if the temperature is outside of the optimal range of the plant's desired temperature.

2.3 Tiered Goals

The following puts goals into three categories:basic, stretch and advanced goals. Basic goals will need to be met, stretch goals will be met if time permits, and advanced goals will be met when all other systems are fully functional.

1. **Basic:** The system shall automatically water one outdoor plant using a water pump with amounts of water based on one of the three possible types of plants that the user inputs, the temperature, and the moisture present in the soil of the plant.
2. **Basic:** The power system shall be capable of delivering at least 12V DC.
3. **Basic:** The power supply shall be capable of powering the sensors.
4. **Basic:** The total cost of the system shall not exceed $250.
5. **Basic:** The system shall have two sensors to monitor soil moisture of the plant and temperature.
6. **Basic:** The system shall support a container of water at least 0.5 Gallons and at most 3-5 Gallons.
7. **Basic:** The system shall use a water pump to draw water from a reservoir of water outside of the system i.e., a bucket or pot of water.
8. **Basic:** The system shall not weigh more than 3 pounds.
9. **Basic:** The power system shall have current leakage protection.
10. **Basic:** The microcontroller must be able to produce a PWM signal that has the amplitude of 1V max and a minimum of 1V.
11. **Basic:** The frequency that the microcontroller must produce on the PWM signal must be able to operate on a range of 50Hz up to 100Hz.

12. **Basic:** The minimum operation voltage of the microcontroller must be at least 3.3 volts.
13. **Basic:** The program of the system must be able to compile and correctly output the expected actions deemed by the user input.
14. **Basic:** The system shall have adequate heat sinks for device and user protection.
15. **Stretch:** The system shall have an interface for users to choose soil moisture and temperature settings based on three types of plants.
16. **Stretch:** The system's watering and temperature settings shall be based on three types of plants: Tropical, Temperate, and Cactus/Succulent.
17. **Stretch:** the system will be able to water multiple different plants with different settings.
18. **Stretch:** The system shall have a spray nozzle to better control the amount of water supplied to each type of plant.
19. **Stretch:** The system shall have waterproof casing to keep moisture from damaging the electronics.
20. **Advanced:** A motorized system designed to switch between watering 3 different plants one at a time.
21. **Advanced:** The user shall be able to input into the system "vacation days", prompting the system to calculate if the water reservoir is passable given the type of plant and the number of vacation days.
22. **Advanced:** The system shall alert the user's Android phone of inclement weather, unfavorable amounts of moisture in their plant as well as updates on the plant when it has been watered.
23. **Advanced:** The system shall alert the user's Android phone when the plant suffers from unfavorable climate using the temperature sensor.
24. **Advanced:** The system shall have a light source to provide light to the plant when the temperature sensor senses a cooler temperature than what is optimal for the growth of the plant.

2.4 House of Quality Diagram

The house of quality diagram is a concise conglomeration of engineering and customer specifications, how they are related, and their level of importance. This can be used to draw conclusions about the importance of each system and how they will have a push or pull relationship. This means that improving one system will have negative consequences on another system. This can be seen below in Figure 1:

		Column #	1	2	3	4	5	6	7
		Direction of Improvement							
	Weight	Customer Requirements (Explicit and Implicit) / Engineering Requirements	Water Pump	Size of Product	Cost	Power	Temperature Sensor	Moisture Sensor	Phone Interface
	4	Cost	○	●	●	○	○	○	▽
	3	Size	○	●	●	○	▽	▽	▽
	1	Usability	●	▽	▽	●	●	●	●
	2	Stability/Reliability	●	▽	▽	●	●	●	●
		Target	250ml a day	2ft x 3ft x 2ft	no more than $250	No more than 24kWh and able to produce 12V, 5V, 3.3V	accurately measure between 0C and 40C with a resolution of 0.2C	0mm-500mm of water per 1m of soil	track values over 3 weeks

Relationships	
Strong	●
Moderate	○
Weak	▽

Figure 1: House of Quality Diagram

2.5 Specifications

Below shown in Table 1 is a list of engineering specifications which will be guaranteed to be in the project. These were chosen after consulting all of the group members and doing research on what is feasible to implement.The specifications highlighted in yellow will have a live demonstration to show our project and its functionality. These specifications were designed to easily be able to be demonstrated in at most 5 minutes. Practically, most of these specifications will happen over a larger time frame to save on water and power, as the scope and scale of the project does not necessarily require fast acting or fast reporting components to operate correctly. These engineering specifications can help verify our project is working as intended and can easily demonstrate the purpose of it.

Watering	Pump will provide 5mL-10mL of water in less than 30 seconds
Volume	Product will not be taller than 2ftx3ftx2ft
Cost	Product will not cost more than $250
Power	Product will not use more than 24kWh and be able to produce 12V, 5V, and 3V
Temp Sensor	Product will accurately measure temperatures between 0C and 40C with an accuracy of 0.75C
Moisture Sensor	Product can detect a difference of 5mL of water in 7 cm^3 of soil
Web Server: Long Term statistics	The web server will record values received from the MCU within 15 seconds

Table 1: Engineering Specifications

2.6 Project Block Diagram

The project block diagram below in Figure 2 details the overall breakdown of the system and the personnel responsible for a particular part of the system. The interactions between each main section can be followed with arrows.

Although each person mainly focused on their section of the system, plenty of coordination took place to integrate all of the systems. The points of integration that are the most important are the arrows between different people's systems. Power and information will have to be able to freely flow to where it needs to go. Stability is also an important factor to consider when combining the systems.

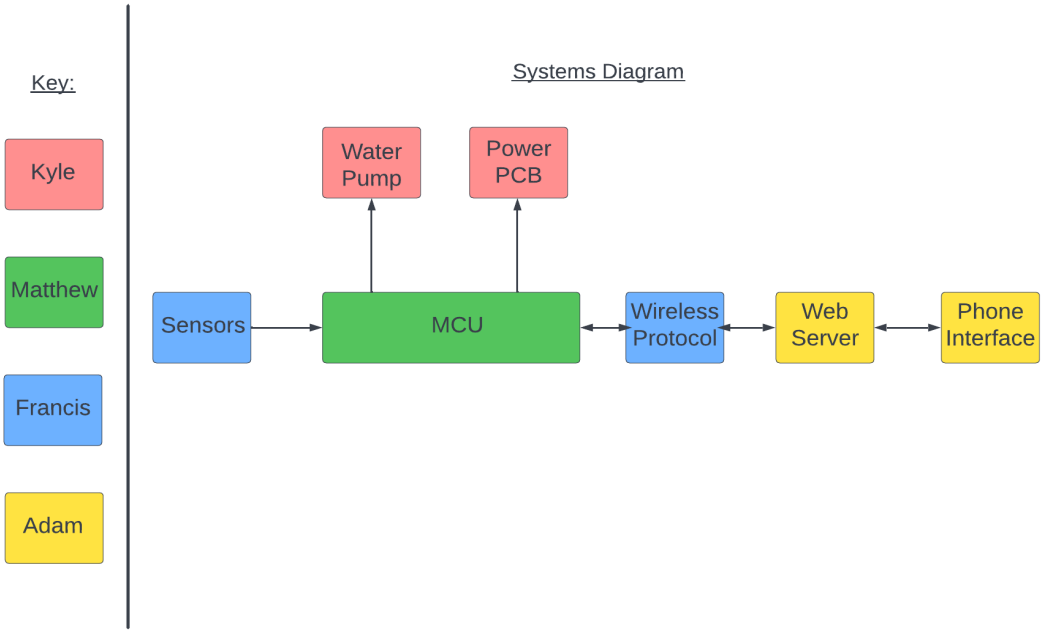


Figure 2: Project Block Diagram

3.0 Research and Project Selection

In this section, we discuss the initial product research that the team underwent in order to understand the most applicable products currently in the industry, a project prototype illustration to provide a visual of the overall design, descriptions of various prototypes that were conducted as research, and a breakdown of the different researched components chosen to construct the project.

3.1 Similar Product Research

Expanding upon previous iterations of innovation and development has allowed the growth and changes that are necessary for continued growth in the field of engineering. We were able to find several DIY kits that are similar to our project idea, and some more commercialized options. It was important for us to explore the wider industry of home irrigation kits so that we could have a better understanding of potential designs for our project. Though we may have a particular perception for how to solve the problem and subproblems of home irrigation, it is important for us to consider the solutions already present in the industry as well as any common and unique features that are present.

Upon further investigation, it appears that most of the solutions available in the wider industry use simple pressure or gravity based irrigation systems. These systems work by taking advantage of simple physics: the system forms an air vacuum with the soil. A certain amount of moisture in the soil, air inside of the system, and water inside of the system creates a pressure equilibrium that traps the water inside of the system. When the amount of moisture in the soil decreases to a certain point, this causes a difference in pressure which forces water from the system to enter the soil until a pressure equilibrium is created again. This whole system works without the use of electric parts. Ultimately, we did not pursue this solution because the skills required to construct such a device are above our station. It requires advanced knowledge of glassware and metalworking due to the geometry of the system being integral to creating a pressure equilibrium necessary for the function of the system. Also, though a purely pressure based system may lack complexity for the user and lacks the dangers of mixing moisture with electronic parts, not including electronic parts creates a huge lack in features that could possibly be useful to the user.

Due to our expertise in electrical engineering, computer engineering, and computer science, our analysis of the automatic irrigation problem suggested to us that features enabled by electronic parts such as timed watering, plant specific watering, and plant monitoring would not only be a relatively easy task to conceptualize, but also would be of greater use to users than the sheer simplicity of pressure based automatic irrigation systems. Our methodology of our investigation basically focused on systems that utilized electronics, sensors and embedded systems to provide greater functionality and value to their users. The results of our industry research revealed to us that, other than pressure based automatic irrigation systems, the automatic irrigation system market consists of

“Snip n Drip” watering systems as well as numerous 3D printer chassis inspired Internet of Things DIY watering systems. Through our expertise, we are aware of the concept of the Internet of Things, which enables greater functionality and value to users through the use of embedded systems, Internet networks, and software applications in different appliances. The “Snip n Drip” watering systems greatest strengths are their ease of use, relatively lower cost, and relatively easy setup. The DIY watering systems were interesting to us as well because they integrate the use of the smartphone as a tool to monitor the health of the user plants, which is a feature that is not present in most automatic irrigation systems already established in the industry.

3.1.1 Snip N Drip Automatic Irrigation Systems

An example of this automatic irrigation system that we came across in our product research can be seen in Figure 3. These systems provide great value to their users through their easy setup and relative low cost. Also, these systems are often expandable based on the amount of rubber or latex tubes attached to the system. The functionality of these systems are very similar to a common suburban garden hose. However, instead of using a faucet to create a difference in water and air pressure to prompt the creation of a stream of water, the difference in pressure is created by a water pump submerged in a separate source of water. When prompted by the system, the water pump creates the difference in pressure necessary to create a stream of water to irrigate indoor and outdoor plants. Most of these systems usually do not have embedded systems capable of advanced programming and complex sensors. Instead, the system relies on basic electronics to take in basic user commands through the use of buttons. Once the user has entered the appropriate commands into the system, the system will automatically water the plant. One downside of this is overwatering that could be dangerous to the plant due to the system never “sensing” the actual moisture needs of the plants. Instead, the system operates on timed intervals entered by the user which prompt the irrigation. Nevertheless, some of the base designs and features of this system were taken into consideration in our design.

The chart below is a chart summary of the different systems investigated by the team. This includes gravity based irrigation systems, pressure based irrigation systems, Snip N Drip systems, and IOT systems. Each has a differing level of complexity and effectiveness. The more complex ones will be more expensive and take more time to develop, while the simple ones will be easy to implement and cheap, but will most likely be limited in effectiveness and features that the user might want.

Types	Parts	Function	Amount of Plants Watered	Effectiveness
Gravity Based Irrigation Systems	A large pressurized metal bin of water, vinyl tubing	The large pressurized metal bin of water is placed above the plants, changes in the soil moisture creates a change in pressure prompting watering	Various	Effective but unable to control and customize the amount of water
Pressure Based Irrigation Systems	A large pressurized metal bin of water, vinyl tubing	The large pressurized metal bin of water is placed above the plants, vinyl tubing is “pinched” to create a vacuum, changes in the soil moisture creates a change in pressure prompting watering	Various	Effective but unable to control and customize the amount of water
Snip n Drip Irrigation System	Water pump, power source and accompanying electronics, container for the source of water, vinyl tubing	Water pump provides water to plants through vinyl tubing	Various	Very effective
IoT Automatic Irrigation System	Water pump, chassis, power source and accompanying electronics, container for the source of water, vinyl tubing	Water pump provides water to plants through vinyl tubing	Various	Very Effective

Table 2: System Study

Figure 3 below shows an example of the Snip N Drip system. It takes water from one central reservoir and distributes it to multiple different plants. Each plant is connected to the same watering system, so the system will distribute the same

amount of water to each plant. This will not be desirable if each plant requires different amounts of water to use. With this system, each plant would have to at minimum receive enough water to sustain the most water reliant plant in the ecosystem. Although simple, this system can lend itself to lots of wasted water and the potential to drown plants.

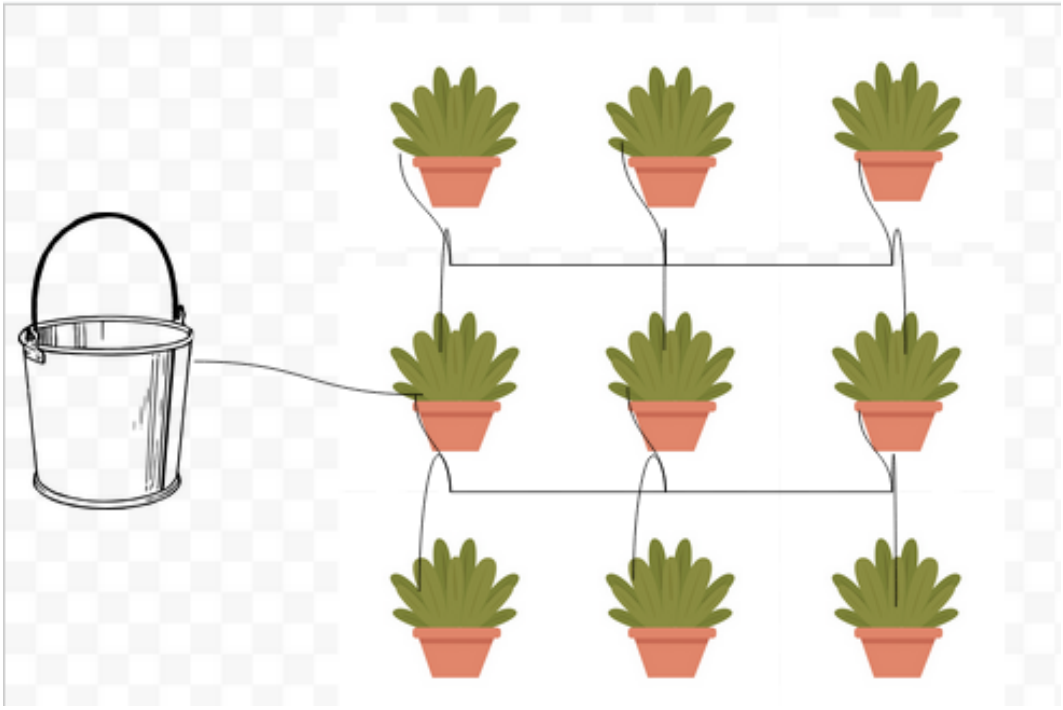


Figure 3: Example Diagram

3.1.2 IoT Automatic Irrigation System

An interesting addition to our industry research were DIY projects that utilized and integrated IoT technology into their design. A downside of the previous automatic irrigation systems in our research was that they often lacked higher levels of control that would greatly benefit the user in solving the automatic irrigation system problem. This is probably due to the addition of features in these systems being dependent on creating more physical parts that would increase the overall price of the system for the user. These older designs probably settled on the features they currently utilize based on the balance between features and overall cost of the system. However, IoT technology can increase the amount of useful features through relatively cheap electronic components and sensors through the use of a network connection; these DIY projects are a good example of such a solution. The example below uses a frame and motor to irrigate individual plants. Through the use of their phone and the software present in a Raspberry Pi microcontroller, individual plants can be watered in different amounts based on the user preference. Also, different sensors are utilized to monitor moisture or temperature to prompt automatic irrigation of these plants. The actual watering of the plants works on the same

water pump and tube process present in other automatic irrigation systems but instead, it is controlled by higher level programming on a microcontroller rather than lower level programming based off of a timer in the previous automatic irrigation systems.

One potential flaw in this DIY example design is the use of a 3D printer inspired chassis frame. In an industry view, this construction would probably increase the overall cost significantly compared to “tube only” irrigation systems. However, this could also potentially provide aesthetic value to the consumer. Another puzzling design choice is the use of the 3D printer inspired motorized “arm” to move the irrigation pipe to multiple different kinds of plants. Again, in the industry, the use of multiple motors would probably increase the overall cost as well as the strain on the power source. Also, this seems to be a case of over design. The use of motors to irrigate individual plants seems to be a very energy expensive solution to a relatively simple problem. As seen in previous automatic irrigation systems, it is much easier to irrigate plants through the use of multiple tubes that can water the plants individually. Nevertheless, the strengths of this design were integrated into our design as well, particularly the use of IoT concepts and designs. The Internet of Things provides a balance due to the integration of lower cost components and increased functionality. The striking differences between the DIY examples and “Snip n Drip” designs prompted the creation of three prototypes. The first prototype focuses on implementing a “Snip n Drip” automatic irrigation system. The second prototype implements a “Snip n Drip”automatic irrigation system but utilizing a frame similar to what is present in the DIY systems. The third prototype is a simple class based Python program to describe the functionality of the irrigation system. The purpose of this is to explore a potential software based solution to the issues associated with irrigating multiple plants, avoiding the use of a motorized arm for irrigation, and the use of multiple pipes to water multiple plants. The diagram below shows an example of the DIY chassis project found in Instructables:

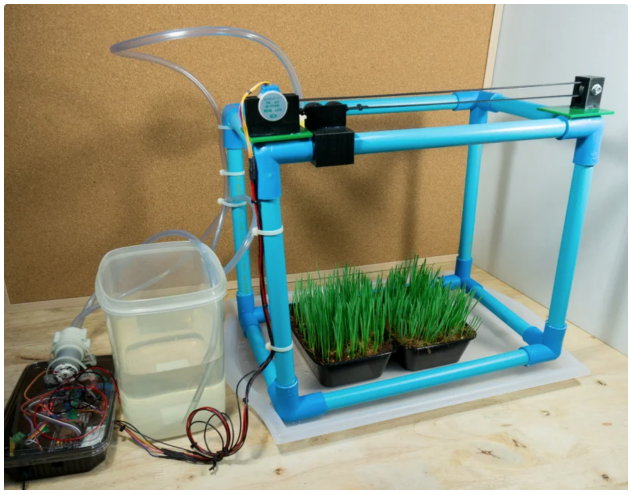


Figure 4: DIY Examples

3.1.3 Project Prototype Illustration

Figure 5 below shows all the components working together in the system. After inputting the user settings, the MCU prompts the water pump, submerged in a container of water, to water the plant. Sensors check the temperature and soil moisture in order to prompt continued watering. An Android phone receives updates on the plant's health from the hardware.

The diagrams below are preliminary depictions of the automatic irrigation system. The first image is a depiction of the connection between the electronic apparatus, the plant, and the water supply. A phone is meant to communicate with the electronic apparatus.

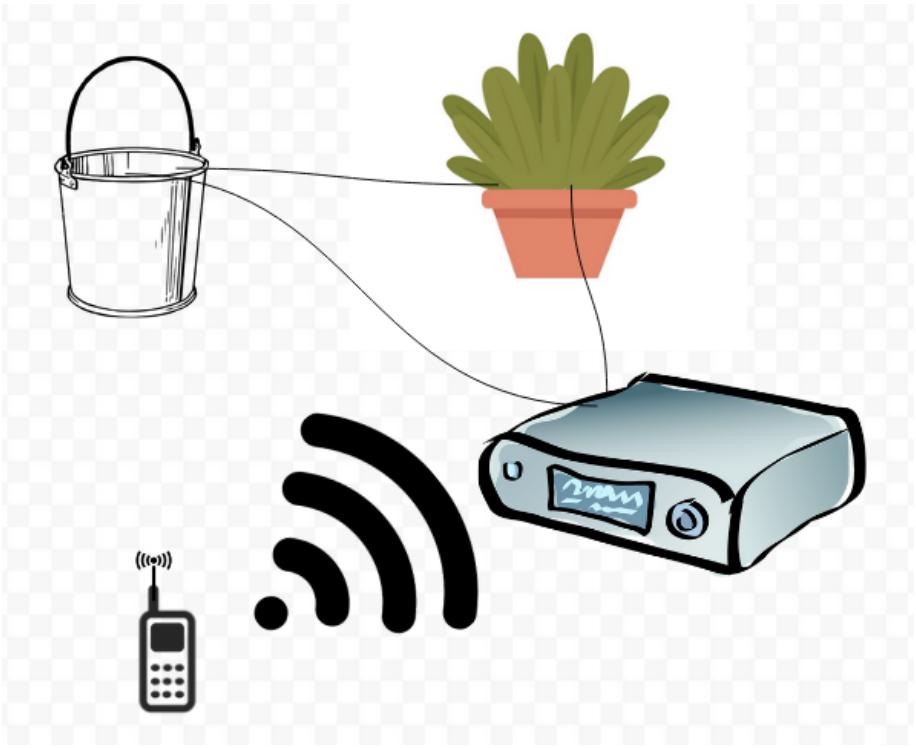


Figure 5: Concept Illustration 1

Figure 6 below illustrates how the system would work with multiple plants. It would be a centralized system with one water reservoir. The advantages of this would be the easy maintenance of filling one water reservoir and powering only one system. The disadvantages would be that if the central system were to fail, all of the plants will not be able to be watered. This would be solved with a decentralized system in which if a part fails, a smaller percentage of plants would go unwatered. The decentralized system would go against one of the project goals, which is easy setup and maintenance. The user would be responsible for filling up each reservoir which takes more time to do each cycle and they would have to make sure that all systems are being powered with five different power

systems, which increases the batteries necessary to make all of the systems function.

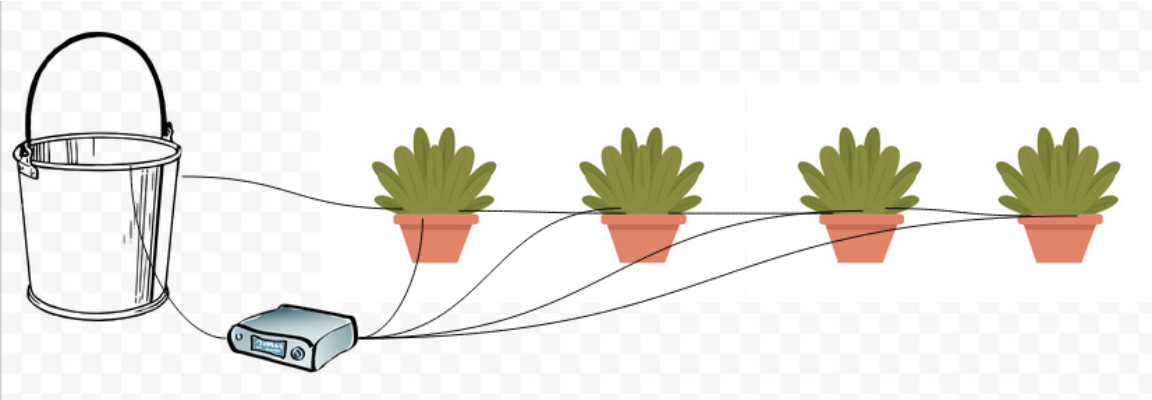


Figure 6: Concept Illustration 2

This centralized system will run off of a battery power supply as seen in Figure 7. This means that a 12V battery power supply will be supplying current to the entire system. Since the MCU and sensors will not require 12V, a step down module, or buck converter, will need to be implemented to supply the correct voltage. The diagram below is a description of the different components comprising the power structure of the system:

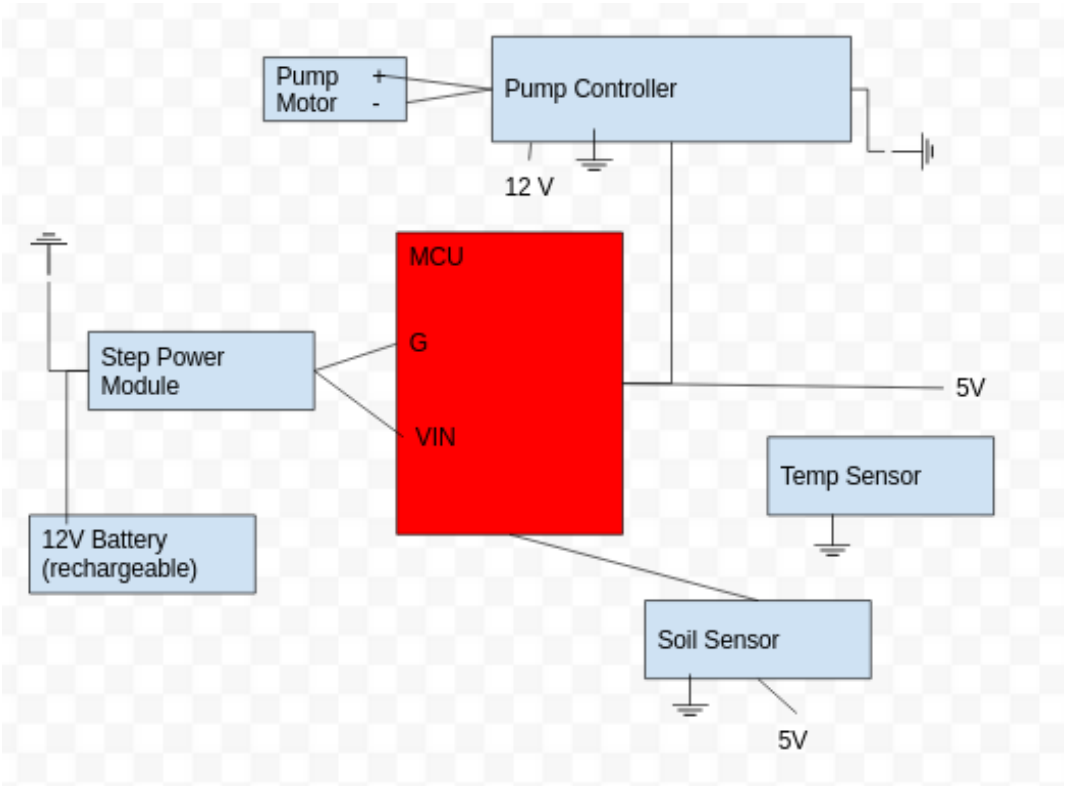


Figure 7: Power Structure of the System

3.1.4 System Functionality Software Rapid Prototype

Objective: The purpose of this prototype was to act as a rapid prototype to gain experience and insight on the functionality code of this automatic irrigation system. The Python programming language was used for this prototype due to the team's familiarity with it, even though the language for the final design of this system will more than likely be in Java or a similar language. Therefore, though Python was used, this prototype was designed using Top Down Development and Object Oriented Programming in structures that would typically be utilized for Java. This was possible because of the relative simplicity and flexibility of the Python language providing a "simulation" of what the Java code would be doing. The overall purpose of this prototype was to design and test the data structures and algorithms that will eventually be used for the functionality of this automated irrigation system.

Environment: This prototype only required software and IDEs already present on a team member's laptop so no extra facilities were required for the construction and testing of this prototype.

Procedure: To test this initial prototype, the following steps were implemented:

1. The chart shown below was constructed to break down the problem. As part of Top Down Development and Object Oriented Programming, the problem was broken into different modules. These modules were broken down into its functional parts. The functional parts were broken down into their base functions and algorithms so that there is little overlap between the different modules. Once a basic understanding of the required algorithms is found, the 23 Design Patterns tool was utilized to cross reference and potentially streamline any common algorithm designs used in this design. This chart also required a "data mindset" approach in that the engineer visualized the movement of data; how the data was received, transformed, and then output by the code.

Figure 8 which is shown below is a breakdown of the problem logic for Top Down Development.

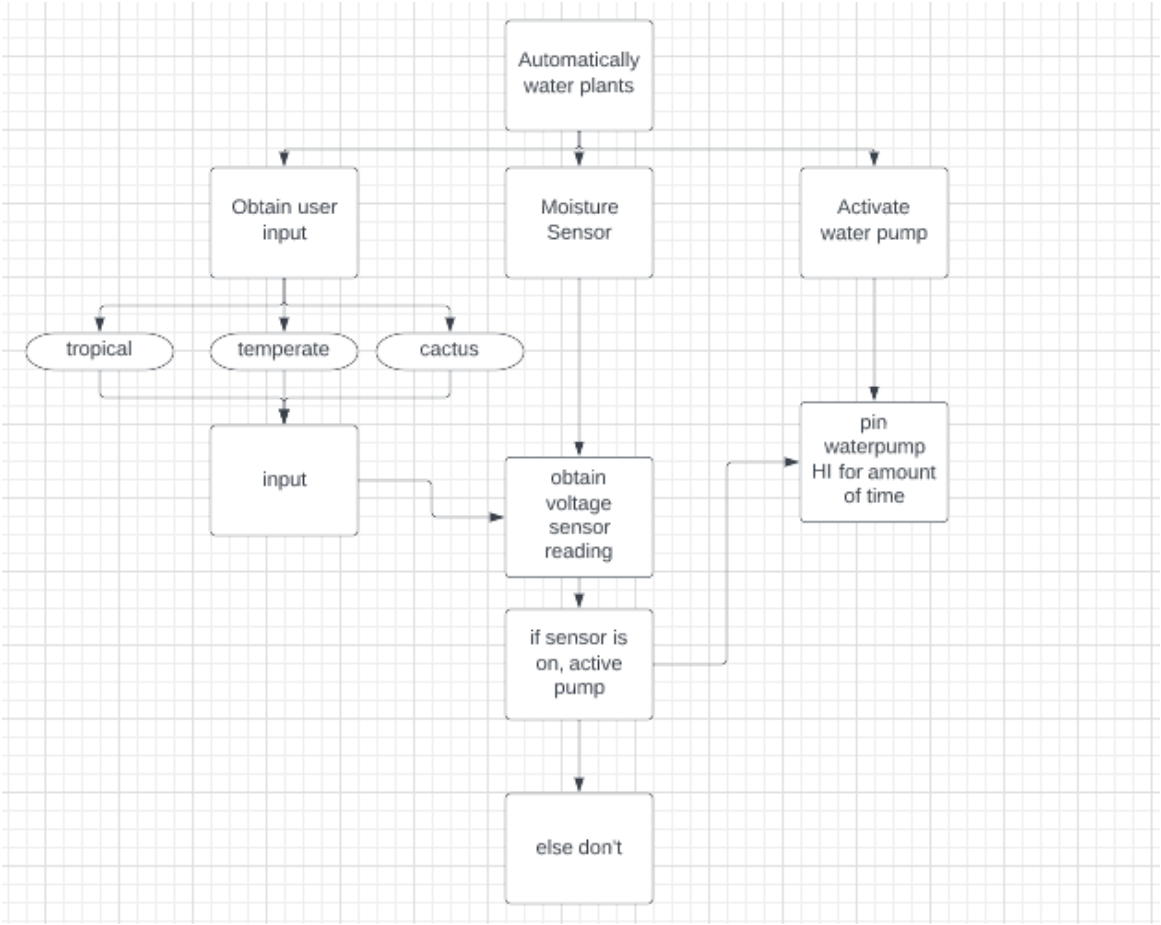


Figure 8: System Functionality Software Rapid Prototype

1. After constructing the chart, a UML diagram of the problem was created for a clearer understanding of the functions and inheritance of the different types and classes present in the code.

The next figure, Figure 9, is a UML (Unified Modeling Language) diagram showing a breakdown of the different classes and types present in the code. UML is a general purpose development based modeling language used by software engineers to standardize the visualization of a system. In using this approach to depict our project, we are attempting to appeal to a simplistic approach so that anyone who looks at Figure 9 can interpret our project's software interface. The different classes and types are very easy to depict when using this method, and it breaks down the tasks we have to implement in a memorable way for us to then incorporate into the final project once we present it.

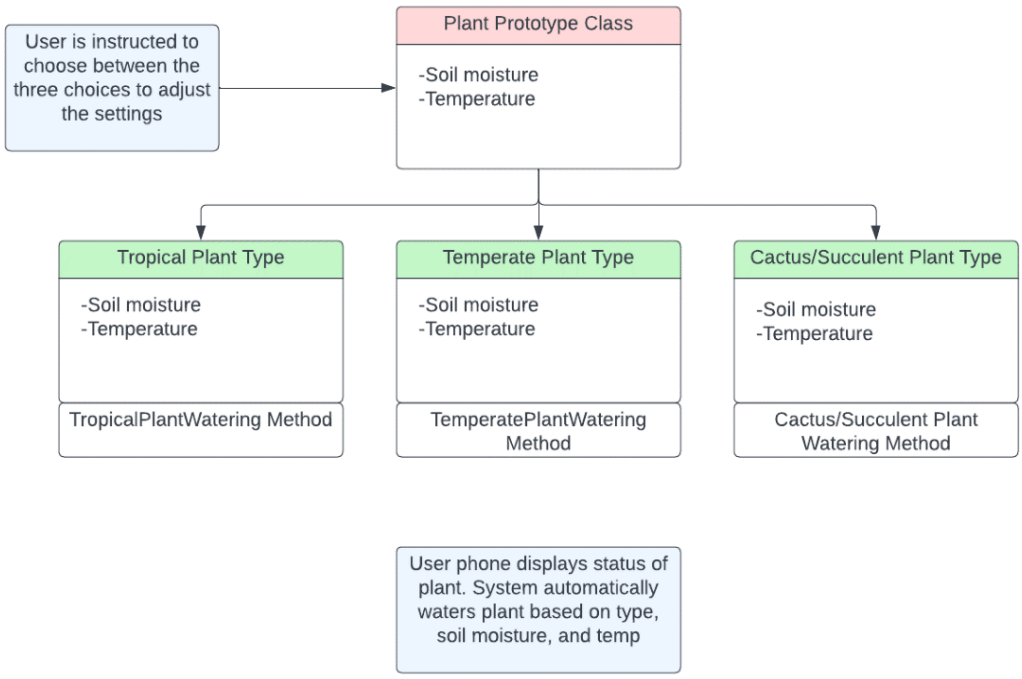


Figure 9: UML diagram

1) When constructing the code, each functional part of the code was tested for basic functionality using basic “user text or numerical input” for the purpose of finding any broken functions or algorithms present in the code. Though different modules could be developed at the same time, the functional parts of those modules are made sequentially so that there is no confusion as to where certain bugs or errors originated.

Conclusion: This prototype provided valuable insight in the functionality of this automatic irrigation system. Rapid prototypes are very valuable because designs like this give the team insight on future bugs and more optimal designs ahead of time. Since the Python code was used in a way that simulated the function of Java data structures and code, this prototype allowed the team to be ahead of schedule since this prototype helped iron out certain kinks and optimize designs that would often have to be dealt with later in the development process. Later on in the project development process, the team only had to worry about minor Java conversion issues when constructing this code as well as the actual integration of this code into the overall system, which ended up becoming highly dependent on the MCU and power PCBs final design.

3.1.5 “Snip N Drip” Initial Prototypes

Objective: The purpose of this prototype was to gain experience and insight on similar automatic irrigation system technologies already present in the industry. This experience is valuable because it makes the team more aware of the

established capabilities that these technologies have so that we may improve and build on them.

Table of Parts

Parts	Important Statistics
(4) Crowtail Soil Moisture Sensors	Analog
(1)Crowtail Smart Pump Shield	Standard
(1)Crowtail Water Pump	12V
(1)12 V Adapter	Standard
(1)Four channels water valve	Standard
(1)Vinyl water pipe	Standard
(1)Arduino Uno	5V

Table 3: Table of Parts

Environment: This prototype was constructed in the home of one of the members of the team. No electronic components and tools such as multimeters were required for this prototype due to it being a kit model. This prototype was constructed with household tools that were readily available.

Procedure:

1. Uploading the required code to the Arduino Board. The attached board is an Arduino Uno which allows you to interact with the code using Arduino IDE.

2. After this, the custom hardware “shield” is attached to the Arduino IDE. This custom hardware shield was created by the kit company as the interface between the Arduino and the sensors, pumps, valves, and switch.

3. The appropriate moisture sensors, water pumps, water valves, and switches are attached to the hardware shield. The four water pumps are connected to the A0, A1, A2, and A3 pins on the hardware shield.

4. This system operates on a pump to valve system. A water pump pumps water from the main water source and passes it to the 4 valves who distribute the water to 4 different plants. This system design is very different from the team's initial design so two different trials were implemented to test the effectiveness of each design using the prototype parts: A design that uses the water pump and four valves to water four plants or a design that uses a water pump and four connected pipes to water four plants.

5. The figures below depict the initial prototype constructed. Only one plant is pictured but this model is capable of watering four different kinds of plants. Figure 10 below shows the configuration of the vinyl tubing and moisture sensors.

The diagram below shows all of the interconnected components in the initial prototype:

Figure 10: "Sip N Drip" Photo 1

The diagram below is a close up of the moisture sensor and vinyl tubing configured in the soil:

Figure 11: “Sip N Drip” Photo 2

Conclusion: Several lessons were learned from implementing this prototype design. Firstly, the valve design seems to be more effective than the pump only design. This is because the valve design provides an extra implementation to reliably provide water to the different plants. However, the electronics and accompanying programming needed for this implementation to be functional may be far ahead of the capabilities of this team. Therefore, a pump only design that can divert water to different plants will probably work well for the purposes of the team. The team also learned about the strategy needed to deal with power supplies and hardware. This particular design deals with the issue of separate components having different voltage requirements. The water pump requires a 12V input. However, the Arduino can only stand a 5V input. The strategy taken by this automatic irrigation system is by designing a “sensor shield”. The Arduino Uno contains the chip and Internet accessibility needed for the functionality of the system but the “sensor shield” is able to interface directly with the sensors. The shield then splits the power between the Arduino and the different attached devices so the Arduino only gets 5V and the pump/switches only gets 12V. The power supply connects directly to the shield and not to the Arduino. This is because the shield splits the power between the Arduino and the attached apparatus. This strategy is analogous to the design of our power PCB and MCU PCB. However, the team will have to consider the differing power needs required to run the MCU PCB versus what the sensors and water pump needs.

3.1.6 System Chassis Initial Prototype

Objective: The purpose of this prototype was similar to the “Snip N Drip” initial prototype in that it was to gain experience and insight on similar automatic irrigation system technologies already present in the industry. This experience

was valuable to the team because it tests the viability and usefulness of the chassis design seen in DIY models of the automatic irrigation system. The DIY models are usually equipped with a chassis design to support the vinyl pipes. This takes away from the modular nature of the automatic irrigation system but provides a support structure and aesthetic value to the user. However, due to the high value to the user provided by the more modular designs, this prototype was constructed to test whether there are any other benefits to the chassis apparatus outside of it being a structure and aesthetically valuable. The working hypothesis is that such a structure might actually impede the function of the vinyl tubing by possibly making it more difficult for the pressure based water pump to move water through the tubing. However, since this takes no additional cost to the team since it is constructed from the initial “Snip N Drip” prototype and scrap wood readily found at a team member’s house, a quickly constructed, scaled down prototype was created to test the viability of such a design.

Environment: This prototype was constructed in the home of one of the members of the team. No electronic components and tools such as multimeters were required for this prototype due to it being a kit model. This prototype was constructed with household tools that were readily available.

Procedure:

1. The setup and implementation for this prototype is essentially the same except that in this version, the team is testing the effectiveness of the increased height of the chassis on the strength of the water pump. In a similar manner to the previous prototype, this will be tested with both a water pump/valve system versus a water pump with no valves system

2. The prototype was initially conceptualized using a drawing as shown below. This is a continuation of what was initially designed in the first conceptual drawing.

3. The other figures below depict the prototype constructed. The functional parts of this model are the same as the previous prototype. They were simply repurposed. For the sake of time, cardboard was used as moisture protection from water that may spill out of the enclosure. Also, the Arduino and other electronic components utilized are inside of a Ramen container to simulate the use of protective casing. Figure 12 below also shows a close up photo of the model, with the vinyl pipe connected to one of the overhead rafters using zip ties.

Figure 12 below is a depiction of the initial design created for the system chassis. This will create an overhang over the plants in which the water will be supplied. This will give structure to the plant enclosure and provide a secure way to fasten the tubing to the plants. This design will take up a lot of space, and will exclude certain plants from being used due to their potential growth size.

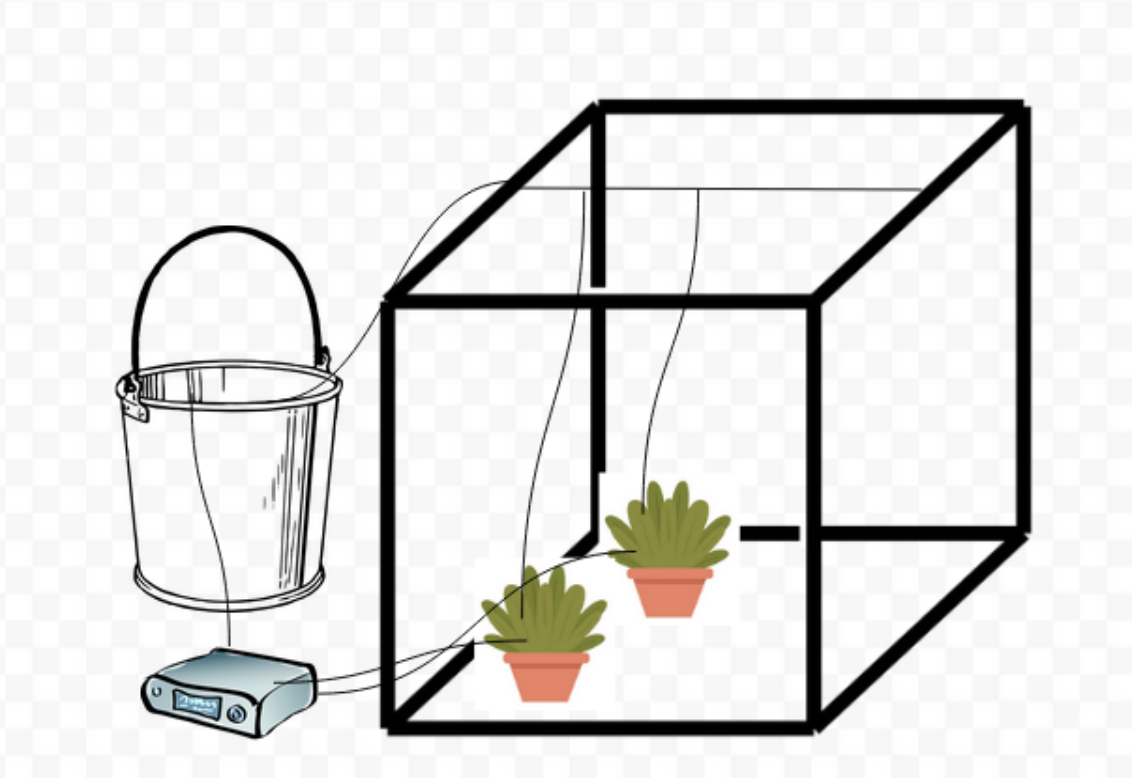


Figure 12: System Chassis Proposal

Figure 13 below shows all of the interconnected components in the system chassis prototype.

Figure 13: System Chassis Attempt 1

The diagram below is a close up of the vinyl tubing connected to the roof of the chassis. The tubing can be run below a support beam to stay hidden from the top view.

Figure 14: System Chassis Attempt 2

Conclusion: Several lessons were learned from implementing this prototype design. Firstly, the valve pump design definitely proved better than the pipe only design. However, like the previous prototype, the valve design requires more electronic expertise so this will most likely not be implemented. This leads to the second lesson learned from this prototype, that of the chassis design being too inconvenient to implement. Regardless of the pump design, the mechanism had trouble pumping water to the plant due to the elevation required to pump water through the rafters. However, there is some anecdotal evidence of the chassis design being more convenient and aesthetically pleasing to the user. It could potentially be used as a modifiable “add in” for this system when the user wishes to use the automatic irrigation system strictly for indoor plants that can fit within the chassis. To make this design work properly, it will require a design that integrates the vinyl piping into the chassis with a method that does not prevent the water pump from pumping water.

The table below is a visual summary of the constructed prototypes and their results, all which have varying degrees of success. Special circumstances or conclusions were also included in this table to help synthesis a final conclusion on the effectiveness of each of the different prototypes that were tested down below:

Initial Prototype	Parts	Function	Results	Status
“Snip N Drip” Initial Prototype	Water pump, power source and accompanying electronics, container for the source of water, vinyl tubing	Water pump provides water to plants through vinyl tubing	Adequate programmed irrigation of four plants	Primary model that will be used for this project
System Chassis Initial Prototype	Water pump, chassis, power source and accompanying electronics, container for the source of water, vinyl tubing	Water pump provides water to plants through vinyl tubing. Vinyl tubing is hung from the top of the chassis	Adequate programmed irrigation of four plants but some difficulty due to water pump strength and the elevation of the tubing	Chassis may be impractical to create but will be considered a stretch goal. Elevation based water pump issues could be fixed easily
Software Functionality Rapid Prototype	Python language due to familiarity	A python rapid prototype of the functionality software	No major bugs	The data structures in this prototype will be recreated using the language of choice

Table 4: Summary of prototypes

3.2 Sensor Research

This system operated with two sensors. There was a temperature sensor, and a moisture sensor. A stretch goal was to have a pressure sensor and light sensor. The temperature and moisture sensor provided information to the MCU about the conditions the plants are in, and the pressure sensor measured how much water is left in the water reservoir. The diagram below is an overview of the different sensor components and how they communicate.

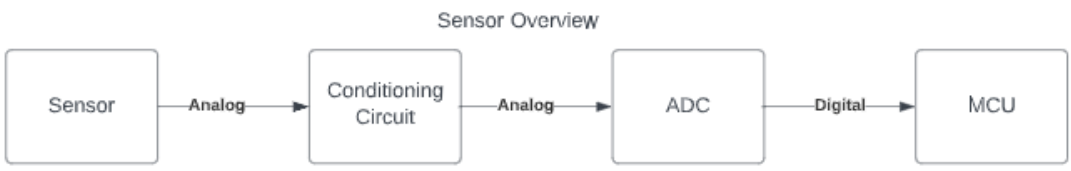


Figure 15: Sensor Overview

The selection of the type of sensor will depend on how many boxes in Figure 15 that sensor includes. Extremely basic sensors will only have the first box, while advanced expensive sensors will have the first three boxes.

An example of a very basic sensor that only has the sensor box would be a thermistor which changes resistance based on temperature. This change in temperature can be observed when a voltage is applied; however, the voltage change is very small and needs to be amplified to properly be fed into the ADC. This amplified signal once through the ADC can now be processed by the MCU. The benefits of this example would be cheap materials cost. This would be traded off with increased development time, as an amplifying circuit would have to be designed and tested for this to be feasible. The desired accuracy of the sensor will most likely not be met if the most basic sensors are used

Most sensors will have a conditioning circuit built in. Using one of these would ensure a stable amplifying circuit, which is one less issue to worry about. This will also simplify the powering process, as most sensors just need 3.3V provided for the conditioning circuit to work. All of the sensors will feed into one ADC if this level of sensors are used. The input to the ADC will be switched with a multiplexor. The ADC resolution would have to be carefully chosen to ensure that the minimum amount of information is lost.

The most advanced sensors that will be considered for this project will include the first three boxes of Figure 15. These are called digital sensors and they output digital signals. This will move most of the design complexity to the software and minimizes potential hardware failures. A communication protocol such as I2C can be established for efficient communications. The downside to digital sensors is that they are much more expensive than their analog counterparts.

These last two types of sensors were considered. It is important to note that they do not have to be mutually exclusive. If parts are unavailable, it is important to know the difference in capabilities of sensors and the design requirements that are necessary to implement them at different levels.

3.3 Power Subsystem Research

All of the systems will require power to operate. How to power these components correctly and determine the capacity of this system will be discussed in this section.

3.3.1 Pump Research

In order to ensure we had enough power in the pump unit, the water from the reservoir was required to flow the entire length of the tubing, we had to determine what power output the pump had to be able to provide. To determine this we looked over other similar products, as mentioned above, and we had to think of the distance constraints outlined on our engineering specifications. Taking into

account how many plants we would be watering, how far away each plant is from the pump, and how much tubing we worked with were all taken into consideration. In looking over different pump sizes, we determined that a micro pump would be ideal for this project. Some pumps were too large to be used with our project, so we decided to search for micro-pumps. In researching the different micro-pump options we had various items to choose, from 3V, 5V, and 12V, as well as submersible and non-submersible. We talk about testing the different options for pumps below.

3.3.2 Motor Drive Controller Board

To ensure the chosen pump was able to function and interact well with the MCU, as well as to allow us to control the pump's output, we determined we required a pump controller, also known as a motor drive controller. To determine what could be used to control the pump, we looked at several of the other designs mentioned above, and went through our engineering specifications to determine our requirements. We knew we needed something to control the speed of the pump's motor, so this was the logical solution. In searching for a motor drive controller, we did not find many options, but we found two that were both a DC Dual H-Bridge, with a datasheet and the necessary pins we required to control the pump.

3.3.3 Tubing

From our chosen pump, we had to determine what the inlet and outlet size tubes we would require. From the pump's data sheet we found the inlet's (suction) diameter was 9mm, and the outlet's diameter was 6.8mm. We determined we wanted to be able to cut the tubing down to size, in order to ensure we had left over we wanted to buy both sets of tubing with a 10 foot length. We chose to go with silicone tubing, as it was readily available in the sizing we needed, and we did not require a food grade silicone, or aquarium grade tubing. In the pump and piping combination pack, we received two separate sets of tubing, which were able to fit into the 3-5 volt pumps, as well as the 12 volt pump, thus allowing us to proceed without testing other products.

3.3.4 Battery

In the initial design of our project we went back and forth trying to determine if the power was going to be from a battery, a solar cell, or from a wall outlet. To ensure the product is able to withstand any challenges it faces, we decided a battery would be the best course of action, with a wall plug adapter via DC jack as a secondary power source. If the power were to go out while the customer was on vacation we would want to ensure they would still come home to a plant that is alive and well. To determine the ideal battery/battery pack we had to look at our power requirements. The pump requires a 12V input, and the other components would require power as well. In searching between 12V batteries, we determined we would need one with a long battery life as well as being rechargeable. The

chosen wall adapter was decided when we wanted to implement a DC jack into our PCB. This wall adapter had an internal inverter to go from 120V AC power to 12V DC power. This led us into looking at the different types of batteries available on the market. Our options were NiMH (Nickel-metal hydride), LiFePO4 (Lithium Iron Phosphate), or SLA (sealed lead acid). Of these three options, some of them were too large to be realistic, namely the SLA was too bulky. Of the three different types the NiMH seemed the most simplistic, as we did not want a battery that was too large.

The differences between Ni-MH batteries and Lithium ion, or lead batteries, are based on the fact that nickel based batteries do not have a “float charge” voltage. This means that charging is based on forcing the current through the battery, and the voltage required to do this is not set in stone like for other batteries. This makes charging Nickel based batteries harder to charge accurately. This leads to different practices of charging the Ni-MH battery. There are at least three different ways to approach the charging of Ni-MH batteries. The first is overnight charging, where you charge the battery at Capacity/10 or below)10% of the battery's rated capacity per hour). This fact is demonstrated with our chosen battery, as the information provided with it recommends standard charging to be 190mA for 16 hours, which is 10% of the batteries rated 2000mAh capacity. Quick charging, or using Capacity/3.33 (33% of the batteries rated capacity) is another method, mirrored by our chosen batteries’ specifications sheet at 380mA for 7 hours. And lastly Fast and or rapid charging. This method uses the batteries capacity at Capacity/0.5-1 (50%-100% of the batteries rated capacity). This rapid charging method is more problematic to the batteries’ health and capacity, but it does allow for much fast charging time allowing anywhere from 1 hour to 2.5 hours for a full charge.

3.3.5 Buck Converter

In order to power some of our lower powered devices we thought we would need a buck converter, or a power supply step down module. A buck converter is used to step down voltage from an input source to an output source. It is based on a DC-DC power connection. It reduces the average current from the input to the output side of the converter. From our requirements we know our battery source will be a 12V battery, and will need to be stepped down to either 5V or 3.3V to power some of our sensors, or our MCU. In implementing our PCBs we determined we did not require a Buck Converter but would instead use two separate linear voltage regulators, one supplying 5 Volts and another supplying 3.3 Volts.

3.3.6 Solar Cell

Another consideration of this project is to be somewhat self sustained, with regards to the power subsystem. Given the focus of this project is to create an autonomous self sustaining plant watering system. The photovoltaic (solar cell) would be used to charge the chosen battery, but the issue surrounding this idea

would be the unregulated current going into the battery. Given the battery options mentioned above, the choice of battery would affect the efficacy of using a solar cell. To fix this issue we could try to find a solar cell with a PWM (Pulse Width Modulator) sensor which would regulate the input voltage and amperage based upon the current solar power level. Given the complexity and added cost adding a solar cell would contribute to the project, and the current budget constraints, we decided against using a solar cell for this project.

3.3.7 Electronic storage

Given our project is working with water and electronics, a big consideration we must account for is to ensure our electronics do not get wet from the pump or external conditions. With the nature of plants, we are considering multiple options to store our electronics in. Tupperware was initially a simple choice, and would allow us to route wires through the sides of the container, while maintaining a water resistant seal. Another option is a glass based tupperware container, in which we would use the plastic lid to route the wires through, rather than the sides of the plastic tupperware container. Lastly another option would be using a battery box, which is something that is used to store larger 12 volt batteries, often meant for cars. This option would give us too much space, and would more than likely be too large and bulky for our project. At the end of our implementation stage we decided to laser cut a wooden box with holes for our wiring to exit the box. Both the PCB and 12V battery were able to fit in the box, and there was more than adequate heat sink with the wiring holes on the box.

3.4 Relevant Technologies Research

In this section we talked about relevant and popular technologies for Web and Mobile Applications. There are so many different frameworks, especially javascript frameworks for web development coming out every month. With so many frameworks coming out there are so many options to choose from and are useful for different applications.

3.4.1 Mobile Application Technology

There are a lot of technologies out there to help create mobile applications for various operating systems such as IOS and android. There is a lot of debate for which technology is the best to use. Some technologies allow you to compile one source code for both android and IOS, while other technologies focus on solely IOS. Some of the most notable and popular technologies are React Native, Flutter, and Swift.

Each technology has advantages and disadvantages for certain use cases. We will analyze all the pros and cons of each technology to see what suits our mobile app the best. We already know we will need to use a technology that allows for both android and IOS devices due to the team having different phones. Using the

same code base for both android and IOS is also another benefit and more convenient than having two different code bases for both android and IOS.

3.4.1.1 React Native

React Native is an open source framework created by Meta that is used to develop android and IOS applications. React Native utilizes the React framework that is used to build web applications. A neat thing about React Native is that it allows you to build mobile applications for both android and IOS devices using the same code base. Due to the flexibility, react native is a super popular framework that was used to develop very successful applications such as Facebook, Instagram, and Skype.

Some notable features of React Native are compatibility, reusable code, and hot reloading of code. Compatibility between operating systems is a breeze because you can use the same code base to release an application for both Android and IOS devices. Having to keep two code bases for each operating system would be a hassle and much more time consuming to change the same code in two different places. React Native also lends itself to a lot of reusable code with making use of components. React components allow the same code to be used in as many places as you want in your code base. These components can be unique in each instance they are used by utilizing props. Props allow you to pass data to the component, so there is not just static data on each component. Hot reload of code is beneficial during development because you can change values/variables on the fly and any changes to the code is shown immediately. Meaning you do not have to re-compile your code to see the changes.

Let's say we have a React component which is called Comment. We want to have comments on a blog, but don't want to reuse code when only the author, text and dates are different. Here is where Props are useful. We can pass unique authors as a prop to each instance of the component. Then the component can render for each unique author, while reusing the code you made for the comment React component. React components are not only good for reusing code but also beneficial for testing. It is easier to test one react component instead of all the individual comments you may have in your code.

3.4.1.2 Flutter

Flutter is an open source software development tool created by Google. Just like React Native flutter can be used to develop applications on both Android and IOS, using the same code base. Flutter is developed in either C, C++, or Dart languages. Flutter also provides some nice features such as GPS coordinates, permission handling and credentials that are easy to use.

Some features of Flutter include hot reload , compatibility, and its own rendering engine. Hot reload is the same as React native. You do not have to re-compile your code for changes to appear, changes are done on the fly and the screen is updated automatically. Flutter also has the same compatibility as React native. It

supports both android and IOS mobile applications to be developed using the same code base. Flutter also has its own rendering engine which allows both android and IOS interfaces to look unique in their own way. Flutter has packages that contain custom widgets and styles for each operating system. This allows for your mobile application to look the best it can with whichever operating system it is running on.

Flutter does have its disadvantages though, such as the lack of third-party libraries and Dart as the language. The lack of third-party libraries is an unnecessary roadblock in development. Libraries are great because they are pre-tested by the community and open source. Libraries speed up the development process and allow for us to focus more on our specific project. Rather than wasting time coding up a common widely used feature in mobile applications. Dart is a decent programming language however our team is much more familiar with JavaScript so learning an entire new language would not be time efficient when we have previous experience with javascript already.

3.4.1.3 Swift

Swift is a programming language that was developed by Apple and open source users. Swift is coded on Xcode, an IDE also developed by Apple. Swift was meant to be easy to use even for people that are not too comfortable with coding. It is user friendly with shorter syntax and easier readability of code. Swift is compatible with all Apple platforms such as IOS, macOS and watchOS.

The huge drawback of Swift is that it only allows for development on the IOS operating system and Apple devices. To create an android app would require a completely different code base and technology/language. Due to our team having both android and IOS devices, this solution does not fit our project well. However, if we were planning on creating only an IOS mobile application swift is a decent option to go with.

3.4.1.4 Xamarin

Xamarin is an open source platform for creating IOS and Android devices. Xamarin uses .NET and C# to develop applications. .NET is a developer platform that consists of libraries, tools, and an assortment of programming languages to build mobile applications. .NET features both Lambadas and Asynchronous programming to help facilitate code logic better for the developer. Lambdas allow for nameless functions and make the code easier to read at times. Asynchronous programming is very beneficial with web design due to having to access databases and hitting API endpoints. Asynchronous code doesn't hang up your entire code and can accomplish other tasks or events while waiting for the asynchronous code to complete, such as hitting a database.

3.4.1.5 Selection for our mobile application

React Native was the best option for this project for multiple reasons. First we all have a mix of android and IOS devices, so we will need the compatibility for both operating systems. React Native allows us to have one code base for both IOS and Android operating systems. Secondly, we are already familiar with React and javascript which will result in less of a learning curve while developing our application. Lastly, React Native has a bunch of external libraries which makes development a lot easier.

Technologies	Supported Operating Systems	Features	Coding Language	Cost
Flutter	Android and IOS	Hot reload, Open source, and Rich widgets.	Dart	Free
React Native	Android and IOS	Reusable components, Open source and Hot reload.	JavaScript	Free
Swift	IOS Only	Open source, Easy to understand syntax, and Multiple return values.	Swift	Free
Xamarin	Android and IOS	Open Source, Complete development ecosystem, and compatibility.	C#	Free

Table 5: Summary and Selection of Mobile Application Software

3.4.2 Possible Web Stack Options

In this section we will compare and evaluate different Web stacks. There are a lot of popular web stacks that are used for different reasons or applications. Typically multiple stacks will work with whatever application you are making, but personal preference may help you choose which stack you want to go with for your specific application.

3.4.2.1 MERN stack

The MERN(MongoDb, Express, ReactJS, NodeJS) stack is a variation of the MEAN(MongoDB, Express, Angular, Node.js) or MEVN(MongoDB, Express,

Vue.js, Node.js) stack. As shown in Figure 15 below , we can see the front end technology is chosen from either Angular, React.Js, or Vue.Js. Certain technologies are favored over others due to the requirements of the project. Or one technology is preferred due to the team having experience with that technology.

The MERN stack consists of four technologies. 1) MongoDB is a non-relational database or sometimes referred to as NoSQL. 2) Express is a node.js web application framework that provides features for building mobile and web applications. 3) React is a front-end javascript library that is used to build user interfaces for websites. For this project we will also use React Native for our mobile application. 4) Node.js is a back-end JavaScript runtime environment that allows both the client and server side to use javascript. Instead of having different languages for server and client side.

The diagram below is an overview of the potential web stack options that could be taken for this project. It shows the different web stacks being MEAN, MERN, and MEVN stacks. They all have different use cases and people also have personal preference as to what technology to use. There is a debate of which javascript framework is the best for web development, the big frameworks that are debated are Angular, React.js and Vue.js.

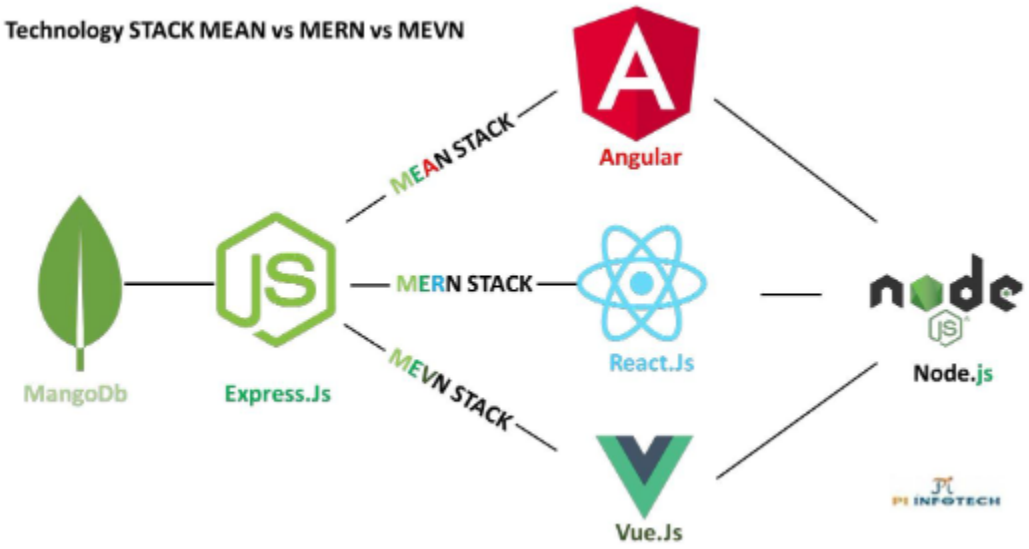


Figure 16: Stack MEAN vs MERN vs MEVN

3.4.2.2 LAMP Stack

The LAMP(Linux, Apache, MySQL, PHP) stack is another commonly used stack for web applications. Much like the MERN stack, a lot of technologies are swapped out, for instance you may use Perl or Python instead of PHP. LAMP stack is flexible and is easy to find support due to the popularity of the technologies.

The LAMP stack also consists of four technologies like the MERN or MEAN stack mentioned above. 1) Linux is the operating system which the LAMP stack is run on. Linux is a highly favored Operating SYstem due to it being open-source and compatible with LAMP technologies. 2) Apache is a HTTP server that processes requests and sends information to and from the user. Apache is also run on linux OS. 3) MySQL is a relational database which features high performance and the flexibility of being an open-source software. 4) PHP is used as the programming language, but it is also swapped out with Perl or Python.

Web Stacks	Technologies	Programming Language	Database	Operating System
MERN	M - MongoDB E - Express.js R - React.js N - Node.js	Javascript frontend and backend	MongoDB	Cross-platform for both windows and linux.
LAMP	L - Linux A - Apache M - MySQL P - PHP or Python	PHP or Python can be used.	MySQL	Linux

Table 6: Mern Stack vs Lamp Stack

3.4.3 Databases

In this section we will cover different Databases we can choose to store all of the data for our Mobile and Web Applications. We are going to go over different types of databases such as SQL and NoSQL databases.

3.4.3.1 MongoDB

MongoDB is a document-oriented NoSQL database. Some notable features MongoDB has is high performance/speed, simplicity, and horizontal scaling with sharding. Due to MongoDB being a Non-Structured query language, we do not need to create tables. Which allows a MongoDB query to be impressively fast. MongoDB is simple compared to other query syntax such as SQL. MongoDB carries less of a learning curve compared to MySQL for example. Lastly, MongoDB makes it easy to scale your database. MongoDB supports horizontal scaling which uses more nodes to share the load, instead of one node taking the entire load as shown in Figure 17. MongoDB also utilizes sharding which is spreading the data across multiple nodes created by horizontal scaling.

The diagram below is a visual representation of horizontal scaling.

Figure 17: Horizontal Scaling

3.4.3.2 Firebase Database

Firebase is a platform for developers to create mobile and web applications. Firebase has many services such as hosting, database, authentication, push notifications and more. Here we are going to talk about the database service firebase provides. They provide a service called Firebase Realtime Database which is a cloud-hosted NoSQL database. The Realtime Database allows your users to see data updating in realtime with little latency. This is a huge bonus for User Experience not having to refresh the page for new data.

3.4.3.3 MySQL

MySQL is an open-source relational database system. The database is relational in which the data relates to each other. The data is stored using Tables, which contains columns and rows. Each table then is connected or related to another table in a one-to-one or one-to-many relationship. Being an SQL database the data is structured and has a predefined schema, where NoSQL databases have dynamic schemas. SQL databases are also vertically scalable where NoSQL databases are horizontally scalable, as we saw with MongoDB, a NoSQL database.

Databases	Data Storage	Key Features	SQL or NoSQL	Data Representation
MySQL	Stored as rows in a table	Supports large databases and is easily scalable for larger applications.	SQL	Rows in a table format
Firebase database	Documents	Realtime database to change values on the fly and Ready-made API.	NoSQL	JSON-like documents
MongoDB	Documents	Horizontal scaling, Simple syntax, and Easy to read JSON like documents.	NoSQL	JSON-like documents

Table 7: Database Comparison

3.4.4 Hosting

In this section we will cover different hosting platforms and which one works best for our project. Having a good hosting platform makes development and troubleshooting a lot easier when running into issues.

3.4.4.1 Heroku

Heroku is a container-based cloud platform which is used to deploy and manage software applications. Heroku is filled with features such as supporting multiple languages/ runtime environments, smart containers, and easy scalability. Supporting numerous runtime environments / languages such as Node.js, Ruby, and PHP allows heroku to be flexible and work with many different web stacks. Heroku utilizes smart containers that provide features such as security, load balancing and error/message logs. Heroku makes scalability a breeze by having both manual scaling and auto scaling features. You may set the auto scaling range based on a minimum and maximum cost that your cloud container will be, as seen in Figure 18 below.

The diagram below shows the autoscaling capability of Heroku.

Figure 18: Heroku

3.4.4.2 Vercel

Vercel is the official hosting platform for Next.Js applications. Vercel makes it easy to deploy and manage web applications. Some advantages of using Vercel are github integration, custom domains, and automatic scaling. You can easily connect your github repository to Vercel and deploy your website. When any changes in the github are pushed to the main branch, Vercel will also redeploy your website to the newest version. Vercel supports custom domains as well as a free SSL, so encrypted data will be securely sent to and from the server. To top it all off, Vercel has automatic scaling with no server configuration needed. Making Vercel a great platform for developers to easily deploy and host web applications.

3.4.4.3 Microsoft Azure

Microsoft Azure is a cloud computing service that provides many services such as analytics, storage, networking and cloud services. Azure has amazing availability with approximately only 4.38 hours of downtime a year. Having a reliable hosting platform is essential for certain applications and use cases. Azure also features easy Scalability for an influx in usage. For example if there are a surge of new users signing up for our application our database will be used more than normal. If this were to happen Azure makes it simple to scale up our database to be able to handle the influx of customers or users.

Microsoft Azure has very flexible pricing by only paying for what services you are using for your application. They offer an estimate where you calculate the hourly or monthly costs with hosting your website using the Microsoft Azure platform. They offer paid services such as virtual machines, Azure SQL Database, Azure Cosmos DB, and Cloud storage. If certain services are unused for one month then you will not be required to pay for those services. This pricing system benefits the developer due to paying for what you need to use, rather than a bundle of services you may not be using but you are still paying for.

Hosting platforms	Is there a free tier?	Features	Featured Customers	Cost
Heroku	Yes	Automatic scaling and container deployment.	ClickMechanic, ThinkMD, and HotelEngine	Free tier 1) Production - $25 2) Advanced - $250 3) Enterprise - Contact Heroku for custom pricing
Vercel	Yes	Deploys with pushes to the main git branch and supports edge functions.	Facebook, Ebay, and The Washington Post	1) Free tier 2) Pro - $20 a month per member 3) Enterprise - Contact Vercel for custom pricing
Azure	Yes	Cloud computing, High availability, and Cost Effective.	Verizon, MSI Computer, and LG Electronics	1) Free tier 2) Customized price for different services utilized by the customer

Table 8: Heroku vs Vercel Hosting Platforms

In the table above we compare three different hosting platforms, Heroku, Vercel, and Microsoft Azure. All of these platforms are great choices for hosting and some work better than others for different applications. Our team has decided to go with Vercel as it integrates nicely with github and Next.js applications.

3.5 MCU Research

An important feature that was required when picking a MCU in the scope of this project. The MCU needs to be relatively low power and be able to support several communication protocols such as I2C and SPI. To be able to support these protocols, an internal clock was necessary for synchronous communications. A second low frequency clock is also important for scheduling interrupts. Depending on the level of sensors used, an ADC will be required for purely analog inputs. To remain flexible, the MCU should have multiple extra GPIO pins and analog inputs to facilitate prototype changes.

3.5.1 MSP 430

The MSP 430 architecture is a line of Texas Instruments embedded processors. They are low power and interface with embedded protocols. They usually have an ADC, timers, and GPIO pins. The CPU is 16-bit and has access to some flash memory. The computer architecture is RISC based, meaning the assembly code has simple and short commands, which is ideal for embedded systems. These types of processors will have a difficult time processing graphics or managing lots of data, but this sort of computing will be unnecessary on the PCB.

3.5.2 PICmicro

The PICmicro architecture is Microchip's line of embedded processors. This architecture is also low powered and uses RISC architecture, but it has a few differences from the MSP430 line. The PICmicro architecture uses Harvard block architecture, meaning that the program memory and the data memory are separate. This allows for each word to have a better ratio of data to interfacing. It also supports pipelining, but it is only two stages, which would be important to use to test speeds. Like the MSP430 architecture, PICmicro also supports ADCs, timers and other modules.

3.5.4 Arduino

The arduino architecture is an embedded testing environment that interfaces with high level coding for quick testing and simple code implementation. Arduino supports timers, ADCs, and embedded communications protocols. Arduino has a vast programing library that supports popular sensors and modules. This allows for rapid prototyping. Arduino boards usually can be directly interfaced with PCs for quick program uploads. Use of an arduino would be excellent for testing. This system usually uses ATmega chips, which will use the arduino IDE and does not need a development board.

Part	Pins	Clocks	ADC	Cost
MSP430FR247x (MSP430FR2475TRHBT)	40	6	12-bit	6$
MSP430F552x	47	4	12-bit	7$
ATMEGA328P-PU	28	External	10-bit	2.87$
PIC24FV16KM204	44	2	80 bit, 12-bit	4$

Table 9: MCU Comparison

The Arduino architecture is the most used architecture for open source users or people who like to do side projects. This means there were a lot of supported libraries that were able to be used in development. The Arduino environment provides a great testing ground for hardware since the coding is easier and it usually comes in a complete hardware and software package. This is ideal for testing. For these reasons, Arduino was chosen as a testing tool and as our final MCU.

4.0 Standards and Constraints

In this section, there is discussion on the different standards and constraints that were researched and eventually imposed on this project. These important design constraints are present in order to make a more functional and usable project.

Standards are necessary in the field of engineering. Given the numerous different aspects of our project, we have to touch upon standards from several different governing bodies. Our main source of standards will be from IEEE SA (Institute of Electrical and Electronics Engineers Standards Association). IEEE has over 2000 different standards, accepted across more than 150 countries. Several other standard advisory groups include: IEC (International Electrotechnical Commission), ANSI (American National Standards Institute), and NEC (National Electric Code). Each of these governing bodies has a focus. IEC focuses on international collaboration in order to provide instruction, guidelines, and rules in the process of designing, manufacturing, installing, and certifying electrical and electronic devices and systems. ANSI focuses on the U.S. standardization system to identify standards in support of emerging technologies and to ensure the U.S. can meet global regulations. Lastly the NEC is a book of standards made for the U.S. with regards to electrical wiring and electrical equipment, in order to ensure safe working conditions for electricians and occupants.

Oftentimes companies and or governing bodies will adopt more strict standards than necessary. For example, NASA has a set of standards that supersedes the NEC code, as it is increasing safety. If doing electrical work at NASA's Kennedy Space Center, the minimum conduit allowed on center is designated to be ¾" conduit, whereas the NEC allows for ½" conduit.

4.1 Wireless and Network Protocols

This system will need to be able to communicate with the phone interface wirelessly. There are two wireless protocols being considered, Bluetooth and Wi-Fi. To understand Wi-Fi capabilities, network standards such as TCP and IPv4 would need to be investigated and explained.

4.1.1 Bluetooth

Bluetooth is a protocol developed for wireless communication. Usually, this form of communication is for short distances only and is done between two digital systems. It is also important to note how the communications will be done between the system, phone interface, and the internet due to the inherent constraints of the Bluetooth protocol.The diagram below shows the different components of the system and how they are connected. The arrows between them show how they communicate with each part and in what direction this is possible in.

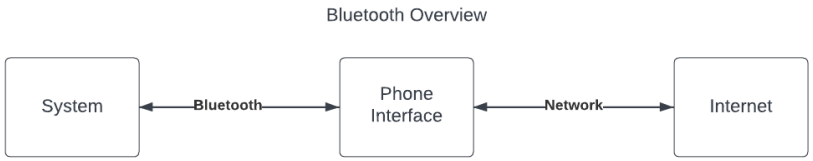


Figure 19: Bluetooth Overview

The bluetooth protocol is considered Peer-to-Peer. It connects the system directly to the phone interface. The range can be about 50-100m. This will change the scope by only allowing the system to communicate with the phone interface when the user is at home as seen in Figure 19. Little information (a few bytes) will need to be transferred to and from the system, so the limited data speed of Bluetooth (1Mbps) is not a limiting factor. Bluetooth uses 40-50 times less power than the standard Wi-Fi protocol. This will reduce the power load of the PCB system. Since only the phone interface will have network connection to the internet, as seen in Figure 19, that means that TCP requests will fully be handled by the phone interface and the system will not have an IP address. This moves complexity from the system to the phone interface. Another consequence of Bluetooth is that the system will only update when in range of the phone

interface, which means that when the user is outside the range for any extended period, monitoring will have to be predictive since live data is not being used.

4.1.2 Wi-Fi 802.11

The diagram below shows the different components interacting with wifi and how they are connected:

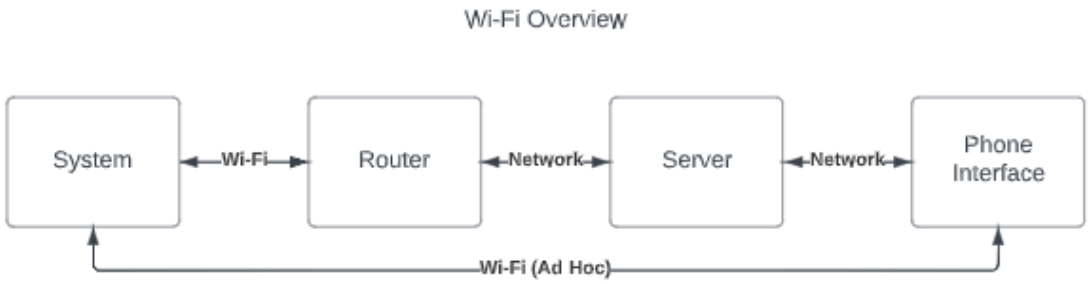


Figure 20: Wi-Fi Overview

This protocol can be either client-server based or peer-to-peer. It will allow the device to directly communicate with a server which can provide information from the internet. This client-server connection can be seen in Figure 20 from the connection of the system to the router, which will allow it to use a network to access the server, which can access the phone interface wherever the network can reach. This greatly extends the operational distance. 802.11 also supports an Ad Hoc mode, which can be seen in Figure 20 between the system and phone interface. This will have a definite range but will allow the phone interface to bypass the server and network to communicate directly with the phone interface. This means that the system can work without a network which will extend the operating lifetime of the system past the website's lifespan. 802.11 has different versions which are denoted by a letter suffix. Each different version differs in range and data transfer speed:

Standard	Bit Rate	Frequency	Date of Release
802.11b	1-1Mbps	2.4GHz	1999
802.11g	6-54Mbps	2.4GHz	2003
802.11n	72-600Mbps	2.4GHz or 5GHz	2008
802.11ac	433-6933Mbps	5GHz	2014

Table 10: Wi-Fi Protocols

It is also important to note that the amount of power required for each protocol increases the newer and faster the protocol is. Since this system is not transferring lots of data, the faster and newer protocols are not critical to implement.

4.1.3 TCP

TCP stands for transmission control point and is a protocol for the transport layer of the internet. This protocol handles initiating and maintaining communications between two hosts. First, one of the hosts sends a request to the second host. The second host sends an acknowledgement back, and the first host confirms the acknowledgement with its own. This completes the three way handshake, and requests for information can start. The information to complete this is held in the TCP segment of the packet header. It contains information such as the source and destination port number and the sequence number, which is important for maintaining connection. Overall, this protocol is to ensure process to process communication and is key for effective communication through a network. In the scope of this project, this protocol will be used between the wifi module and the website.

4.1.4 IPv4

IPv4 stands for internet protocol version 4. This protocol adds a header to a packet that gives information so that the packet of information can be routed through a network. For this version of IP, the router assigns a host a 32 bit address in the form of four numbers that range between 0 to 255. The first two numbers on the left (most significant bit) identify the network, and the second half identifies the host. Both the destination IP and the source IP are in the header of the packet. The other information in this header communicates information that helps with routing (such as time to live), what is in the header beneath (TCP or UDP), and the total size of the packet.

4.2 Hardware Communication Protocols

Three hardware communication protocols are being considered for the digital communications of this system. This includes UART, I2C, and SPI. Each of these protocols uses serial transfer, which means that each transfers one bit at a time. These protocols will be used between PCBs. This information is binary, which means it can be transferred through GPIO pins which only read high or low. These protocols do not have to be mutually exclusive as long as the MCU supports more than one.

4.2.1 UART

UART stands for universal asynchronous receiver/transceiver. UART allows for the clocks of the transceiver and receiver to be asynchronous, meaning they do not have to have the same clock edges. Depending on the type of MCU chosen, this protocol can be implemented without a communication module. It starts its communication with a specific start signal and ends it with a specific end signal. This transfer of information is usually done at a specific Baud rate, which would be chosen when configuring the UART. When configuring UART, several parameters need to be set. The first parameter that is set is the rate of communication, or Baud rate. A popular Baud rate for these types of systems is 9600 baud, which should not be too taxing for this system. The next parameter is parity, which reserves one of the bits of the byte transferred as a means to check if there was an error in transmission. This is to ensure that the correct signals are being sent, but due to the low noise environment of this communication, parity will not be needed for this system. The amount of stop bits will also be chosen between one or two bits. For this system, one bit would be considered. These past two parameters, the parity and the stop bits can be changed if the communications are faulty. There is a parameter that can control the flow of information, which would be used if the receiver is much slower than the transceiver and needs time to process information. The least significant bit is usually transferred first in this protocol. This way of communication can be half duplex or full duplex. Half duplex means only one wire is used, which means communications can only happen in one direction at a time. Full duplex means that there are two wires used in communication which allows bits to travel in both directions at the same time. Either way, this means that if multiple devices are using the same bus, collision will occur.

4.2.2 I2C

I2C stands for inter-integrated circuit. This protocol comprises of at least one master device connected to slave devices by two buses. One of the buses is called the serial data line (SDA) and the other one is called the serial clock line (SCL). The SCL transmits the clock signal which originates from the master device, and the SDA transfers the data. In this protocol, only the master can initiate a request and it can only make a request to one slave device at a time because the information is transferred with half duplex. To prevent information collisions, the voltage is either pulled up or pulled down on a bus. Usually, when the voltage is pulled up, that means the bus is unused. When communication starts, the line is pulled to low, which lets all the other devices know that another device is transferring data. This communication starts when the master device sends a start signal and ends when it sends a stop signal. Each slave device has its own address so that the master can accurately send requests. This protocol would work well between the MCU and the sensors. The sensors will have to be digital sensors, and the master device will read their registers for the sensor input. This would skip the need for designing an ADC. I2C has a standard, fast,

and high speed mode. This experiment does not need very fast communications, so standard mode will be used as a default.

4.2.3 SPI

SPI stands for serial peripheral interface. This protocol has a master system and a device it connects to. The communications are full duplex, so two wires are used for communications, and it is synchronous, so the master drives the clock. It is important to note that SPI can also be half duplex, with only a serial in out (SIO) bus for information. There is also another bus that is called the chip select which the master can use to select which device to transfer bits to or from. This comes out to a total of 4 buses. The two communication buses are called master out serial in (MOSI) and master in serial out (MISO). The master will select a device with the chip select and start communications with MOSI to give the device information or MISO to receive information. To have multiple SPI devices, the easiest way would be to daisy chain the devices together by having the master connect its MOSI bus to one of the devices, and have the devices connect their MISO buses to each other while having the last one in the sequence connect its MISO bus back to the master. This means to get information from the first device, it will have to go through all the other devices in the daisy chain first before making its way to the master. The disadvantage of this would be the loss of data privacy, but in the scope of this project the data being transferred does not need to be secret to the other devices in the system. The only other way to avoid daisy chaining the devices together would be to have the master have multiple chip select buses, which quickly does not scale. Ideally, this system should have no more than 2 SPI devices.

4.2.4 USB

USB stands for universal serial bus. In USB communications, there is a master device with a slave device. Information between the two devices are sent in packets. Each packet contains sections that relay certain information, and different sections are used for different interactions. First, a token packet is sent to start transmitting information. This information includes SYNC information, which is used to match the clock speeds of the devices, the PID information which describes the type of data being transferred, the ADDR field which contains the address of the receiver device, the ENDP which contains the endpoint number, the CRC field which has ensures the validity of the bits, and finally the EOP which signifies the end of the packet. Once this is complete, data packets can start to be sent, which replace the ADDR and ENDP with the data information. There is also a handshake packet and a start of frame packet. The handshake packet is for acknowledgement or sending error messages while the start of frame packet is for scheduling. Other important standards that govern USB is the form factor of the connection and the data transfer speed.

The form factor of USBs come in different sizes. The most popular ones are USB-A, USB-Micro, USB-Mini, and USB-C. Currently, the most widely used size

is USB-A, but USB-C is becoming more popular. USB-C is the newest USB form factor to be adopted, and is much faster and smaller than USB-A. Another thing to consider is the USB version. USB version 1.0 has a transfer rate of 1.5Mbps, USB version 2.0 has a transfer speed of 480Mbps, and USB version 3.0 can support speeds up to 5Gbps. Newer versions of the form factor have been made to support newer versions. This is done by adding more connector pins inside the USB. This allows USB-A to have some versions that are 3.0 compatible.

Another aspect to consider is that USB transfers power as well. In the scope of this project, this will not be relied on, but it is important to know that interfacing with USB has the possibility of sending too much power.

4.2.5 JST cable

JST stands for Japanese solderless terminal. Each connection can be part of a different series which are denoted by two letters (Ex: VH, RE, GH). Each series differs by pin to pin pitch, which is the distance between pins, by wire size, and by locking mechanism. JST cables are always one row wide of pins. Considering the scope of this project, it is safe to assume that at most JST series ZH, GH, or SH would be used if the mobile sensors require it. The mobile sensors include the sensors that cannot be soldered onto the PCB so that they can be appropriately positioned for proper measuring. The moisture sensor would be an example of a sensor that needs to be mobile.

4.2.6 Wire size AWG

In the United States, the standard for wire size is AWG, which stands for American wire gauge. For the largest gauges, wire AWG is standardized to 0/4-0/3 with 0/4 being the largest, and then for smaller AWG is numbered between 1-40, with 40 being the smallest. Each AWG has specifications for max current, and also standard resistance which depends on the wire material.

4.3 Battery/Power Standards

Batteries are often overlooked regarding the standards around them. In order to accurately include battery standard information you must take into account each aspect of the chosen battery. The more common batteries have more standards written around their use, disposal, and installation.

4.3.1 Nickel-Metal Hydride Battery Standards

In looking through the IEEE battery standards, there is not one that pertains specifically to the chosen Ni-MH (Nickel Metal Hydride) battery type. In researching other standards, we were able to find reference to IEC (International Electrotechnical Commission) 1441 97/204158 DC, a standard which “Gives marking, dimensions, tests and requirements for secondary batteries containing more than one sealed nickel-metal hydride rechargeable cell in series and incorporating a standardized interconnecting system to the device.” This

standard has since been withdrawn. To determine best practices with this type of battery I have referenced several sources. To understand Ni-MH batteries you have to understand both Nickel-Cadmium and Nickel-Hydrogen batteries. These were similar configurations in that they both use the nickel-hydroxide positive electrode and KOH electrolyte [31]. The difference between these two batteries was that the nickel-hydrogen battery uses hydrogen gas to replace the cadmium in the negative electrodes. These two batteries paved the way for the Ni-MH battery we know today.

4.3.2 Sealed Lead Acid Battery Standards

In looking through IEC standards, section 60896-11 is applicable to lead-acid cells and batteries which are designed for service in fixed locations. Given that this project will revolve around a plant that will be stationary I believe that this part of IEC standards will be applicable if we choose to use a sealed lead acid battery. Another standard would be IEC 62485-2 which references stationary secondary batteries with a maximum voltage of 1500 Volts. This standard goes into detail regarding electricity, gas emissions, and electrolytes. This international standard provides safety requirements associated with the inspection, maintenance, disposal, and use of either lead-acid or NiCd/NiMH batteries. IEEE 485 references methods for defining DC loads and sizing lead-acid batteries for stationary applications. This IEEE standard is only suitable for lead-acid battery types, and references Installation, maintenance, qualification, and testing procedures.

4.4 Software Standards

In this section we will go over the Software Standards we will follow in our project. Our mobile application and both the frontend and backend of our website application are built on javascript. We will also cover Software Testing standards and the frameworks we will use for testing. We will use Jest and Enzyme to thoroughly test our application.

4.4.1 Javascript Language Airbnb Style Standards

In this section we will go over the standards set by the Javascript Airbnb Style Standards, which is the style that is used in our linter, EsLint. These Airbnb style standards are used by many organizations such as General Electric, reddit, Zillow, and many more! These standards allow Javascript code on each platform to follow the same coding style to make the code as easy to read as possible. There are many areas of javascript which are standardized so we will not cover every single one. Instead we are going to focus on the main standards.

4.4.1.1 Objects

In Javascript objects are containers for a collection of key-value pairs. Each key-value pair is known as a property. In Javascript objects can be made by

either creating a new object with the “new” keyword or the literal syntax such as “const item = {};”. Airbnb style standards say to use the literal syntax for object creation and refrain from using the new keyword.

4.4.1.2 Variables

When declaring variables, always use “const” or “let”. When we just declare a variable without them a global variable is made and is unnecessary for a global variable. Another standard is to group all “const” and “let” variables together. The last standard we will cover for variables is to not allow unused variables. Any unused variables should be removed due to them taking up unnecessary space and are confusing to anyone reading the code.

4.4.1.3 Comments

For single line comments it is standard to use the “//” syntax and for multi line comments the standard is to use “/** …. */”. It is also standard to start all comments with a single space so it is easier to read that comment. Lastly it is standard to prefix comments with either FIXME: to annotate problems and code that needs fixed. Or a TODO: prefix to annotate solutions that need to be completed.

4.4.1.4 WhiteSpace

We use 2 spaces to indent code inside functions and 1 space before the leading brace. The standard is to place one space before the opening parenthesis in control statements such as if and while statements. Operators should also have spaces in between them such as “y = x + 2”. All of these whitespace standards make code a lot easier to read and manage.

4.4.1.5 Naming Conventions

The standard for naming variables, functions, etc. Avoid using single characters for naming such as a function named x. Use camelCase when naming objects, instances, and functions. Use PascalCase when naming constructors or classes. Lastly do not use trailing or leading underscores.

4.4.2 Software and Systems Engineering - Software Testing Standards

In this section we will lay out Software Testing Standards and different Software Testing frameworks to thoroughly test our application. Following testing standards is pertinent to ensuring our application works properly.

4.4.2.1 Jest

Jest is a popular JavaScript testing framework that is typically used to test React applications. Jest is widely used in industry for popular applications such as

Facebook, Twitter, Spotify, Instagram and more. Jest can also be used to test React Native applications. Jest focuses on simplicity and easy setup on your applications/projects. You can use Jest right out of the box with little to know configuration on your JavaScript projects.

We will use Jest as one of our Software Testing Standards due to its great features and it being widely utilized in the JavaScript application testing world. Jest has code coverage which shows which files are being tested and which are not in your project folder. It goes further in detail to show what percentage of your individual pages are covered by unit tests.

4.4.2.2 Enzyme

Enzyme is a JavaScript testing library that makes it easier to test your JavaScript application. Enzyme is paired with the above framework, jest, to fully test our application. Enzyme allows for shallow and full rendering of React Components to thoroughly unit test our code base. Enzyme was created and is used by Airbnb as a standard for testing which our team will adopt and use in this project as well.

Enzyme Shallow and full rendering are powerful tools that make testing a lot easier. Enzyme Shallow Rendering allows for a single react component to be rendered, only that component. So we can easily unit test that component. Then we can render in full that will render all components that are using a specific component to fully test our application and make sure everything works smoothly together.

Testing Environment	Core Function	Key Features
Jest	Jest is a testing framework which ensures our tests run properly. Jest ultimately handles the configuration, execution, and output of all tests for our application.	1. Jest is a task runner allowing tests to run properly in a testing environment. 2. Built in mocking features that work but are typically paired with Enzyme.
Enzyme	Enzyme is a testing library which is an add-on to Jest. Enzyme makes it easy to render and test components in our application.	1. Shallow Rendering of components to test a single component 2. Mounting of components to test the integration of each one.

Table 11: Testing Environment

4.4.3 Design Impact of Software Testing Standard

ISO/IEC/IEEE 29119 standard was considered for this design. The group will try to adhere exactly to these standards and demonstrate the fulfillment of the necessary requirements in order to claim full conformity to the standard. That being said, some of the techniques in the ISO/IEC/IEEE 29119 standard are not required or necessary for the function of this automatic irrigation system's design. Therefore, the team will adhere to tailored conformance of the ISO/IEC/IEEE 29119 standard. This means that the team will conform and demonstrate met requirements chosen by the team itself.

The organizational test processes will have a design impact on the software. Unit tests, possibly automated unit tests, will be created so that the code performs satisfactorily and to the expected standards. Computer Engineering members of the team will be expected to do more involved tests for this design but all members of the team will participate in user functionality type tests. This organizational test policy and the different strategies developed will be implemented by the team. All of the test techniques will adhere to requirements and standards and will be specified.

4.4.4 Programming Languages - Javascript

JavaScript is a popular coding language used to build various different applications. Most of our website and mobile application code base will be Javascript with the exception of HTML, CSS, and JSX. Developers use different coding styles and syntax however certain syntax is typically universal for the purpose of easier reading and understanding of the code. For example it's standard to have spaces between operators, operands and parameters. Without spaces between operators it makes the code very hard to read even though it will compile no problem.

4.5 Other Standards

This section includes standards that do not relate to wireless communication, wired communication, power standards, or software standards.

4.5.1 IPXX

IPXX is an international standard for waterproofing and protection. The amount of protection guaranteed is denoted by two digits after IP. The first digit rates how well the casing protects against non-moisture particles. The number zero denotes unrated, the number 1 denotes protection from objects no smaller than 50mm, two denotes protection from 12mm objects, three denotes protection from

2.5mm objects, 4 denotes 1mm objects, and 5 and 6 is protection against dust with and without a vacuum seal. The second digit determines how well the device is protected from water. This number can be 0-9K. A few notable numbers include zero which is unrated, one which is protection from vertically falling droplets, four which needs to survive splashing from multiple angles, and seven, which is full immersion for thirty minutes between 15cm and 1m. Due to time constraints, our project will most likely not get properly rated, but this standard will be considered when designing and prototyping the protective casing.

4.5.2 IPC-2221

The ICP-2221 is a set of standards for generic PCB design. This group includes IPC-2223, which includes standards for flexible PCBs (PCBs that can bend). There is IPC2225, which covers standards for PCBs with organic substrates called MCM-L PCBs. It also has IPC-2226, which goes over PCBs with 120-160 pins per square inch called HDI PCBs. Finally, IPC-2221 includes IPC-2222 which is for rigid PCB design, which is the style of PCB to be used in this system. All of these standards cover ways to make the PCB more reliable and easy to manufacture. An example of this would be the standards on clearance, which is the space between the tops of PCB traces. This standard is implemented to prevent electricity from arcing from one trace to the other. The clearance between two components depends on the specific components. For example, it is recommended to keep the clearance of leads to be 0.13mm. If a mask is applied to the traces, which means the metal is covered in an insulator, the clearance can be much smaller. The width of the traces need to be the proper size for the right current. If it is too small, the trace would heat up and melt. There are equations that can be used to ensure that the traces are the proper size. For higher voltages, the clearance and thickness need to be increased.

4.6 Realistic Design Constraints

These realistic design constraints are imposed on the design so that implementation can go smoothly. The team considered the various constraints shown below in relation to the design of this automatic irrigation system.

4.6.1 Economic and Time Constraints

There are economic constraints imposed on this design because it is a student funded prototype project. Also, after some brainstorming, the team came to the conclusion that the problem being solved (automatic irrigation for indoor and outdoor plants), is a relatively simple problem and therefore, it would be hard to justify a higher overall price for this project, especially a price that is higher than the required funds needed for its constituent parts. In short, the most advanced and expensive technology should not be required to solve a problem that is relatively simple. As seen in our industry research, some related automatic irrigation systems run into issues due to added features needing a higher overall cost in order to fund these features. Therefore, many of the desirable features of

this design are software based, so they do not require a significant amount of funding to add more value to the user. Avoiding an overdesign or overcomplicated design with added value due to software should be able to alleviate some of the economic constraints overall.

In terms of time constraints, the design, prototyping, and construction of this system should not exceed the deadlines established in the timeline portion of this report. Also, due to supply chain issues and the coronavirus pandemic, time constraints should be considered conservatively due to the possibility of delayed or subpar electronic parts required for the construction of this system. Time constraints must be imposed because ordered electronic parts have an estimated delivery time of multiple weeks or even months. To deal with this, prototype design is going ahead of schedule and parts are being ordered ahead of time from multiple vendors to deal with potential delivery delays and potential subpar quality in parts.

4.6.2 Environmental and Social Constraints

The design of this automatic irrigation system takes into consideration certain environmental, social, and political constraints. In a social sense, automatic irrigation systems on the market are currently typically pretty accessible at this point in time, so our design tries to remain close to the financial constraints imposed by other systems on the market. Also, this system is inherently a relatively accessible member of the Internet of Things in that its setup and required parts from the user are pretty accessible. Increased custom irrigation to separate plants is typically easy to accomplish with household scissors and the user usually only needs to provide a source of water and any plants that they wish to be irrigated. This system is not a “Do it Yourself” kit, so therefore, the user should be able to set up this system without much hassle. Environmentally, this automatic irrigation system is inherently valuable because its design is based around the idea of wasting less water. Timed irrigation can help the user promote better health for their plants as well as help prevent runoff, which could provide dangers to the electrical equipment, create a mess or danger for the user, or move unneeded nutrients to locations outside of the irrigated plant’s soil. Though this system requires the use of an electric outlet to charge the battery that maintains power, a considered design for this project uses solar panels as the main source of energy to charge the attached battery to maintain its own function.

4.6.3 Ethical, Health, and Safety Constraints

Several precautions have been taken due to the dangers associated with water and electronic parts. Much of the value of the system is reliant on the proper function of electronic parts, which unfortunately brings into consideration the possibility of water damage that will cause short circuiting or electrocution. In order to deal with this, electric contact points, batteries, and other components are insulated and properly grounded. Proper components to disperse heat and

protect against moisture are implemented into the design of the electronic boards. Electronic parts necessary for the implementation of the design are purposefully separate or farther away from parts of the system that are associated with the movement of water and moisture.

4.6.4 Manufacturability and Sustainability Constraints

Manufacturability constraints are mostly imposed by following the designs of automatic irrigation systems already on the market. Though manufacturing facilities for students are limited to university facilities, this should not be too much of a problem due to the design being easily implemented with household tool kits. Whether the design implemented is the “Snip N Drip” system or the one that requires a structured chassis, one important customer value for the automatic irrigation system is its relative modularity and ease of setup. This prompts a more “stripped down” or “minimalistic” approach to the design, which luckily for us, will simplify the manufacturability of our automatic irrigation system. Also, the actual problem that automatic irrigation systems solve is not overly complex, so this should prompt a relatively easy to implement design, which usually means an easily manufacturable product. The only real issue (and main issue overall) for manufacturability, other than the actual design of the PCBs, are the networking and connectability of the boards. The implementation of this system is dependent on the constituent parts connecting and operating together properly, so the key will be to manufacture a system that connects together smoothly.

In terms of sustainability, working with electronic parts for a system that can potentially support both indoor and outdoor plants leads to the potential problem of water’s corrosive and damaging nature to electronic parts. Much consideration is necessary for electronics intended to be used outdoors or intended to interact with water in any way. Failure to do so will lead to overall failure of the system that cannot be diagnosable and repairable with simple household appliances. Proper insulation and weather proof structures are implemented into the design to deal with potential moisture damage. Also, the design, though possessing electronic components connected to water pump components, is purposefully designed in a way so that the moisture being pumped can be set farther away from the delicate electronic components. The user manual will advise the user to, if possible, extend the provided wires on the design to full length so that this separation can take place. It is possible that the important electronic parts could be placed in a user’s porch or outdoor seating area, while the water pump and water source rests in another area, with both of these components being a couple feet away from the different plants actually being watered. This strategy will be detailed and pictured in the user’s manual for advised and intended use of the automatic irrigation system.

5.0 Part Selection

In this section, we discuss the parts selected to comprise the project. These parts were chosen based on the previous selection and standards discussed in the previous sections. The main components chosen are motor controllers, batteries, buck converters, temperature sensors, soil moisture sensors, light sensors, wi-fi modules, bluetooth modules, pumps, and MCU modules.

5.1 Motor Controllers

Given that this project heavily relies upon controlling a pump (motor) we have to take the controller of that device into consideration. There were a lot of different factors to take into account when deciding what motor controller should be chosen, such as voltage, amperage, wattage, if it can handle more than one motor at a time, and how it can interface with our chosen MCU.

5.1.1 Icstation 5A 3V-14V Dual DC Motor Drive Controller Board Module Motor Commutation PWN Speed Regulator Dual H Bridge

One of our options for a pump controller was the Icstation, boasting that it was “better and more stable than the L298N”. The main difference between the two boards was centered around the current the board can withstand without heating up. The Icstation can independently control two DC motors, or in our case pumps, and can also control one 4 wire two-phase stepper motor. Each channel can output 5 Amperes with a peak of 9 Amperes and the standby current is only 10 micro-amperes. The board provides a 3.3 volt, 100 mili-ampere power supply to the control system, and allows for interface with an arduino board. The power supply ranges from an input of 3 volts to 14 volts, and an output of 2.2 volts to 6 volts. This board has Pulse Width Modulation (PWM) control, which is necessary for our pump, in order to control the speed at which our pump will disperse water. In reading the datasheet this Motor drive controller board is meant for connecting to a remote control module, which is not necessary or ideal for our project.

5.1.2 Qunqi L298N Motor Drive Controller Board Module Dual H Bridge DC

Similarly to the Icstation option, the Qunqi L298N Motor Drive board is a dual channel H bridge driver allowing the control of two separate DC motors. The Qunqi board uses large capacity filter capacitors in order to provide freewheeling protection and increase the reliability of the board. The current can reach 3 amperes peak, and continuous current is around 2 amperes. The maximum output power is 25 watts, which would allow us plenty of power for the pump we have chosen. Similarly to the other board, the Qunqi has Pulse Width Modulation control. The Qunqi L298N is the better of the two options for the motor drive board.

5.1.3 Cytron 13A, 5-30V Single DC Motor Controller

Another motor controller is the Cytron 13A Single DC Motor controller. This is designed to drive high current brushed DC motors up to 13 Amps continuously. It supports a single motor with bi-directional control, and allows for motor voltage ranges from 5 to 25 Volts. It comes with a full NMOS (N-type Metal Oxide Semiconductor Field Effect Transistor) H-Bridge, which boasts high speed, low heat dissipation, and overall increased efficiency over BJT (Bi-polar Junction Transistor) H-Bridges. [Demonstrating Motor Control Using Nmos…]

5.1.4 Table Comparing Motor Controller

Below is a table comparing each motor controller, with our chosen motor controller highlighted.

Feature	Icstation	L298N	Cytron
Input Voltage (Volts)	2.2-5	5-25	5-25
Motor Current (Amps)	5	2	13
Temperature Range (Celsius)	-20 - 135	-20 - 85	NA
Dual Motor	Yes	Yes	No

Table 12: Motor Controller Comparison

5.2 Batteries

The battery we choose must be capable of supplying our pump and electronics with enough power, as well as being able to keep them powered for long durations. Ideally the battery would be rechargeable, and would not be too heavy as our design is meant to be compact.

5.2.1 Tenergy NiMH Battery Pack 12V 2000mAh

Once we determined a Nickel-metal hydride battery would be ideal, one of the most compact options was the Tenergy NiMH battery pack. This fulfilled our

criteria of being 12V, rechargeable, cost effective, compact, and minimal weight. Another benefit was the fact that the battery would come with bare leads, allowing us to choose how we would prefer to wire our devices.

5.2.2 ExpertPower EXP1250 12V 5Ah

The ExpertPower EXP1250 12V battery boasts 5 Amp hours of battery life, which is estimated to last 20 hours of continual use. This is a sealed lead-acid (SLA) battery, which has significantly more standards than the NiMh battery listed above. These standards would give us a lot of information as to the best practices if we chose to use this battery. Given that this battery is significantly larger, weighs more, and would not allow us to store it given the dimensions we have specified in the engineering specification table, it is not a good option for this project.

5.2.3 Amazon Basics 9 Volt Performance All-Purpose Alkaline Batteries

The Amazon Basics 9 Volt battery is another option. It was tested for its performance by rightbattery.com, and was shown to produce a charge of 481 mAh at 50 milli-Amp load. If used in parallel to extend battery charge this battery would still be less cost effective, as it would take more than 4 batteries in parallel to equal 2000 mAh. Given that this charge is quite low compared to our other options for batteries, and that this battery is not rechargeable, this battery would not be an ideal candidate for this project.

5.2.3 Battery Comparison

Below is a table comparing the different battery options we looked at.

Feature	ExpertPower 12V	Tenergy 12V	Amazon 9V
Voltage (Volts)	12	12	9
Electric Charge	5000	2000	481
Rechargable (Y/N)	Yes	Yes	No

Table 13: Battery Comparison

5.3 Buck Converters and Voltage Regulators

Rather than implementing resistors and other devices to make a buck converter, or voltage regulator from scratch, it initially thought it would be easier and more cost effective to order a buck converter. The goal of having a stand alone buck converter would allow us to regulate the voltage from 12 Volts down to a voltage the MCU and sensors would need which would be anywhere from 3.3 Volts to 5 Volts. In the final implementation we did not use the buck converter but instead used two linear voltage regulators.

5.3.1 Valefod 6 Pack LM2596

This buck converter gave us a large range of input voltage as well as output voltage to work from, as well as coming in a pack with multiple units so we can perform testing on them. Input voltage ranges from 3 Volts to 40 Volts, and an output voltage range of 1.5 Volts to 35 Volts, this would allow us many options as to how we would prefer to power our different devices. In the table below, we see that the input voltage range is the largest, and the price per unit is the smallest, so this is the most economical choice for our project.

To test this module, I connected two pumps, to a breadboard, with input power from the Tenergy 12 volt 2000 mAh battery, and I was able to have both the 12 volt pump on, as well as the 5 volt pump on, and I was able to adjust the 5 volt pump in using this buck converter, and a screwdriver to lower and raise the voltage as I required.

5.3.2 HiLetgo 2pcs LM2596

This buck converter would give us slightly less input options while allowing us more output options, but only a two pack. The added benefit of having a voltmeter display could be beneficial though as we could adjust voltage as necessary rather than requiring an external voltmeter.

5.3.3 eBoot Mini MP1584EN DC-DC Buck Converter (6 pack)

The eBoot mini buck converter allows input voltage from 4.5 Volts to 28 Volts and would allow an output voltage from 0.8 Volts to 20 Volts. Boasting a 92% conversion efficiency rate that is within our ideal range of 12 Volt input voltage and 3-10 Volt output voltage, that makes this a good candidate for our project. This component is usually inexpensive and will cost about 2$ before added shipping costs. The voltage will have to be changed with a screwdriver with the guidance of a multimeter that will act as the calibration component of this buck converter.

5.3.4 LM7805

The LM7805 is a voltage regulator that outputs 5V. It has three terminals and is a through hole component. It can output up to 1.5A and has an input voltage range of 7V to 25V. It is inexpensive and widely available.

5.3.5 PDSE1-S12-S3-D

The PDSE1 is a voltage regulator that outputs 3.3V. It has four terminals and is a through hole component. It can output up to 303mV and has an input voltage of 10.8V to 13.2V. It is widely available but has a variance of up to 20%.

5.3.6 Table Comparing Buck Converters and Regulators

Feature	Valefod	HiLetgo	eBoot Mini	LM7805	PDSE1
Input Voltage Range (Volts)	3 - 40	4 - 40	4.5 to 28	7 to 25	10.8 to 13.2
Output Voltage Range (Volts)	1.5 - 35	1.25 - 37	0.8 - 20	5	3.3
Price Per Unit ($)	1.83	5.25	2.83	0.69	3.12
Maximum Output Current (Amps)	3	3	3	1.5	.303
Operating Temperature (Degrees Celsius)	-45 to 85	Not Listed	-45 to 85	0 to 125	-40 to 105

Table 14: Buck Converter Comparison

5.4 Temperature Sensors

Given that there are several different types of temperature sensors on the market, this section will go into the details about each specific sensor and what each one would benefit our project. We took input voltage, amperage, accuracy, and compatibility with our MCU into consideration.

5.4.1 MCP9808

The MCP9808 is a digital temperature sensor. This means that it outputs a digital signal and is I2C compatible. This sensor is advertised to be able to read temperatures between -20C and 100C with a typical error rate of plus or minus 0.25C. It has an operational voltage range of 2.7-5.5V and a max voltage of 6V. This device has two pins for power, two pins for I2C and an alert pin that can notify the user when a certain temperature is reached. Before taxes and shipping, this sensor costs about 5 USD.

5.4.2 TMPXXX

The TMP family of temperature sensors is Texas Instrument's temperature sensor selection. The following were chosen based off of if the parts were still actively being produced and if they were appropriate for this system:

5.4.2.1 TMP275

The TMP275 outputs a 9-12bit digital signal and is I2C compatible. This sensor can read temperatures between -20C and 100C with an error rate of plus or minus 0.5C. It has an operating voltage of 2.7V-5.5V and a max voltage of 7V. This sensor has 2 pins for I2C, 2 pins for power, and 1 pin for alert. Before taxes and shipping, this sensor costs around $3 USD.

5.4.2.2 TMP112X

The TMP112X comes in a few variants in which the TMP112A and the TMP112B will be considered. They both read temperatures between 0C and 65C with an error rate of pulse or minus 0.5C. Where they differ is the operating voltage. The TMP112A best operates at 3.3V while the TMP112B best operatues at 1.8V. They both have the voltage range of 1.4V to 3.6V with a max voltage of 4V. This sensor has 2 pins for I2C, 2 pins for power, 1 pin for an alert, and one pin for address select. These sensors are NIST traceable.

5.4.2.3 TMP126

The TMP126 outputs a 14-bit digital signal and is SPI compatible. This sensor has an error rate of plus or minus 0.3C for temperatures between -20C to 85C. This sensor has an operational voltage of 1.62V to 5.5V with a max voltage of 6V.

This sensor has 6 pins, two pins for power, one pin for clock input, one pin for chip select, one pin for SIO communications, and one alert pin.

5.4.3 LM92

The LM92 is a digital sensor that outputs a 12-bit signal and is I2C compatible. This sensor can measure temperatures between -10C and 85C with an accuracy of plus or minus 1C. It has a better accuracy of plus or minus 0.5C for temperatures between 10C to 50C. This sensor has an operational voltage of 2.7V to 5.5V with a maximum voltage of 6.5V. It has 8 pins, two for SDA and SCL, two for power, two for address select, one for critical temperature alert, and one pin for interrupt output drain output.

5.4.4 BME280

The Adafruit BME280 is a digital temperature, humidity, and baramic pressure sensor that is I2C or SPI compatible. This sensor can measure temperatures between -40C and 85C with an accuracy of plus or minus 1C, can measure atmospheric pressures of 300hPa to 1100hPa with an accuracy of plus or minus 1hPa, and can measure the humidity as a percentage between 0%-100% with a plus or minus 3% accuracy. This sensor has an operating voltage of between 1.71V and 3.6V and the digital interface has an operating voltage of 1.2V to 3.6V. This sensor has 7 pins. Three pins are for power, one is ground, one is the supply voltage for the sensors, and the last is the supply voltage for the digital interface. The remaining four pins are used for communications which includes a serial clock pin, a chip select pin, a serial data out pin, and a serial data in pin. I2C mode is achieved by tieing the chip select to the voltage pin for the digital interface. This sensor costs about 15USD.

5.4.5 Temperature Sensor Table

Part	Interface	Accuracy	Special Function	Cost
MCP9808	I2C	0.25C	alert pin	5$
TMP275	I2C	0.5C	alert pin	3$
TMP112X	I2C	0.5C	Alert pin	3$
TMP126	SPI	1C	alert pin	2$
LM92	I2C	1C	alert pin	6$
BME280	I2C OR SPI	1C 1hPa 3%humidity	Measures Pressure and Humidity	15$

Table 15: Temperature Sensor Comparison

5.5 Soil Moisture Sensors

There are not as many different soil moisture sensors available as there are temperature sensors. This section describes the differences between each soil sensor and a rough depiction of how they sense the moisture level of the soil they are placed into.

5.5.1 STEMMA Soil Sensor

The STEMMA soil sensor is a capacitive digital moisture sensor that is I2C compatible. It outputs 200 for very dry and 2000 for very wet. Since it is capacitive, there is only one prong. This sensor also comes with a temperature sensor with an accuracy of plus or minus 2C. This sensor requires a 4PH JST connector. The four pins include a ground, a Vin, SDA, and SCL (last two for I2C). The operational voltage is 3V-5V. This component costs about 8USD.

5.5.2 PR46-7 Soil Sensor

This soil moisture sensor is a resistive digital moisture sensor that is I2C compatible. It has 12 bits of resolution and outputs values between 0-4095. This sensor has an operating voltage of 5V. This sensor has two sets of 4 pins. The four pins include a ground, Vin, SDA and SCL for I2C. The second set is to chain other sensors to itself. The major downside to this sensor is that it costs $30 USD.

5.5.3 Songhe Soil Moisture Sensor

This soil moisture sensor is a capacitive analog sensor. It has an operational voltage of 3.3V to 5.5V. This sensor has 3 pins, two for power, and one for analog out. The output voltage of this sensor is between 0V-3V. The pins support JST cables. These sensors are cheap compared to the other soil sensors costing only 2.3USD.

5.5.4 Soil Moisture Sensor Table

Part	Interface	Type	Special Function	Cost
STEMMA	I2C	Capacitive	Temperature Sensor	8$
PR46-7	I2C	Resistive	-	30$
Songhe	Analog	Capacitive	-	2$

Table 16: Moisture Sensor Comparison

5.6 Light Sensors

Light sensors are a part of our stretch goals, but it is a consideration we must account for as it would be beneficial in our project. Each light sensor has slightly different characteristics which are outlined below.

5.6.1 TSL2591

The TSL2591 is a light sensor that outputs digital signals and is I2C compatible. It can detect light ranges between 188uLux and 88,000Lux. The input voltage is from 3.3V to 5V. This sensor has 5 pins, two for power, two for I2C and an interrupt open drain output pin. This sensor is also JST compatible. This component costs about 7USD.

5.6.2 OPT4001

The OPT4001 light sensor that interfaces with I2C. It can detect and output digital signals for light levels between 312.5uLux and 83kLux. This module has an operational voltage of -0.5V and 6V. This sensor has 4 pins, two for power and two for I2C communications. These lines need to be pulled up to Vcc. It is recommended to do this with a 10K ohm resistor connected to the voltage source.

5.6.3 XINGYHENG Photosensitive Sensor

The XINGYHENG Photosensitive Sensor Module Digital Light Intensity Detection uses a sensitive photoresistor sensor. This sensor uses a comparator output to determine if it will produce a 0 or 1. The operation voltage ranges from 3.3 Volts to 5 Volts and uses a LM393 comparator. The only application for this light sensor would be to detect if the light hitting the plant and or electronics (depending on where the sensor is located) is sufficient enough to send a 1 output, which could either tell the MCU it is day time or night time. Given these design constraints this is not an ideal candidate for our project.

5.6.4 Light Sensor Table

The table below looks at several different aspects of each of the light sensors listed above and compares them. The chosen sensor is highlighted. It was ordered but never implemented due to time constraints.

Part	Interface	Accuracy (Lux)	Special Function	Cost per unit ($)
TSL2591	I2C	188u - 88,000	JST compatible	7.00
OPT4001	I2C	312.5u - 83,000	Small	1.26
XINGYHENG	Hardware	Not applicable	Simple	1.20

Table 17: Light Sensor Comparison

5.7 Wi-Fi and Bluetooth Modules

In this section we discuss the different issues with choosing either a Wi-Fi module or a Bluetooth Module. Ideally, with the engineering specifications and the objective listed above, we would prefer a Wi-Fi module as we want the user to

5.7.1 ESP8266

The ESP8266 is a collection of different Wi-Fi modules, each with their own set of features. Some of these parts might not be available for testing, and will affect which one will be selected in the end. All of these Wi-Fi modules will support peer to peer connection. The major differences between the modules will be size, supported input/output pins and interfaces, such as SPI, UART, and I2C, and the amount of memory and processing power available to use. These modules will most likely only be used for their Wi-Fi capabilities and will most likely fall under the main MCU.

5.7.1.1 ESP-01

The ESP-01 is a relatively simple module. It supports Wi-Fi protocols 802.11 b/g/n, and its peripheral bus supports UART. The network protocol it supports is IPv4. The operating voltage is 3V to 3.6V. Its size is 14.3mm X 24.8mm X 3mm. There are 8 pins, two for power, two for transceiving and receiving, a reset pin, a chip enable pin, a reset pin, and two other GPIO pins. There is an external 1MB flash that uses SPI. This module contains its own MCU and clock. It also supports three different low powered modes.

5.7.1.2 ESP-12E

The ESP-12E is a more advanced chip with a multitude of features. It can host an application on the board from an external 4MB flash that communicates with SPI. It supports SPI, UART, and I2C and has a 10-bit analog to digital converter. The Wi-Fi protocol it supports is 802.11 b/g/n and the network protocol it supports is IPv4. Its operating voltage is 3V to 3.6V. It has a 32 bit processor and a crystal clock. This module supports three different low powered modes. This device can use 11 GPIO pins. There is a version of this that comes with an advanced development environment, which can be used to test out the module before choosing it to be fully integrated into the final PCB. Just the basic chip is also available.

5.7.2 ESP32

The ESP32 is the successor to the ESP8266 line of Wi-Fi modules. It offers both Wi-Fi 802.11 b/g/n support and Bluetooth v4.2 support. Its MCU can come with a 1 or two core processor and supports pipelining and floating point integers. It comes with two crystal oscillators and two RC oscillators. It has 34 GPIO pins that can support I2C, SPI, and UART. It has a supply voltage of 2.3V to 3.6V. The dual support of Bluetooth and Wi-Fi makes this module sit outside the scope of this project.

5.7.3 CYBLE-333074-02

The CYBLE-333074-02 is a bluetooth module that interfaces through SPI or I2C. Its operational voltage is between 2.5V and 3.6V. It uses Bluetooth 3.0 and has a max data transfer rate of 1Mb/s. This module costs approximately $11 USD.

5.7.4 NINA-B221-03B

The NINA-B221-03B is a bluetooth module that interfaces with UART. It operates between 3V and 3.6V. With a max data transfer rate of 1Mb/s, this module uses Bluetooth 4.2. It costs around $13 USD per module.

5.7.5 BGM220SC22HNA2R

The BGM220SC22HNA2R is a bluetooth module that interfaces with SPI, I2C, and UART. Its operational voltage range is between 1.8V to 3.8V. This module uses Bluetooth 5.2 and supports data speeds of up to 2Mb/s.

5.7.6 Wi-Fi and Bluetooth Table

Part	Interface	Protocol	Special Feature	Cost
ESP-01	SPI, UART, I2C	b/g/n	Simple	3$
ESP-12F	SPI, UART, I2C	b/g/n	Easy to interface	2$
ESP-32	SPI, UART, I2C	b/g/n	Advanced processor, Bluetooth	4$
CYBLE-3330 74-02	SPI, I2C	4.2	1Mbps	$11
NINA-B221-0 3B	UART	4.2	1Mbps	$13
BGM220SC22HNA2R	SPI, I2C, UART	5.2	2Mbps	$10

Table 18: Wi-Fi and BlueTooth Table

5.8 Pump Selection

We dive more into pump selection in the testing phase of the project. Picking a good pump will be crucial to ensure every plant gets a sufficient amount of water.

5.8.1 12V Mini Brushless DC Water Pump

The 12V Mini Brushless water pump comes in either 6 Watt or 7 Watt options. The maximal input current is 600mA, the pump is able to handle fluid temperatures from 0 degrees Celsius to 65 degrees Celsius, and the pump capacity is stated as being 2.2L a minute. This is a submersible type pump, where it will sit in the chosen water container, and we will only require a tube going out to the plant. The suction caliber is estimated to be 9mm, and the discharge caliber is estimated to be 6.8mm.

5.8.2 Sipytoph 4Pcs DC 3-5V Micro Submersible Mini Water Pump

The Sipytoph DC 3-5 Volt Submersible Mini Water Pump comes in a 4 pack with two sets of tubing. We chose this pump in the selection process as it would allow us to test multiple voltage levels and allow testing of the pump's throughput, or capacity. The load rated current is 180 milliAmps, the flow rate is 2 liters a minute (120L/H), and is only suitable for experiments not recommended for continuous use. The diameter of the water outlet is 4.5mm and the inlet diameter is 5mm.

5.8.3 LEDGLE Mini USB Fountain Pump Compact Submersible Pumps Efficient 5V

The LEDGLE Mini USB 5V Submersible Pump was chosen as a backup option if we thought we would require a USB pump. This pump can be powered by any USB capable device, from a power bank, solar charger, or a USB wall socket. This pump runs off 5 volt DC power, and boasts an estimated 3 Liters per minute (180 Liters per hour). Given the USB requirements we felt it was not the ideal candidate for this project.

5.8.4 Table Comparison for Pumps

The table below shows a comparison of the pumps listed above, and the selected device is highlighted in the table. The choice was determined based on performance, availability and compatibility with the rest of the system that will be based off of a 12V battery source. This pump will have to perform well in the demonstration.

Feature	12V Mini Brushless	Sipytoph	LEDGLE Mini USB
Voltage (Volts)	12	3-5	5
Amps(Milli-Amps)	600	180	300
Price Per Unit ($)	15.96	2.85	11.58
Estimated Throughput (Liters/minute)	2.2	2	3

Table 19: Pump Comparison

5.9 MCU Options

A microcontroller was picked from the team's selection process. This section focuses on the different development boards that were considered. Numerous types of chips and development boards could be considered for this project. For example, though they did not make it through the initial selection process and were no longer considered, the team did actually consider using an Arduino Uno or a NVIDIA development board of some kind. However, it was eventually decided that the team's familiarity with the MSP 430 family of boards would be the best microcontroller option for the project.

This family of microcontrollers are beneficial for the team due to the team's previous use of them in the Junior Design lab class. The MSP430 has relatively easy to use software with an IDE that could make programming easier. There are

large amounts of beneficial tutorials online to deal with any simple troubleshooting for this MCU. Similar to other boards for development such as Arduino, Raspberry Pi, NVIDIA, Beaglebone, and Samsung, MSP430 and its family has appropriate software libraries to access important functions. These functions are important for the actual functional software for this automatic irrigation system.

Lastly, the MSP430 family's extensive software library also means that it is highly compatible with various sensors that will be required for the function of this automatic irrigation system. System integration will be the hardest aspect of the project for the team. However, this can be alleviated by the different components that are already compatible with the MSP430 for the automatic irrigation system being designed.

Overall, the MSP430 is the best fit for the team's uses and design of the automatic irrigation system. Though other boards and chips such as the Arduino, Raspberry Pi, NVIDIA, Beaglebone, and Samsung family of products could probably achieve or even surpass the abilities and functionality of the MSP430 board, given the time constraints and the team's familiarity, the MSP430 will be utilized for this project.

5.9.1 MSP430FR247x

The MSP430FR247x is a 16 bit microcontroller that is designed by Texas Instruments. It operates between 1.8V and 3.6V. This model includes several clocks, including a 32kHz RC oscillator, a 16MHz oscillator, a 10kHz oscillator, VLO, MODOSC, and a 32kHz crystal oscillator. This module also includes a 12 bit ADC with an input voltage of 0V-3V and 12 channels that can be switched between for multiple analog inputs. This version of the MSP430 chip also has 40+ GPIO pins for digital input and interrupts. This microcontroller supports I2C, SPI, and UART. It can operate in many low powered modes, making it ideal for embedded systems. This version comes in different packages, which differ between 48 pins to 32 pins.

5.9.2 MSP430F552x

The MSP430F552x is a 16 bit microcontroller that has similar features to the MSP430FR247x. The differences lie in how many clocks it has, low powered modes, and ADC input channels. The F552x only has one ADC input, so it requires an external MUX if multiple analog devices need to be used. There are much fewer oscillators on this chip, which include VLO, REFO, a 32kHz crystal oscillator, and a 32MHz crystal oscillator. The feature that this chip has over the FR247x is USB support.

5.9.3 MSP430FR6989

The MSP430FR6989 is a board that was not considered for the final product but rather for prototype testing since it is very similar to the current choices. It is a Texas Instruments Incorporated product that is well respected in development and electronics circles. This specific board type is used in several classes at the University of Central Florida. This is beneficial to the team because due to this, most of our teammates have this board already on hand and therefore, the familiarity lets us rapid prototype with breadboards fairly well. In the Embedded Systems class at the University of Central Florida, this development board was studied in depth. Its components, datasheets, and programming were all discussed in class.

The MSP430FR6989 uses the same MSP430 microcontroller as other models mentioned. In terms of its statistics, it requires 3V with a power consumption of 210 to 1845 microamps. It has 83 I/O ports that will be useful for attaching sensors. It has a clock rate of 16MHz with 128KB of flash memory and 2KB of SRAM. However, this board has no wifi module, which could lead to issues with testing, especially when dealing with the website and mobile application aspects of this automatic irrigation system.

However, there is lower risk for the team because as mentioned, most of the team has this board already on hand and if there are major damages or if the team decides to discard this model, the model only costs $20. Also, this microcontroller board has benefits in that it has very low power consumption, which leads to an overall longer battery and overall product life span.

5.9.4 PIC24FV16KM204

The PIC24FV16KM204 is a 16-bit MCU made by Microchip. The operational voltage is from 1.8V to 3.6V. This family of chips can have up to 4 ADCs, three of which are 8-bit and one which is 12-bit. It has a 8MHz oscillator which is RC and a 32kHz oscillator as well. This chip has a built in temperature sensor that has its own A/D conversion. SPI, I2C, and UART are supported in this chip. This MCU has an overabundance of ADC options and not a lot of timers.

5.9.4 ATMEGA328P-PU

The ATMEGA328P-PU is an 8 bit microcontroller which supports the Arduino environment. It operates between 1.8V and 5.5V and supports SPI, I2C, and UART. It has two 10-bit ADCs. It requires an external 16Mhz crystal to operate which needs two 22pF capacitors.

5.9.4 MCU Table

Part	Pins	Clocks	ADC	Cost
MSP430FR247x (MSP430FR2475TRHBT)	40	6	12-bit	$6
MSP430F552x	47	4	12-bit	$7
ATMEGA328P-PU	28	External	10-bit	$2.87
PIC24FV16KM204	44	2	80-bit, 12-bit	$4
MSP430FR6989	83	6	12-bit	$20

Table 20: MCU Comparison

5.10

Wall Power DC Jack Connector

During testing we implemented a new power connector for added functionality for the end user. This was decided due to small issues with the battery. This wall adapter used a standard plug with no ground, and would provide a stable 12 volt input and connected via a DC jack male connection into our PCB's DC female port.

6.0 Project Prototype Testing Plan

This section details the plan for the testing phase of this project. This methodology will be utilized to determine the effectiveness of our design as well as the functionality of the different components present in the design. The tests are separated into hardware and software sections. As the building of the official design and final project progresses, these tests will change and will be modified based on any issues that may occur while implementing the design.

6.1 Hardware Testing Environment

The environment that was used for initial breadboard testing will be the labs available at University of Central Florida, mainly those designed for the Senior design course. The Senior Design lab on the fourth floor of the engineering building was one of the more important environments utilized. The Texas Instrument Innovation lab will also be useful. Located on the first floor of the same engineering building, this lab will be utilized for breadboard testing, multimeter, soldering, and oscilloscope measurements. Also, the new makerspace in the University of Central Florida library will be useful due to the vinyl laser cutters and 3D printers which were used to make our electronics box. Due to this area being a relatively new space, this will be less known by other

students, which will allow us to have more time in the lab to construct our automatic irrigation system. This area will particularly be helpful when constructing the water resistant container for the hardware and PCBs.

6.2 Component Specific Testing and Integration

In order to ensure the correct function of each component, each will be tested individually utilizing a breadboard and power supply. The individual plans to test these different requirements are shown below. The verification process includes testing the part on a stable power supply and establishing communications. Once a communication link was established, the data received was investigated for its accuracy. Once the individual function of these components was verified, the team began integration to find if these components functioned correctly once combined. This meant making sure the parts worked at the same time and did not conflict with each other. All of the techniques to complete these tests were updated through the progression of the project.

6.3 Water Pump Testing

The water pump is integral to the function of the automatic irrigation system. It is imperative to properly test the functionality of the chosen water pump. It is important because it is the only feature of the project that is directly responsible for the health of the plants and therefore, the success of the project with solving the main user problem. The water pump must be able to not only function but also be controlled to meet the three different settings of our automatic irrigation system: Temperate, Tropical, and Cactus/Succulent.

The actual testing of this system will consist of the following:

1. Appropriately connect the water pump to a power source and vinyl tubing to a source of water
2. When connected to the power source, the objective of the test was to test whether the flow rate of the water pump remains consistent given different volumes of water from the water source and what threshold that this consistency holds up to. This can be easily measured by measuring the amount of water, in mL, coming out of the pump under a controlled amount of time given different volumes of water in the water source.
3. The ideal result of this test will be the water pump consistently dispensing the same volume of water under a certain time period regardless of the amount of water in the water source.

6.4 Moisture Sensor Testing

The moisture sensor is important because the overall moisture of the soil is the main indicator of plant health that we are measuring. The system as a whole would not work properly if it could not detect moisture levels in the soil. The water pump and moisture sensor create a “closed loop” of functionality in this sense.

The following steps were required to test the moisture sensor:

1. Record the digital value of the ADC connected to the soil moisture sensor when the sensor is dry and only touching air. This value will be the “completely dry” or 0% soil saturation.

2. Place the soil moisture sensor in a cup of water. Record the digital value of the ADC. This will be the “completely soaked” value, or 100% soil saturation.

3. Place the soil moisture sensor into a known amount of soil. Record the digital value of the ADC. Start adding known amounts of water to the soil, and continue recording the ADC’s value for each amount of water. Make sure to wait a few minutes between each recording to let the water reach equilibrium in the soil. Continue this process until the soil is at maximum saturation and water starts leaking outside of the pot.

There were a few issues with this method of testing. The first issue comes in the first step. When recording the value of the moisture floor, it is not actually 0% moisture since the sensor is being affected by the humidity of the air. This does decrease the accuracy only a little bit, since the soil moisture sensor is not super sensitive. Due to the climate of Florida, which on average has 74% humidity outside, this needs to be considered. It is recommended to calibrate the soil moisture sensor inside to avoid issues like this.

The second issue comes in the last step and has to do with test result accuracy. The amount of soil has to be measured with its volume and not its mass. This is because in Florida, it is very hard to find soil that is completely dry. This means that when weighing the soil, the water that is already in the soil will bias results a lot due to it being heavy. This means the soil should be measured volumetrically. It is important to note that this means that the type/consistency of the soil is not taken into account. This will produce different graphs, as some soil will not be able to retain much water as other types of soil. The retention rate and maximum amount of water allowed is not as important, as it does not play a role in calibration.

6.5 Temperature Sensor Testing

The temperature sensor testing required a very simple test. This test included us making sure that the I2C protocol is working and that the temperature sensor was returning valid results. If the temperature sensor is returning junk values, that means the proper connections have not been made, and the contacts need to be fixed. Once the proper connections have been made, the ambient temperature will be measured and compared to the thermostat. Next, the temperature sensor will be taken outside and measured against the reported weather. Finally, the temperature sensor will be touched and measured against the average human temperature. This is the simplest way to do basic testing without extensive equipment or setups.

6.6 Power System Testing

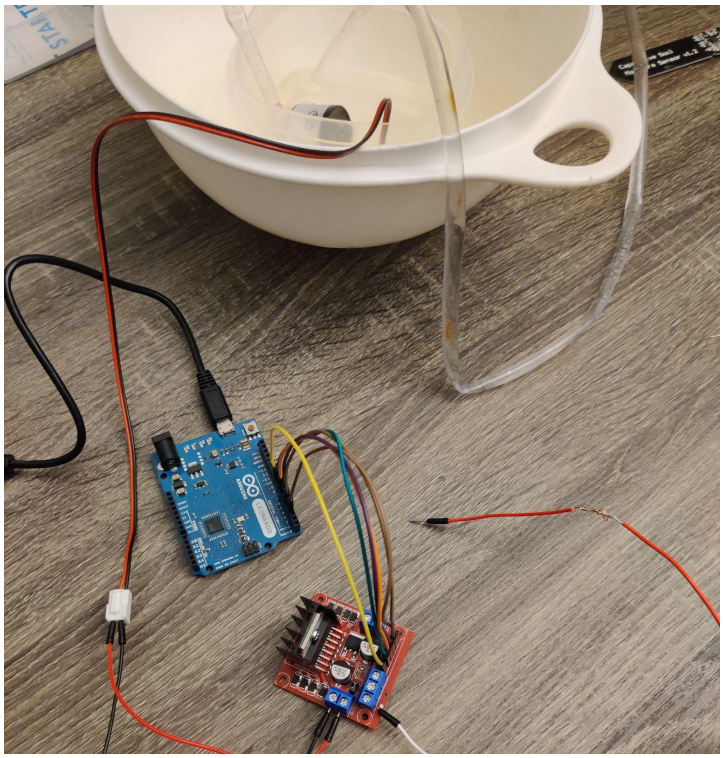


Figure 21: Power System Testing

In order to ensure our hardware components work well together a full power system test was conducted. In order to perform this testing we required everyone to meet up in order to work on the wifi module subsystem and the pump subsystem. In testing these subsystems separately we were successful in ensuring wifi connectivity from the wifi module to the computer. We were also successful in testing the pump (motor) controller. This motor controller testing is vital to the efficacy of our project as if we have no way to regulate the pump output into the plant our system will not function as we intend it to. To test the pump controller we experimented with the MSP430FR6989 board, as well as an Arduino Leonardo. The MSP board was problematic as we could not find or create functioning code for the board, whereas the Arduino had many more user friendly libraries and implementation help. We were able to test the pump controller by flashing a simple RC car demonstration code to the Arduino, and plugging in the 12V battery to the controller and the pump. We were able to successfully run the pump on a 5 second off and 3 second on loop, with the pump pushing water through our chosen tubing. In continuing to test the pump controller we were able to adjust the duration and speed at which the pump expelled water. In attempting to interface the pump subsystem and the wifi module subsystem, we were unable to implement them together due to the size restraints of the wifi module and in housing the wifi module on styrofoam not a

breadboard. In between semesters we plan to test everything together to ensure cohesion between the subsystems.

6.7 Software Testing Environment

The software components are integral for the function of the system. The system software will be divided into different sections and levels to make testing easier. Also, basic unit tests and procedures will be applied during actual development so that the least amount of bugs can occur. This method will be a combination of Top Down development and Incremental Development with Unit Testing.

The actual testing will take place on the personal computers of the team members. Members of the team with Android devices will use their devices to test the Android application. After actual debugging and deployment, fine tuning will have to occur on the actual user devices in order to better tailor the web and mobile applications. Due to testing occurring on the team's personal computers and devices, software testing will not have to actually occur in a specific area. However, when software integration into the wifi and microcontroller modules begins, this software testing will take place in one of the University of Central Florida engineering labs.

The chart below is a visual detailing the software testing process for the automatic irrigation system.

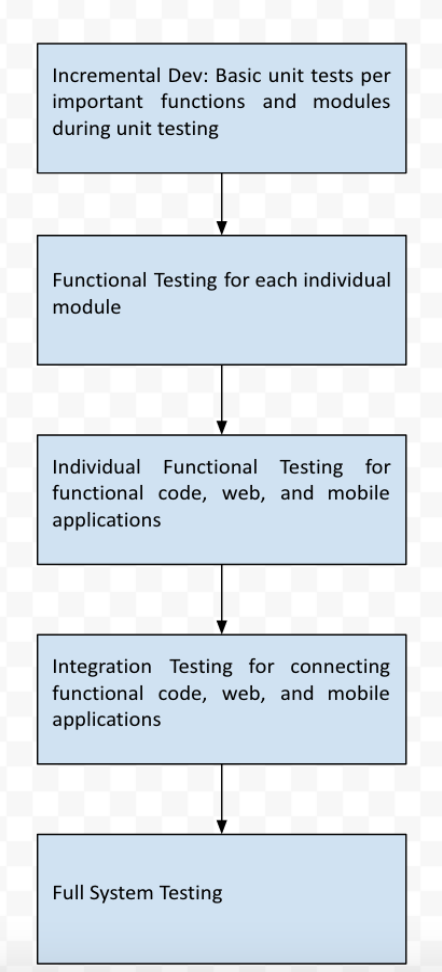


Figure 22:Software Testing Process

6.8 Software Specific Testing

This software system consists of web and mobile applications, database, and server. Testing will technically take place during development, in which each sub-module will be tested for basic functionality as each function is created. Then, the separate components will be tested for functionality. The more difficult test will be getting all of these components to integrate and interconnect together. As a total system, the software will be initially tested for basic functionality to see if it can at least accomplish its intended purpose. Then, different, more extreme cases will be introduced in order to see if there are lingering holes in the code that could potentially damage the system.

The testing chart below details a summary of the different testing modules, their methods, and the prospective amount of time.

Testing Module	Location	Consists Of	Methods	Time Required
Hardware Testing	Texas Instrument Innovation Lab and Senior Design Lab	Hardware, PCB, Power board, MCU, Wifi module	Breadboard testing	1-2 sessions
Water Pump Testing	Senior Design Lab	Water Pump	Breadboard testing and functional testing	1 session
Moisture Sensor Testing	Senior Design Lab	Moisture Sensor	Breadboard testing and functional testing	1 session
Temperature Sensor Testing	Senior Design Lab	Temp Sensor	Breadboard testing and functional testing	1 session
Power System Testing	Texas Instrument Innovation Lab and Senior Design Lab	Power PCB	Breadboard testing	1-2 sessions
Software Testing Environment	n/a	Software sub-system	Module and whole system testing	throughout development and 1-2 sessions officially

Table 21: Testing Summary

Another factor to consider is the Wi-Fi module which needs to be tested into two parts. First, the communications between the MCU and the module need to be tested, and then the communications between the Wi-Fi module and the website

need to be tested. Setting up the MCU to WiFi module to network connection can be done with a testing board. A UART connection can be made between the board and wifi modules, and a few commands can be sent to the Wi-Fi module. Among these commands include one that requests access to a network, which requires the network ID and password. Once connected to the network, the next phase of testing can begin.

The IP address of the server needs to be known by the Wi-Fi module, which can be implemented through code. A TCP request can be made to the website, and if it is configured correctly, the website will allow the Wi-Fi module to start sending data. For basic testing, predetermined messages will be sent, but in the future the data transfer from the MCU to the Wi-Fi module will have to be constructed and tested during device integration.

7.0 Project Hardware and Software Design Details

In this section, there is discussion on the components comprising the design of the hardware and software. The software design section focuses on the functionality, algorithms, function and modules definitions, software architectures, web stack, database, hosting, website prototypes, and mobile application components. The hardware design section focuses on test results of different chosen hardware components.

7.1 Software Design

The software portion of the automatic irrigation system will be implemented on the ATmega microprocessor. This MCU will utilize the input data received from the sensors to produce a required data output using the code implementation. The code has three main functionalities: taking in moisture sensor input from the moisture sensor embedded in the soil to calculate moisture, triggering the water pump to pump water when directed, and taking in user input preferences based on the type of plant.

The diagram below shows a more detailed breakdown of the software design, consisting of user interface on the left and software control functionality on the right of the diagram.

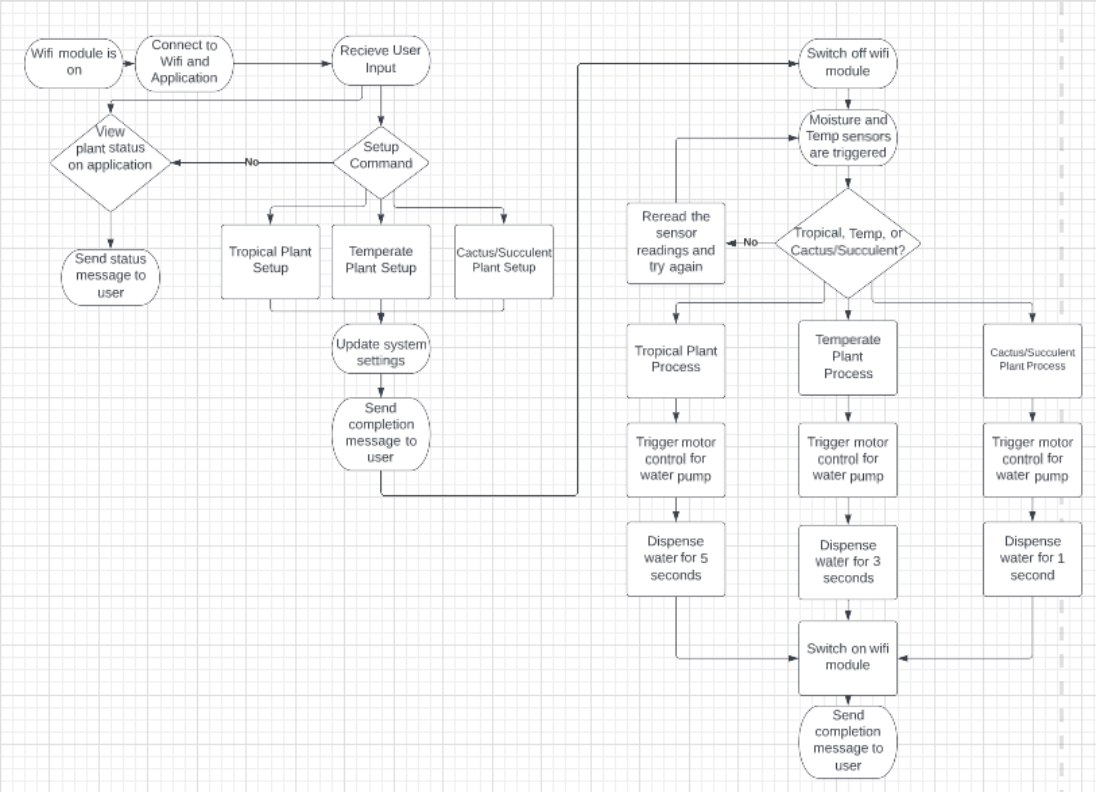


Figure 23: Software Design

7.1.1 Software Functionality

The functionality of the software is made up of a user interface required to display information to the user, controlling the water pump when directed, and any software needed for the soil moisture sensor. This can be accomplished through the functions present in the ATmega microprocessor. An IDE will be used to compile and transfer the code as well.

The diagram below shows the system integration of the two main software components. The main software components are the UI Software and the Control Software. The integration diagram shown below demonstrates how these separate components actually function. The actual connection between these two components will be achieved with wifi Internet connection. The control software is already present and uploaded to the microcontroller. The UI software exists on the mobile/web applications, server, and database.

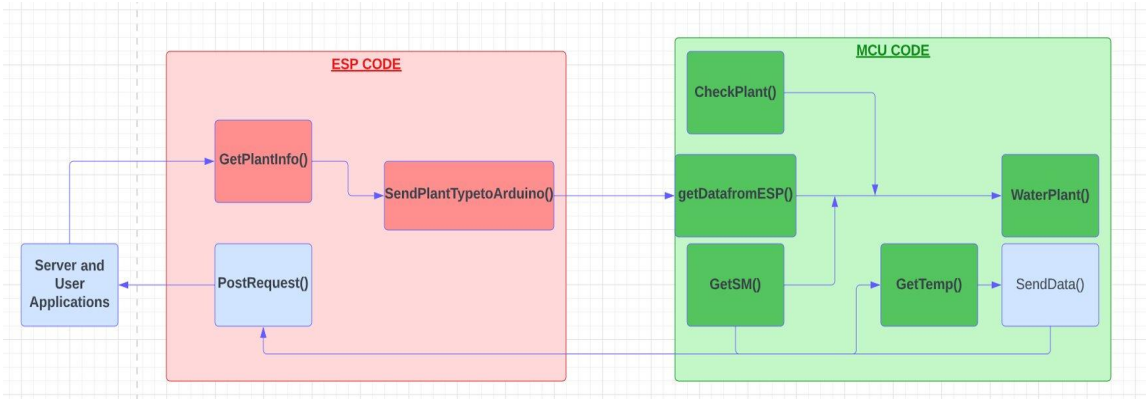


Figure 24: Software Controls

7.1.2 Algorithm Description

The implemented code will be using three different modules in order to implement the three main tasks required for functionality. all variables and constant integers are defined within each sub-module. At the beginning of the implemented code, the required libraries and headers will be initialized as per the requirements of the code. Also, appropriate comments will be utilized to break down and explain the function of the code and its procedure.

7.1.3 Moisture Sensor Module/Function

Purpose: This module has the purpose of reading the moisture level of the soil it is embedded in continuously.

- START

- Read in moisture based off a signal utilized by the actual moisture sensor embedded in the soil
- If there is a voltage reading detected by the moisture sensor
    - If the value is equal to 0, reset and continue. This is based off of a value set by the user base on their plant type moisture preference
    - Else, proceed
- END

7.1.4 Water Pump Control Module/Function

Purpose: This module has the purpose of turning on the motor of the water pump when prompted under certain conditions

- START
- Read in the moisture sensor value from the previous module
- If there is a moisture sensor value from the previous module
    - If the value is based off of a value set by the user base, then prompt the motor ON. Enable the required digital output pin to HI
    - Else, do not turn on the water pump control module

7.1.5 User Display Module

Purpose: This module has the purpose of taking in the user input. However, this is merely a rough approximation of the actual function of this code because this code is tied to the mobile application

- START
- Take in user input preference for plant type based off of “Tropical Plant”, “Temperate Plant”, “Cactus/Succulent Plant”
- Print the appropriate description for each plant type and confirmation message to determine if this is the user’s preference
- Allow the user to choose the type of plant and print a confirmation message
- END (this data is used in the previous two modules)

7.2.1 Potential Software Architectures

The structures, pictured below, show two possibilities for the overall structure of the software. Structure 1 as seen in Figure 25 shows the three types of plants

that could be inputted (Tropical, Temperate, and Cactus/Succulent), as three separate classes made possible by a “Plant” prototype class. The second potential structure as seen in Figure 26 shows a more stripped-down approach, with one “Plant” class with three methods for Tropical, Temperate, and Cactus/Succulent. The movement of the data in each approach is relatively the same, in that the user inputs user data which is interpreted by the programming to prompt the system to do work, as well as prompt the system to display status updates to the user’s Android phone.

We will utilize techniques such as tight cohesion/low coupling, the 23 software design patterns, clean code philosophy, and Top-Down development with incremental refinement to insure code optimization and cleanliness. Development will take place on a development board and IDE, to create an application interface to interface with the system using an Android phone. Decisions made by the system, based on the temperature sensor will be aided by a weather API. The system will be able to connect to the Internet to connect to the interface with the phone.

Structure 1

The diagram below depicts the initial prototype structure for the functional software.

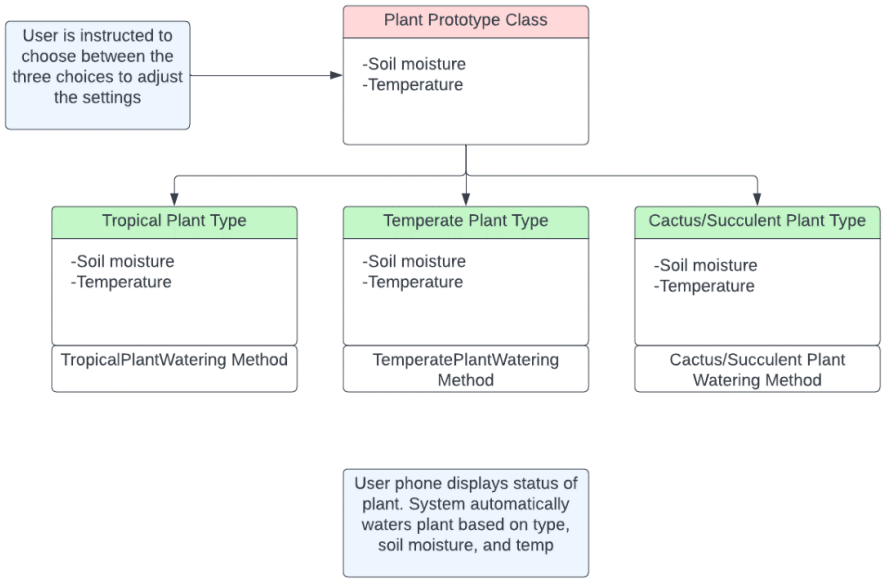


Figure 25: Structure 1

Structure 2

The diagram below depicts the secondary prototype structure for the functional software.

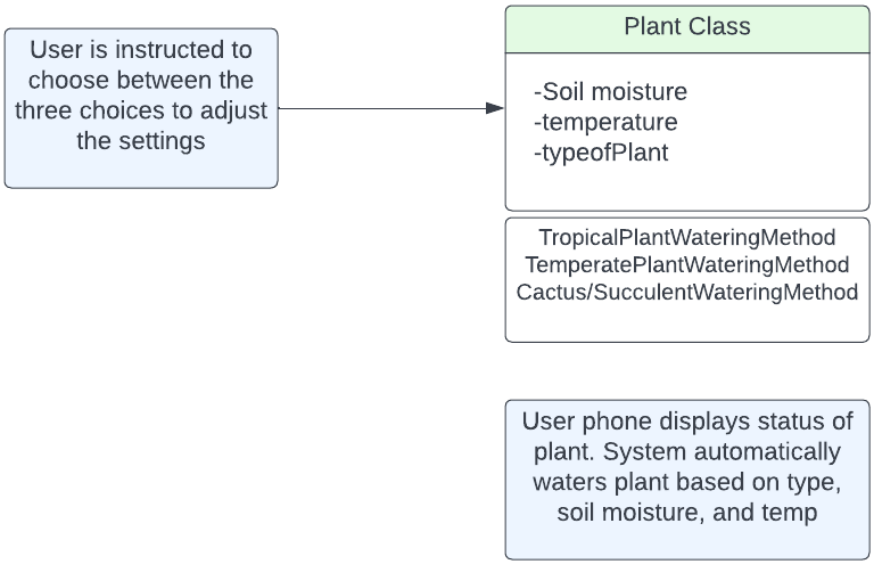


Figure 26: Structure 2

7.2.2 Web Stack

We decided to go with the MERN(MongoDB, Express, React.js, Node.js) stack. Instead of React we are going with Next.Js due to its optimization and speed. Next.Js Makes routing between pages way easier than vanilla React, making development that much easier. We chose the MERN stack over the LAMP stack for a few reasons. Our team is more familiar with javascript and the MERN stack is basically 100% javascript, while the LAMP stack can be a mix of a few languages, PHP being one of them that we are unfamiliar with. Node.js also requires little to no setup while Apache web server in the LAMP stack has some initial setup you need to go through to get it running properly. Table 22 shows some of the differences between the LAMP and the MERN web stack.

Differences between LAMP stack and MERN stack		
	MERN	**LAMP**
Development Time	Very little setup to get node.js server setup and working. Hot reloading also speeds up development time significantly.	Requires a bit more setup to get an apache web server setup and running properly
Cost	NoSQL databases are much more cost efficient when it comes to scaling, adding capacity horizontally is typically cheaper.	SQL databases are more expensive to scale up because they scale vertically.
Performance	Express.js allows for high-performing web applications and great user experience.	Performance can compare with MERN if Nginx is used, however some more setup is required.
Support and Documentation	MERN is a newer stack so less support and documentation, but in recent years this has improved significantly.	LAMP has been out longer and has a lot of community support and documentation available.

Table 22: Differences between LAMP stack and MERN stack

7.2.3 Database

We used MongoDB for our database. The main reasons we have chosen to go with MongoDB is ease of scaling, familiarity, cost effective, and flexible schemas. MongoDB easily scales horizontally to lessen the data load on one node onto multiple nodes or shards. Each shard can be seen as an individual database, and they can all cluster together as one logical database. Our team is also familiar with MongoDB so we will not have a learning curve of getting another database setup.

MongoDB is cost effective compared to other database systems. If we ever needed to scale up to hold more data, MongoDB can automatically scale when needed. Keeping costs at an absolute minimum. MongoDB also features flexible schemas so that data can be manipulated with ease. Data is also stored as BSON format, which is a binary-encoded serialization of JSON, allowing the data to be traversed, decoded, and encoded faster than plain JSON. BSON allows objects in our MongoDB cluster to have different sets of fields. This lets our objects be unique and only have the necessary fields that they need. For instance we may have two user objects in a users cluster. One user object may

have a middle name so they would have a middle name field for that specific object. The second user object may not have a middle name, so the middle name field will not be included.

A simple Entity Relationship Diagram of our database is shown in Figure 27. This ERD shows how our data will be organized and stored in our MongoDB database.

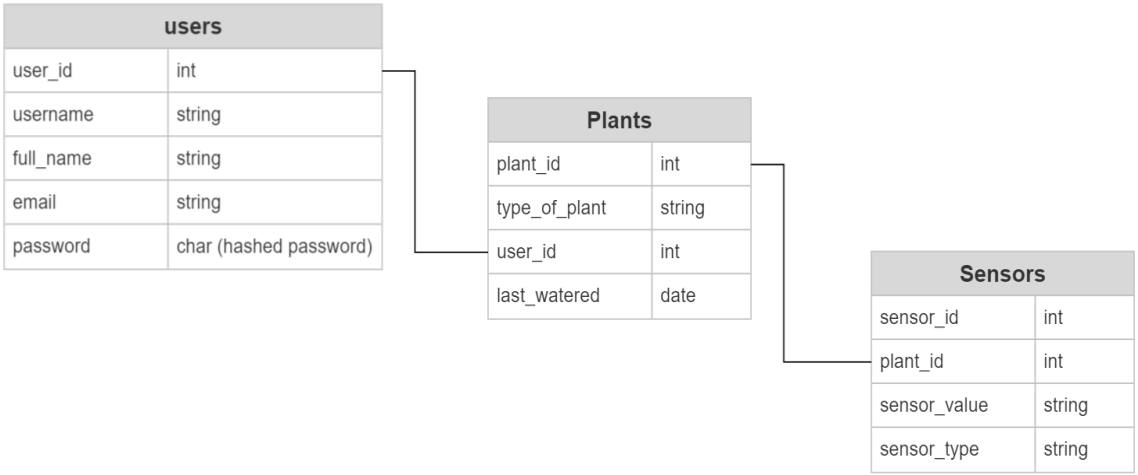


Figure 27: Entity Relationship Diagram

7.2.4 Hosting

Vercel was the service we used for hosting our website. We are using Next.js as our frontend framework which was created by the same team as Vercel, making Vercel the most compatible service to deploy our Application. Vercel features three pricing tiers, Hobby (free), Pro ($20 a month), and Enterprise(Contact Vercel for a quote). As of right now our application will run smoothly on the Hobby version, however if this project were ever to be public and on the market we would use either Pro or Enterprise to deploy our Application.

Vercel also features Integration which allows a database, code repository, monitoring, or even logging tools to be connected to our application with ease. A database such as MongoDB is easy to connect and utilize on our web application. Several Git repositories such as github can be configured to auto deploy our application when any changes to main occur in our codebase. Monitoring services such as Checkly can be used to ensure a working build is deployed, any faulty builds will be blocked from being deployed. Lastly, logging tools can be added to check Web vitals and insights for our website.

7.2.5 Website

In this section we will cover our Web Application and what framework we decide to use. We will also show some mockup pages for the website as a general idea

of how our pages will look. Website design is important to the user experience. When using a product, consumers look at all aspects of that product. Sometimes a well designed website is the first thing that a potential customer will look at. To use this first impression to our advantage, creating a welcoming and good looking landing page is the first step.

7.2.5.1 Website Mock Pages

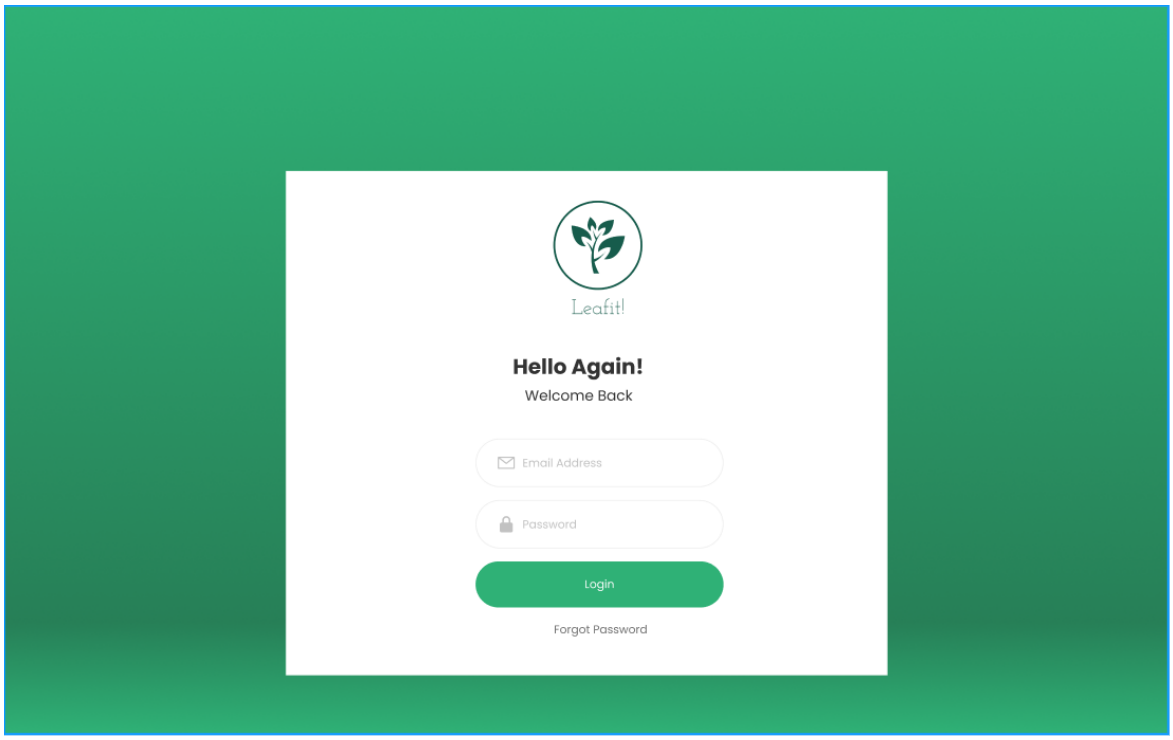


Figure 28: Website Mock Page

Our Website is one of two ways the user can interact with LeafIt. A user can register for an account and login to view their plants on either the website or mobile application. We are aiming for the website to be intuitive yet have a modern and simple design to make the User Interface as least intrusive as possible.

To separate ourselves from other similar products, we leaned heavily on the website and app side of LeafIt! This separation will provide us with novelty in this market unseen in other products. Given that “50% of consumers believe that website design is crucial to a business’s overall brand.” (Top Design Firms, 2021). The ability to store and keep track of your plants via website or phone app is something that many consumers will admire.

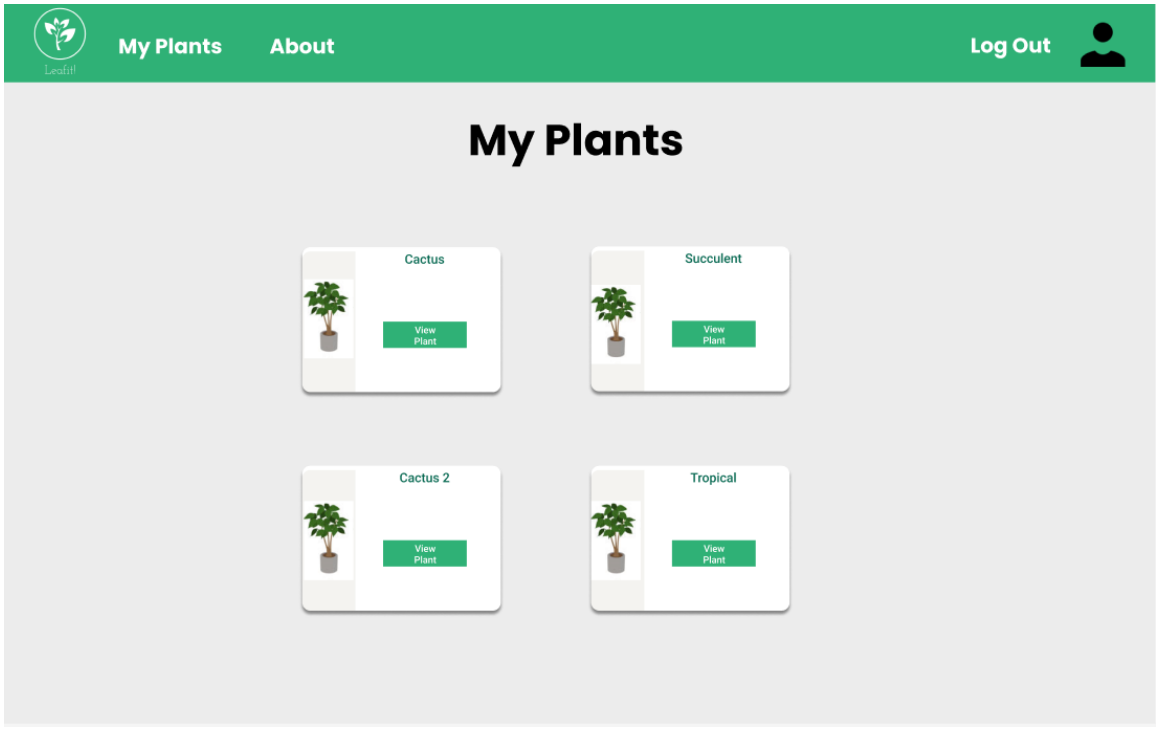


Figure 29: Website Mock Page “My Plants”

In the figure above, we see a user logged into their account, and the webpage shows them their plants. We see the different types of plants, as well as having more than one cactus plant.

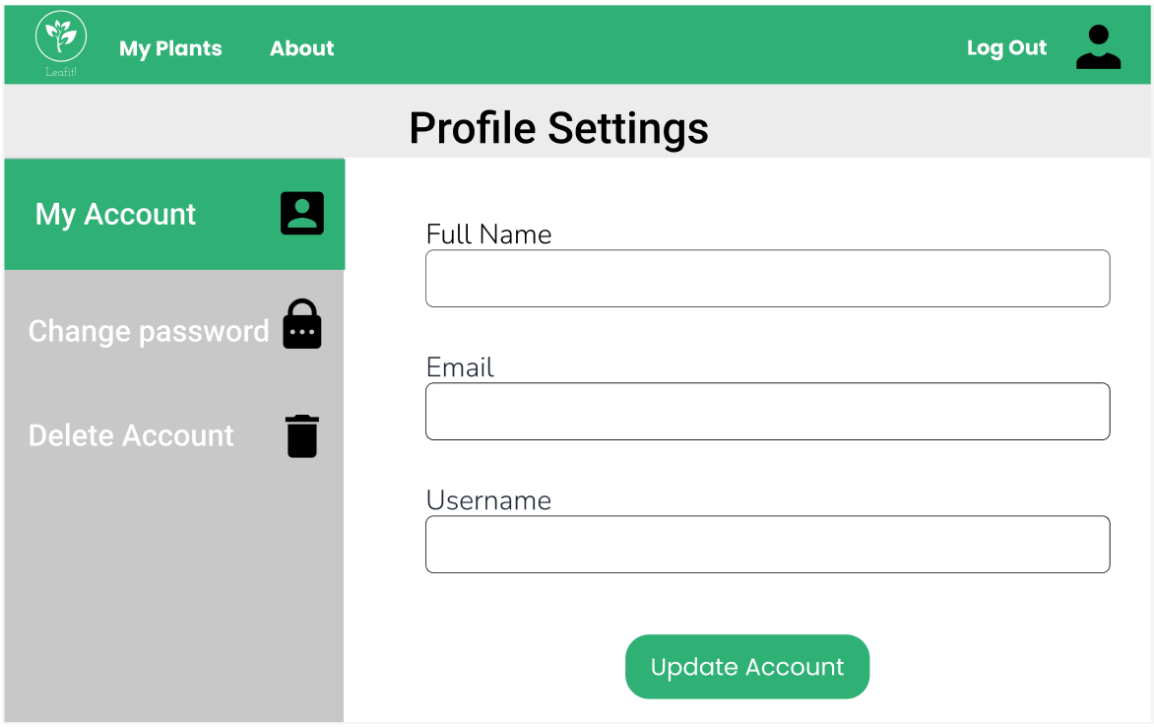


Figure 30: Website Mock Page “My Profile”

In the figure above, we see the account section of the webpage. This will allow users to change their name, email, and username and or adjust their password. Keeping the design simple is the goal, as overly complicated websites and applications will add to the complexity of our design and confusion for the user.

7.2.5.2 Next.js

Next.js is a react framework that is built on top of Node.js. Next.js features file-system routing which makes development a lot easier. Every component in the pages directory becomes a route automatically. In vanilla react you have to add it to a router file and do a bunch of extra steps. Another great feature of Next.js is image optimization which allows for improved performance and faster page loads for our website. The main reason we went with next.js is because we have previous experience with it and are comfortable using this framework.

Next.js is built on the React.js framework which brings a ton of features in it of itself. React features JSX, Components, and Simplicity. JSX allows for javascript code to co-exist with HTML code. This allows for conditional rendering and a lot of neat tricks that can be done with rendering HTML. React components allow for a lot of code reuse and cleans up the code tremendously. For example we can have a button component if we are going to be using the same style and functionality for each button on our website. Instead of making a new HTML

button each time and styling each one, we can make a React Button Component to reuse everywhere and save a lot of time and code while developing our application. Lastly, Reat has simple syntax that is easy to work with even if React.js is foreign to you. The intuitive syntax and code styles makes it a breeze to develop web applications.

Additionally, of all those features Next.js also excels in SEO (Search Engine Optimization) which is not a priority for this project at this time. However, if this product were to be available to consumers it is wonderful to have the option of SEO to draw more eyes onto our product. Next.js incorporated two ways to make SEO easy for websites by providing SSR(Server-Side Rendering) as well as SSG(Static Site Generation).

SSR(Server-Side Rendering) allows for the JavaScript code on our page to be rendered on the server rather than the client side. Therefore it is generated at runtime allowing it to reach search engines and users at the same time. Next.js makes it very easy to incorporate SSR into our application, abstracting a lot of issues that arose with SSR before Next.js such as on-demand content and load on the server itself.

SSG(Static Site Generation) on the other hand builds all the HTML on build time. With static sites we can compile the HTML on build time because the data is static. This allows certain static pages on our website to load very fast and is a great user experience with little to no wait time. The user-friendly application about Next.js is that you can pick and choose what individual pages have either SSR,SSG or neither. This allows a lot of flexibility with our application having both static and dynamically rendered pages. For example our “About” page will be static, that info will not change much unless we push an update to the text on that page. We may also have a dynamic page that will update when a plant is watered. This capability will allow us to customize the experience and adapt to situations as they arise, which will lead to a better user experience overall.

In the table below we summarize the key features and their descriptions to show why Next.js is a perfect framework for our web application.

Next.js Key Features and Descriptions	
Key Features	Description
Hot Reloading	Next.js quickly reloads the page when any code is changed and saved in the project directory.
Reusable code components	React and Next.js feature that allows for code reuse when using the same component in different areas.
Automatic Routing	Any files in the “pages” directory will be automatically routed with no configuration needed.
JSX	Allow HTML and Javascript to co-exist allowing for conditional rendering of our HTML.
Server-Side Rendering	Javascript is rendered on the server which makes web scraping easier for bots.
Static Site Generation	HTML is ready to serve as it is compiled during build time, great user experience.
Team has prior experience with React and Next.js	Having prior experience makes development much easier.

Table 23: Next.js Features and Descriptions

7.2.6 Mobile Application

This section will cover the mobile application design details and some mockup pictures for what the mobile application looks like. As development goes on we will update the UI to try and make the User's experience the best possible. This will come with some trial and error to see what works well and what buttons or components are necessary to have on each page.

Furthermore, we cover the mobile application framework we chose to go with due to the constraints of our project. Having a framework that allows for one code base for both android and IOS devices would be preferred due to the team

having a mix of devices for testing. Also having more compatibility never hurts, with not having to take care of a separate code base for both IOS and Android platforms.

7.2.6.1 Mockup pages

In this section we see some sample mockup pages that our mobile application will somewhat look like. The UI is bound to change a little, for the users experience sake to make our application as intuitive as possible. Application design is very important to making our product stand out compared to similar products. Ensuring that user's have a good experience using the app will bring them back for more. Being able to easily see each of your saved plants will give consumers peace of mind towards their houseplants, and will allow them to focus on their task at hand, be that vacation or traveling for work.

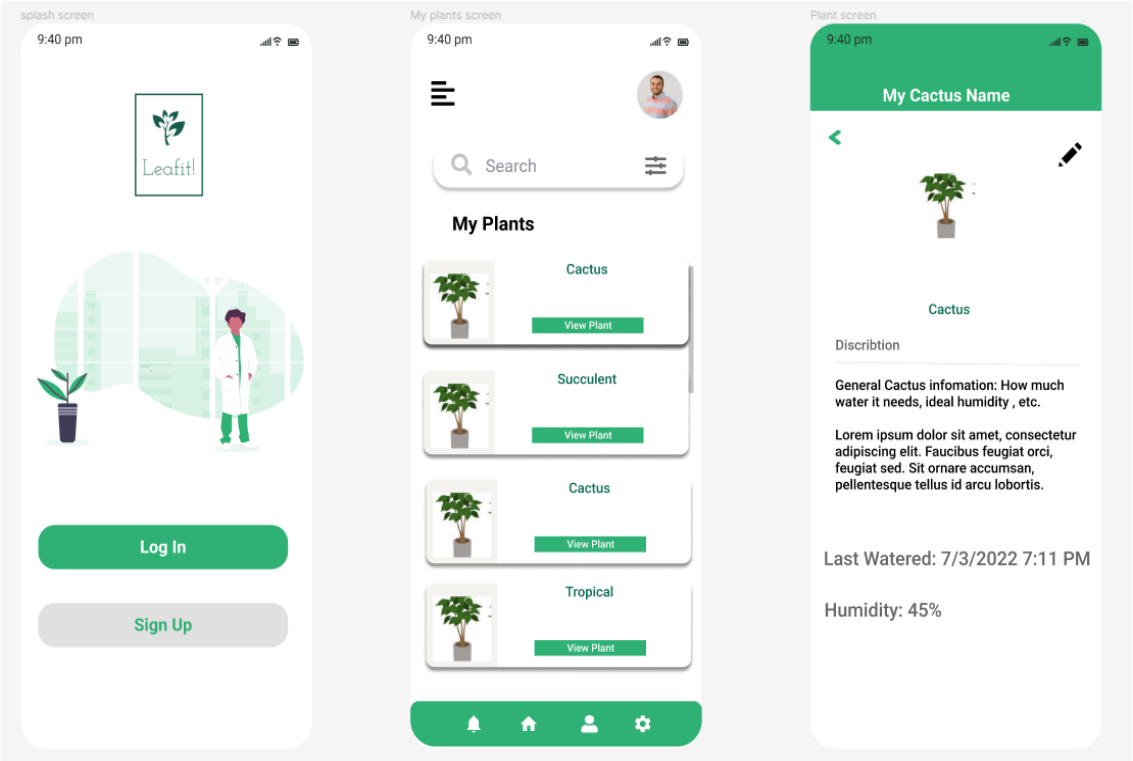


Figure 31: Application Mockup Page

7.2.6.2 React Native

We decided to go with React Native to build our Mobile Application. React Native is very similar to React, which will allow our codebase for both mobile and web design to be very similar. This will prove beneficial as any changes that may need to be done will be similar on both platforms or code bases. Due to our team having both IOS and Android devices, React native makes it easy to develop for both Systems using the same codebase.

React Native is very similar to React in that the syntax and coding style are very similar. The main difference between React Native and React is that React uses HTML such as <h1> or <p> tags, while React Native renders components such as <view> or <text>. All of the logic and data flow will be very similar, resulting in a small learning curve for the development of our mobile application.

React Native was a perfect choice for our project due to a multitude of reasons. React Native

7.2.7 Mobile and Website pages breakdown

Our mobile application and website have some of the same pages, while also having unique pages for just the website or just the mobile app. These pages with short descriptions are shown in Table 24.

Pages	Page Description
Landing Page	Exclusively for the desktop version. The first page the user sees when they visit our website's URL.
About Page	Informational page about our project.
Login	Users will have to provide a username/email and a password to be granted access.
Register	A new user can register for an account and provide a name, username/email, and password.
Plant Dashboard	Users can view all their plants that are being monitored/watered.
Individual Plant Page	Once a specific plant is clicked on the dashboard you can see in detail humidity levels, when the plant was last watered etc.
Settings	A user can edit different settings such as their name.

Table 24: Page Descriptions

7.3 Final Coding Plan

This final coding plan is intended to give an overall breakdown of the important sub-modules and subsystems that connect the hydroponics system together. These sub-modules and subsystems include the web and mobile applications, microcontroller, database, and sensors.

The flowchart depicted below is a visualization intended to show the flow of data from user input to the microcontroller and other subsystems.

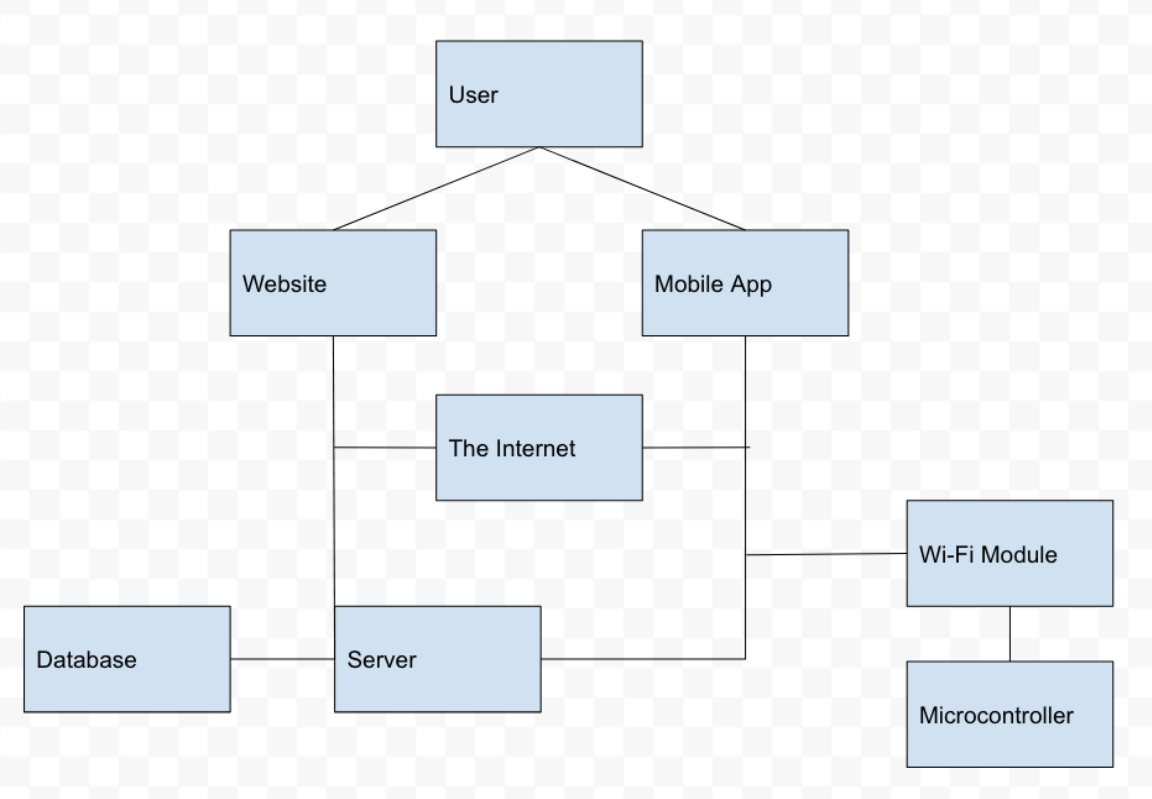


Figure 32: Network Layer

7.4 Hardware Design

This section covers the description of the hardware design for the automatic irrigation system. Each sub-section will either cover a particular module or layer of functionality for the hardware, prototypes, or parts selection. Hardware choices and design are important and significant because the physical interaction between each component will affect the overall performance and function of the automatic irrigation system.

7.4.1 Hardware Block Diagram

This figure shows the intended function of the project in relation to the separate hardware components comprising it. Vertical lines demonstrate a power connection. This figure is a simplified approach to problem solving.

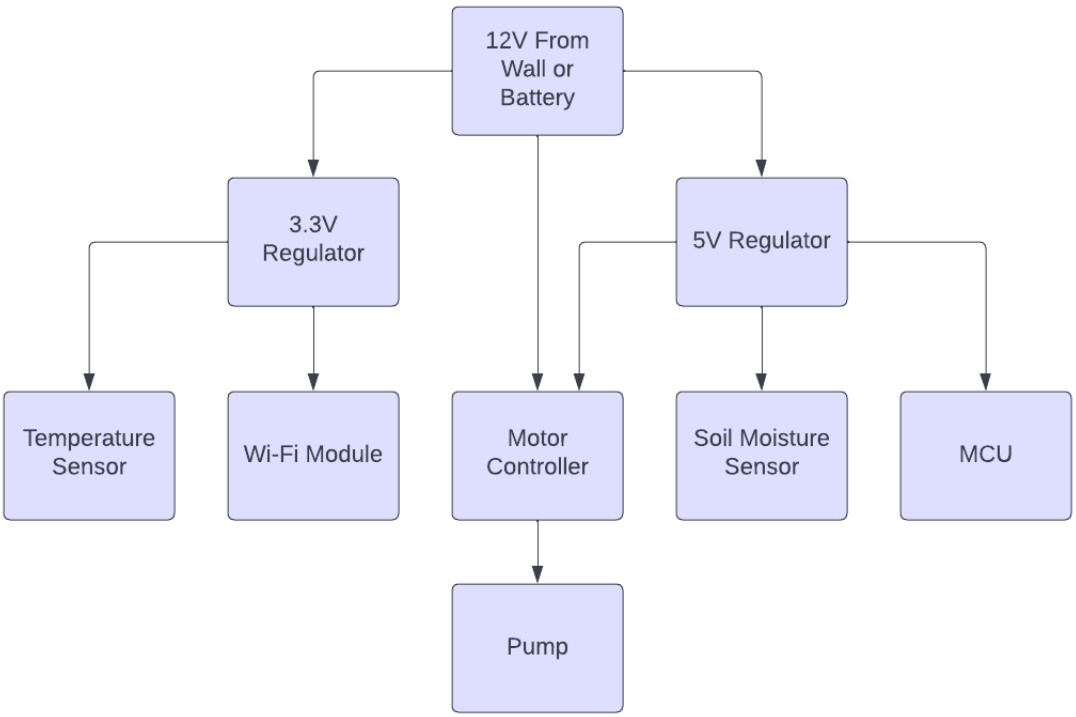


Figure 33: Hardware Block Diagram

7.4.2 Printed Circuit Board (PCB)

This section will focus on a discussion of the PCB design. This PCB will use the ATmega as the microcontroller. A wi-fi module will be attached as well as the necessary components needed for the sensors. Different types of pins are required for the design of this PCB. The microcontroller is connected to voltage and ground and has some I/O pins.

When the PCB design is ready, the gerber file is required using the design software. The main design software utilized is Eagle, which enables 2-layer design. This software design tool is the most useful to the team because this is the tool utilized by Junior Design class.

The first step when designing the PCB is to model all of the important circuitry involved. Then, use airtraces to connect the components together that are

important for the circuits. Then, it is important to detangle the wires in Eagle to make sure that the PCB operates correctly. This will put the wires in their optimal positions in order to have the best functioning PCB given the design. Then, it is important for the engineer to manually inspect the PCB design. The engineer is mainly looking for the distance between the traces and wires to make sure that short circuiting does not occur. After designing, printing the PCB may take several weeks. It is important to check different vendors in order to get good financial deals as well as get the PCB in a timely manner.

The table below shows a short investigation into PCB vendors.

Vendor	Minimum Order	Cost per PCB
allpcb	5	$5
jlcpcb	1	$2
pcbastore	1	$5
pcbgogo	5	$5
pcbway	1	$5
7pcb	3	$55

Table 25: PCB Vendors

It took four generations of PCB design to get a fully integrated circuit up and running.

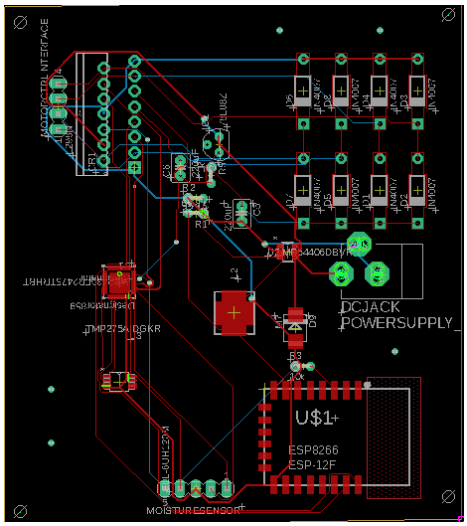
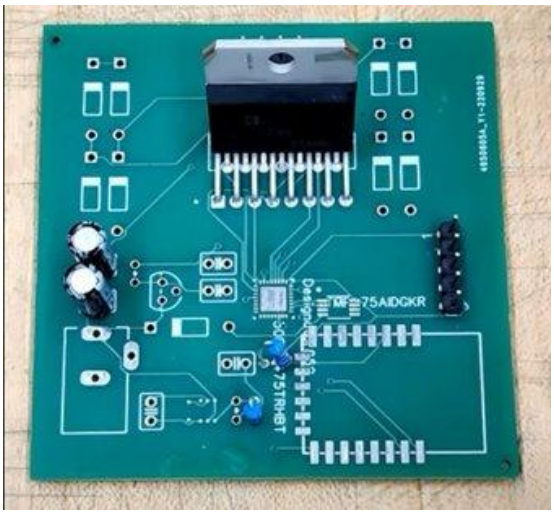
Schematic	Picture
	

Figure 34: First Generation PCB

Generation 1 was created with the MSP430 chip in mind . The software was developed with the Energia IDE. We were having issues with software development on the board. To save on development time, we decided to switch microarchitectures. There were also several issues with supply. The 3.3V regulator in this design was out of stock, and the manufacturer did not have the MSP430 chip, so we would have to solder it on ourselves. There were also a few issues with component libraries, which led to a several week delay. Before this PCB came in, The second generation was ordered.

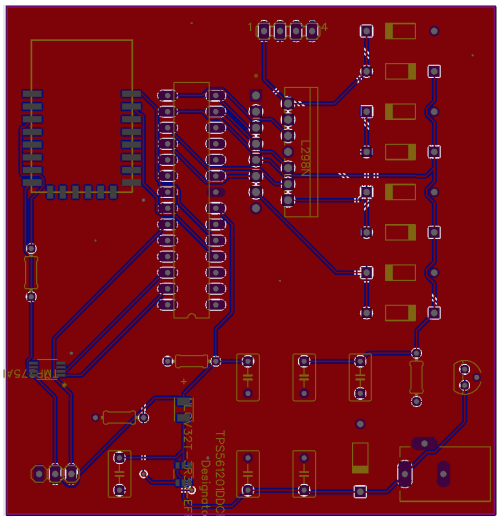
Schematic	Picture
	

Figure 35: Second Generation PCB

For the second generation, we changed the MCU, and the 3.3V regulator that was in stock. Unfortunately, this regulator was a surface mounted component that was too small to solder. There were also a few issues with the setup of the ATMEGA and ESP-12. The ATMEGA was missing a crystal oscillator and the ESP-12 was missing a trace for a GPIO pin.

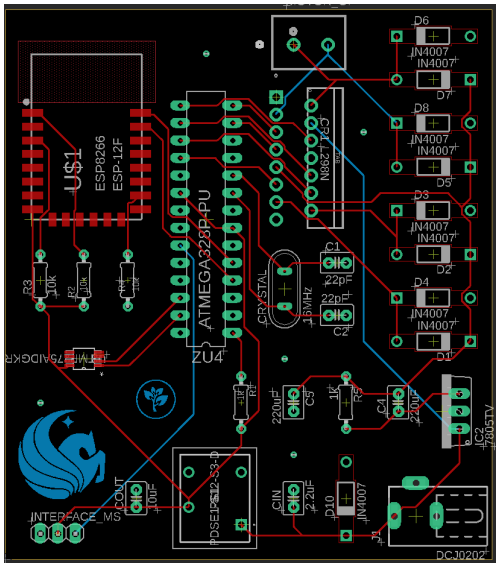
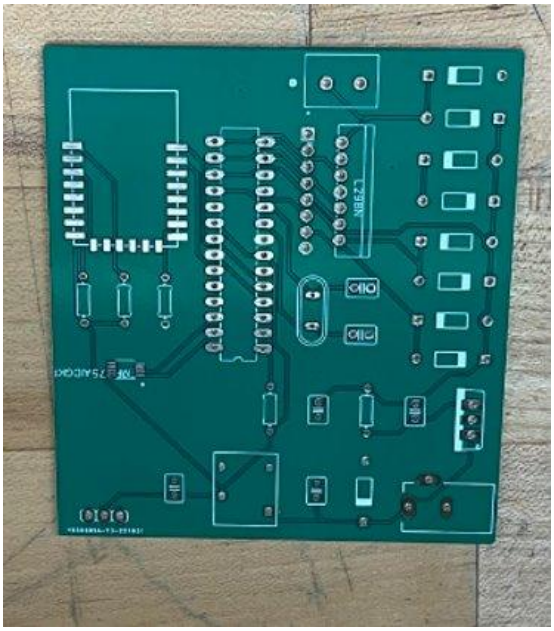
Schematic	Picture
	

Figure 36: Third Generation PCB

We were able to do extensive testing with our third generation PCB. We were able to identify a small issue with the 3.3V regulator, which required it to have the traces rotated. With this change, the power system is able to supply the correct voltages. The ESP-12 was able to operate correctly. We were also able to identify an integration error with the 5V to motor controller connection. These changes were all included with our fourth generation PCB.

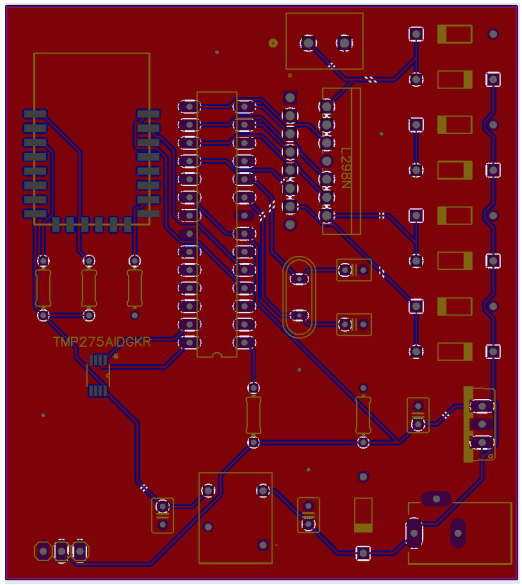
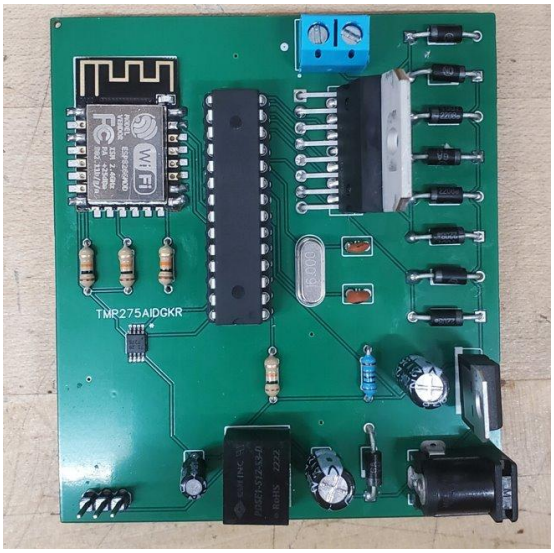
Schematic	Picture
	

Figure 37: Fourth Generation PCB

For software testing, the system was divided into different parts and then integrated into the hardware testing. During development, the system was divided into its “functional control” software and the user/application software. The control software was tested by running print statements connected to various functions that related to different hardware components. For example, when the function to trigger the water pump was triggered, the test prints out a statement to the console to ensure data transfer as well as actually turning on the water pump. The wifi capability subcomponent was unique in that it was tested by connecting the ESP-wifi chip and printing two statements between two different consoles to make sure that communication could occur over Wifi. The second component of the software, the user/application software, was tested by integrating it into the already functioning wifi capability test. However, this time, user information was entered through the application and either passed through the hardware to trigger the performance of a particular hardware component or gathered information either from the user or the sensor was verified to be stored in the database.

7.4.3 Part Selection: Power and Pump

To understand the power subsystem, testing and trial and error were necessary. This section goes into the testing used to compare the 3-5 Volt pump, and the 12 Volt pump. We also touch upon implementing the buck converter to allow use of both pumps at the same time.

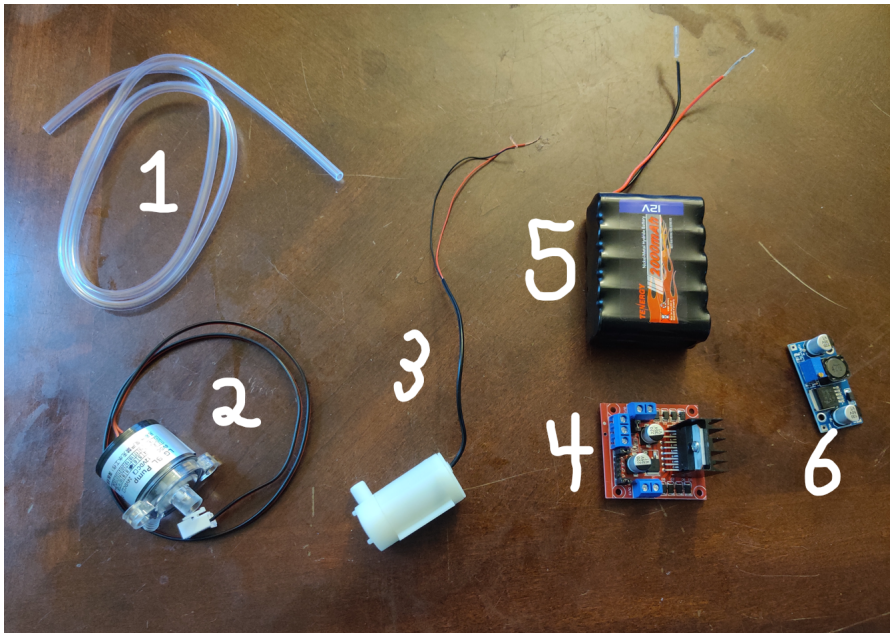


Figure 38: Picture of Parts

Power Parts List

Number	Part Name
1	Aquarium Tubing
2	12V Pump
3	3-5 Volt Pump
4	Motor Controller
5	12 Volt 2000 mAh Battery
6	L298N Buck Converter

Table 26: Power Parts List

3V-5V pump

The first set of tests were done with AA batteries (double A), and we wanted to test the various voltage levels of the pumps. We started with one double A battery, and allowed the pump to pump water from a bowl to a tupperware

container sitting on a food scale, in which we measured the ml (milli-Liters) of water the pump was able to move from the bowl to the tupperware in a 1 minute time span. This test was run several times at each voltage level, 1.5 volts, 3 volts, and 4.5 volts. To obtain these voltage levels we connected several double A batteries in series and attached the leads of the motor to the positive and negative ends. The table below shows our findings.

Voltage (V)	Time (m)	Water moved (mL)	battery configuration
1.5 volts	1 minute	40	red - black +
1.5 volts	1 minute	54	red + black -
1.5 volts	1 minute	48	red + black -
1.5 volts	1 minute	55	red - black +
3 volts	1 minute	406	red + black -
3 volts	1 minute	321	red - black +
3 volts	1 minute	368	red + black -
3 volts	1 minute	291	red + black -
4.5 volts	1 minute	332	red - black +
4.5 volts	1 minute	427	red - black +
4.5 volts	1 minute	750	red + black -
4.5 volts	1 minute	404	red + black -

Table 27: 3-5V Pump Testing

12V pump

We wanted to test a pump that had more voltage due to the possibility that we would be watering multiple plants spread out over a few feet of distance through tubing. To test this pump we used the Tenergy 12 volt 2000mAh battery we chose below, and connected the battery, once it was fully charged, to the breadboard, and attached jumper wires into the pump's connector. This configuration and testing is shown below in Table 28. To test the pump's output we followed a similar experiment as described above. The results of our experiment are shown in the table below. This pump only allowed power to flow through in one configuration, so we did not have to test using the different methods as seen above.

Voltage	Time	Water Moved (in mL)	Estimated water moved per minute (in L/m)
12V	30 seconds	721	1.442
12V	30 seconds	531	1.062
12V	30 seconds	815	1.630
12V	30 seconds	902	1.804

Table 28: 12V Pump Testing

Given that our testing concluded that the 12 volt pump was significantly more efficient, albeit more power hungry, we chose the 12 volt pump. Given that it could disperse the amount of water we will require to water the plant in a more efficient manner, thus requiring less time on power needs than the 3 volt and 4.5 volt options.

In the final week of implementing our project we determined the motor controller and the 12V pump were one big issue. The motor controller would provide a large inrush of current to power the 12V pump. This large inrush of current fried our crystal oscillator which is necessary for the ATMEGA chip and the temperature sensor to work. In switching from the 12V pump to a 5V Sipytoph pump we were able to rectify this problem. We then replaced the crystal oscillator and this was our final design on our PCB.

We have several options for how we want the pump to disperse the water necessary for the plant. Using the soil sensor we can use a simple approach in which when the soil sensor determines the moisture level of the plant's dirt is too low (less than 50% soil moisture) the MCU would send out half pre-designated amount of water such as 250ml. Alternatively we could use this percentage given by the soil sensor to send out either half or a quarter of the pre-designated amount of water. For instance if the soil sensor is reading 50% we could have the pump, through the pump controller, distribute 125ml of water rather than 250ml.

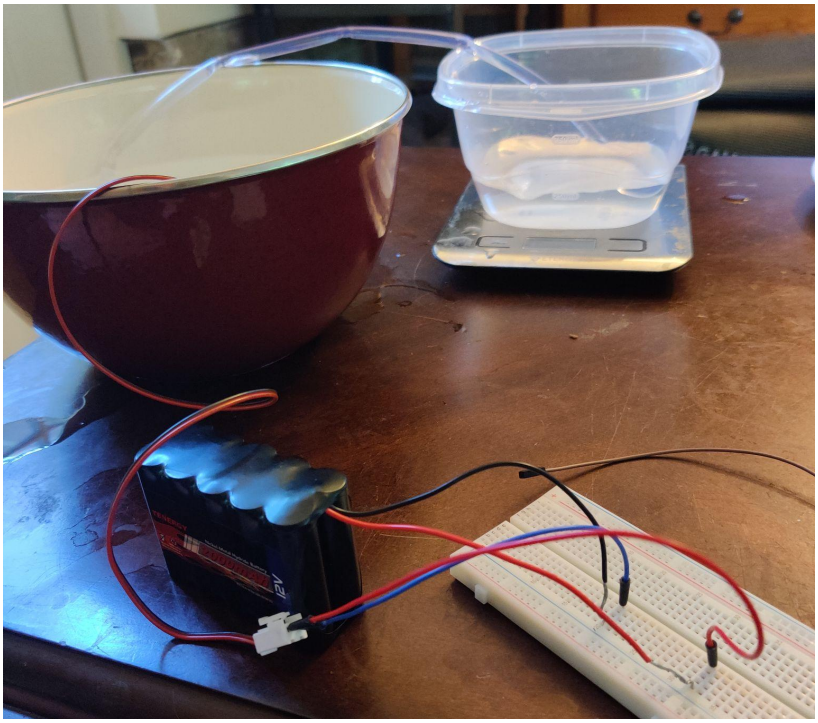


Figure 39: Pump Testing Configuration

7.4.4 Sensor Integration

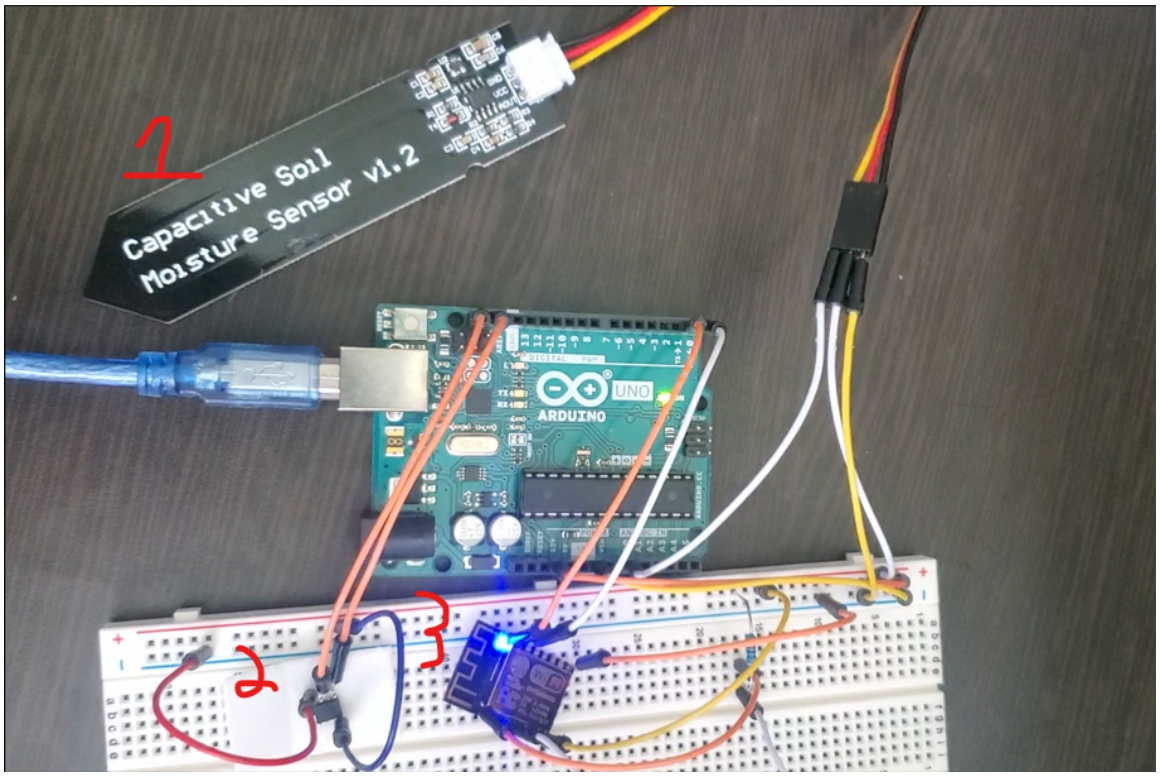


Figure 40: Temperature, soil, and Wi-Fi testing

Sensor Parts List

Number	Part
1	Soil Moisture Sensor
2	Temperature Sensor
3	WiFi Module

Table 29: Sensor Part List

This was the first attempt at sensor integration. The issue was identified to be connections of the temperature sensor. On the second attempt, all the sensors were able to output data.

7.4.5 Soil Moisture Sensor Testing

In order to determine the ideal way to implement our soil moisture sensor, we must first conduct controlled testing to understand how the sensor works without the added complexity of the whole system.

Figure 41: Soil Moisture Sensor Testing

First, the soil moisture sensor was connected to the arduino's analog input pin, 3.3V power pin, and the ground pin. Testing code was used to return the analog output of the sensor. Since unitless values are returned from the analog output, the moisture sensor will be used as a percentage. This means that the ambient air will be measured as 0%, and being fully submerged in water will be 100%. To calibrate for the air, the analog input of 586 was set as 0%, and for the total submerged, the analog value of 305 was set as 100%. To test, a potted plant as seen in Figure 38 will be used, and 1 tsp or 4.93mL of water will be added at each step. The values will be put into a table and graphed below:

# of tsp	analog value	%
Ambient Air	586	0
Ambient Soil	519	24
1 tsp	509	27
2 tsp	499	31
3 tsp	479	38
4 tsp	415	61
5 tsp	392	70
6 tsp	384	73
7 tsp	370	77
8 tsp	360	81
9 tsp	351	85
10 tsp	347	86
11 tsp	344	87
submerged in water	305	100

Table 30: Moisture Sensor Data

This was done in about 738 mm^3 worth of soil. This soil had a plant in it, and it was bought in a store in south Florida. This is important to note as different soils will have different consistencies. These changes will affect the curve of water saturation, as different soils will be able to hold different amounts of moisture (More clay= less water, more sand= more water)

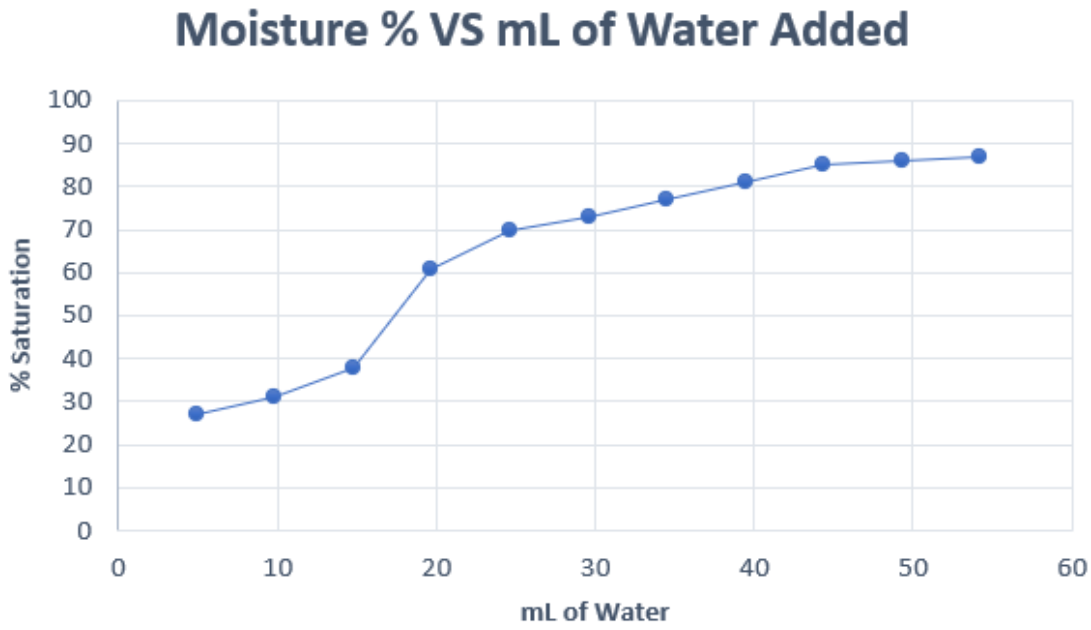


Figure 42: % moisture vs mL of water added

From Figure 39, it is clear that the analog output of the soil moisture sensor is non-linear. The known values are the outputs of when the SMS is out of the soil, when it is submerged in water, and the values of the SMS when water is added to the soil. The initial amount of water in the soil is unknown, but can be estimated to be about 5mL of water, or 20% saturation. This makes sense as this soil was outside and rain has been quite frequent. It has been estimated that most plants like to have a soil saturation of around 20%-60%, in which this sensor is the most sensitive. Once the soil reaches about 70% saturation, a reduction in the increase in saturation becomes apparent, and by 50mL of water added, excess water starts to leak out. For a healthy plant, the saturation levels should be kept no lower than 20%, and no higher than 60%. These bounds should not be exceeded to prevent the plant from drowning/soil degradation or from drying out. This means that the software should recognize the analog input values of 519-415 as the bounds to keep the water level stable. More specific values per plant type can be determined through more testing or can be a decision that the user of the software can make when observing their plants.

7.4.7 Wi-Fi Module Testing

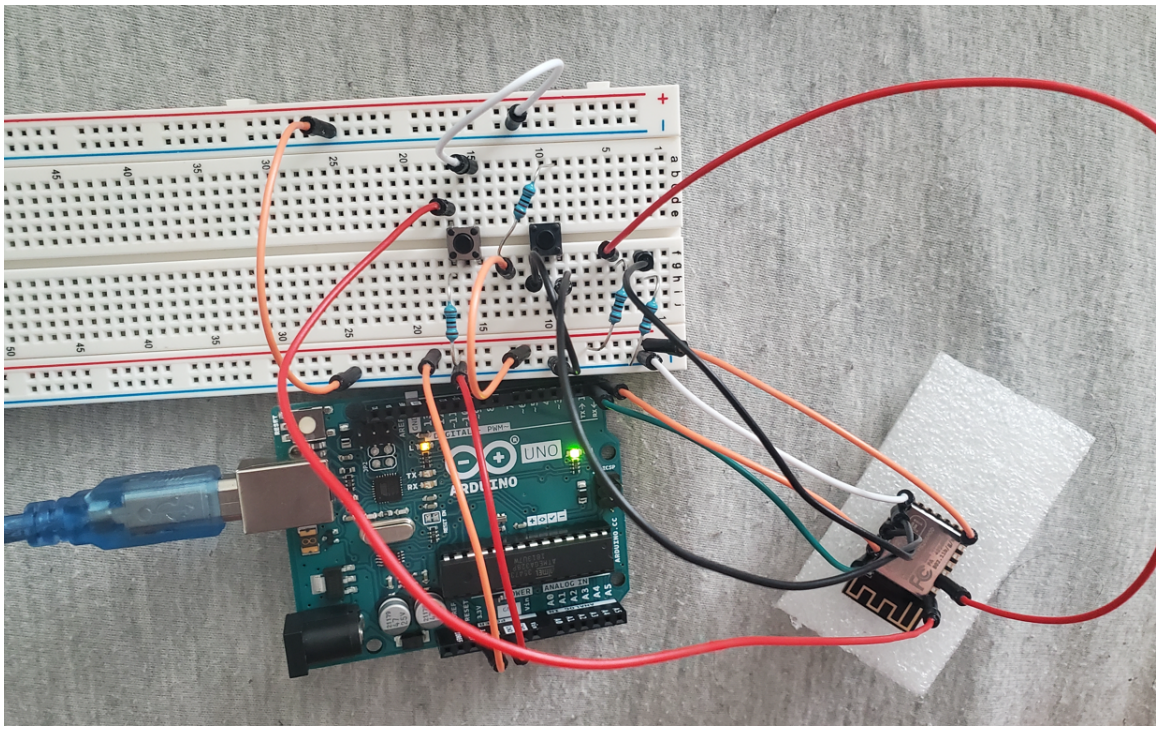


Figure 43:Wi-Fi Module Testing

The ESP-12F was tested using basic circuitry and an arduino. The wiring was based off of an online tutorial which can be found in the appendix. This circuitry was created to make resetting and flashing the module easy. The module was supplied with 3.3V from the arduino. It is important to note that the arduino cannot supply enough current for the module to operate at max efficiency, in this test the Wi-Fi module will not be used to transmit any information to any web server. The RX and TX of the module and the arduino were connected to enable communications. The reset and flash pins of the Wi-Fi module were connected to pins for easy resetting and flashing. The baud rate of the arduino was set to 115200, which is what the Wi-Fi module communicates without changing any of its settings. The arduino and Wi-Fi module are communicating using SPI. The communications between the Arduino and the Wi-Fi module will be done through the serial monitor, which does not require uploading any code. Several commands were used to ensure that the module worked and to record several parameters of the Wi-Fi module. These were all typed out through the use of the serial monitor, which does not require transmission code to be written. The information gathered includes the MAC address, IP address, and other system settings such as firmware version and Wi-Fi mode. Open networks can also be searched for.

Command	Result
AT	Checks to see proper working conditions, responds “OK”
AT+CMODE=1	Sets the Wi-Fi mode to station mode, can now connect to access points
AT+CWJAP="SSID","PASSWORD"	Connects to an access point with it’s network name and password
AT+CIFSR	Returns the IP address and MAC address

Table 31: Wi-Fi Module Commands

The IP address was assigned to be “192.168.191.35” by the network and the MAC address was discovered to be “58:bf:25:da:61:ac”. Every computer has a mac address, which can be used for certain communications in certain networks. The fact that the network assigned the Wi-Fi module an IP address means that it was connected to a network. Since the network recognises the Wi-Fi module, that means the module will be able to reach out to servers. This proves that the hardware works and can send TCP requests to web pages. The connection to the network is shown in Figure 41 from both perspectives. The top picture is from the perspective of the network, and the bottom picture is the perspective of the Wi-Fi module. This will allow for further testing of communication between the Wi-Fi module and the website, which can be implemented with code and further integrated with an API. This will fully establish a link between the PCB system and the website which is vital for the communications with the phone interface and data storage.

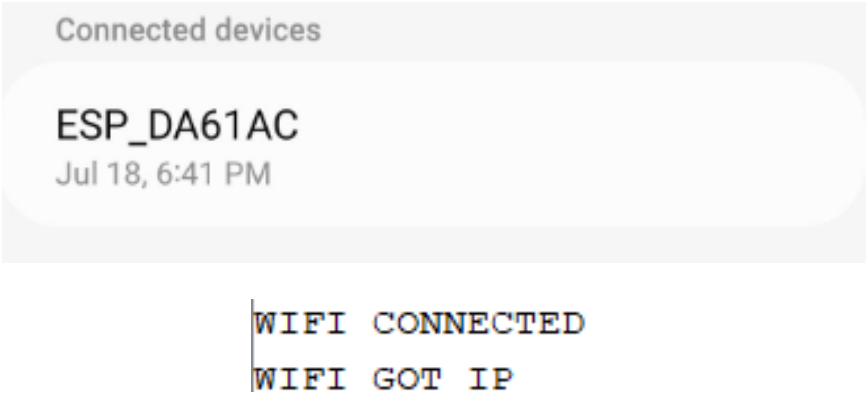


Figure 44: Proof of Connection

7.4.8 Temperature Sensor Testing

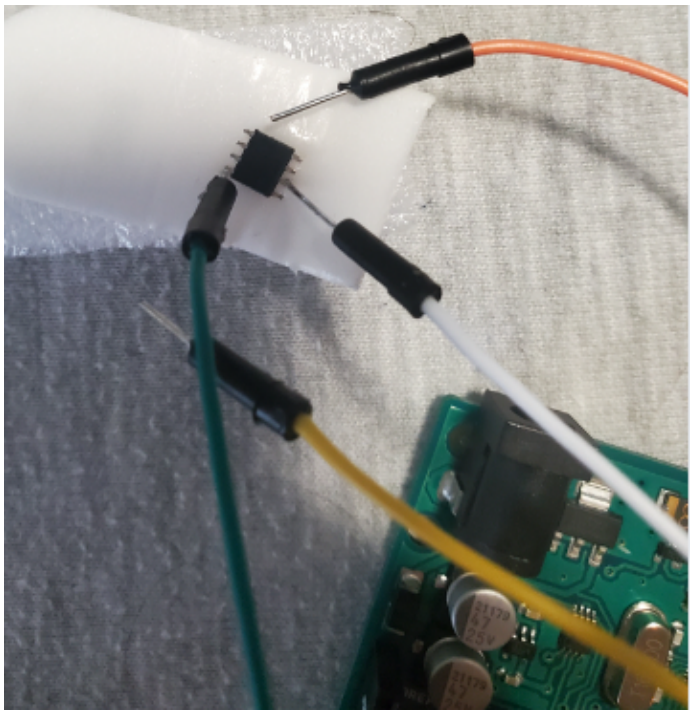


Figure 45: First round of temperature sensor testing

The TMP275 is a digital sensor that uses I2C to relay sensor values. An arduino was used with the LM75 library to read the outputs of the sensor. As seen in Figure 42, the temperature sensor is too small for a breadboard. A piece of sticky material was used to hold the sensor in place, and the wires that connected into the arduino were pierced through the material. The wires had to be carefully placed, and the sensor had to be head down by another wire. The connections included a ground wire, a 3.3V power line, an SDL line, and a SDA line. When all the connections were correctly implemented, the arduino was successfully able to receive values from the sensor:

```
Temperature = 27.00 C
Temperature = 26.50 C
Temperature = 26.50 C
```

Figure 46: Temperature sensor values read

This comes out to around 80F, which is about the temperature of the room. In order for more thorough testing to be done, a better way of securing the sensor's connections will have to be implemented. A suggestion would be to solder connections to the temperature sensor which will bypass the foam all together. This method worked in the second test, and the heat produced from the fingers of the tester did not affect the temperature sensor.

8.0 Design Integration

In this section, there is discussion on the design integration of the automatic irrigation system project. Design integration proceeds to show an overall system overview of this system. The sections present are controls and power integration, data storage, system casing, and heat dissipation.

8.1 Controls and Power Integration

The MCU, sensors, the power system, the Wi-Fi module and the web server will need to transfer information and make requests with each other. The MCU will operate at 5V. First to be considered is the communications between the MCU and the sensors.

The light sensor and the temperature sensor will use I2C for communications and will operate at 3.3V. With proper setup, these sensors will send digital readings to the MCU with two information lines, SDA and SCL. SDA will need to be able to be pulled up to Vcc, which is 3.3V in this case. The soil moisture sensor will operate at 3.3V and will send an analog signal to the MCU's ADC. The soil moisture sensor outputs a voltage of 0V-3V, and the 12 bit ADC on the MCU has a 0V-5V input voltage. This means the analog range will not be fully mapped to the 10-bit ADC, and will lose sensitivity. Also, data loss will occur due to noise and quantization error information loss. Further amplification of the input signal would fix the issue for the soil moisture sensor, which is currently the only analog sensor.

The Wi-Fi module will operate at 3.3V. The Wi-Fi module to MCU data connection will be through UART. This means that digital data will be transferred between the MCU and Wi-Fi module. The Wi-Fi module will use protocol b, g, or n to send TCP requests to the server. The system will be able to send live information to a server, and the user can access this information whenever and wherever they are through the phone interface, which will connect to the server through a network. The watering system would send information to the server through Wi-Fi every 15 seconds, and the user can check at any time and send requests to the server, which will communicate with the PCB system. Further experimentation will be required to bypass the PCB to server to phone interface connection and just have the PCB and phone interface communicate. The Wi-Fi protocol does support this measure; however, the experimentation will be done to enable a TCP request to the embedded system to turn on the rest of its functionality. This means of communication is to prevent the PCB system from becoming inaccessible if the server shuts down.

The pump controller will have digital inputs to choose which pump to run (up to two pumps can be implemented). The controller takes an analog input to determine the power consumption of the pump. This can be a static value in the case of this system.

8.2 Data Storage

There are three places that data can be stored in this system. The first place would be on the PCB in a memory module, the second place would be the phone interface, and the third place would be the web server. Although each of these pieces should have access to some memory, there are a few aspects to consider when choosing where to store the data long term. The advantage of storing it on the PCB would be the lack of dependance of the internet. For example, say if the internet goes out, if there was no memory module for storing data, the information gathered would be lost. The locality of the data is a trade off: on one hand it does not require paying a monthly fee to have a server store the data, but on the other hand there would be no backup. If the PCB gets fried, all of the data would be lost. A disadvantage would be solving the data analytics question, as analyzing the data on the PCB would have power drawbacks. A compromise to balance out the pros and cons would be to have enough memory on the PCB to store enough information to prevent a lapse in the internet resulting in loss of data. By looking at the digital sensors of section 3, a good assumption would be that a total of 4 sensors would be used and each of them output 12 bits. This will make for 48 bits, or 6 Bytes. The MCU will be set to collect data every 15 minutes, which equates to 96 times a day. This will result in about 576 Bytes of data that will be collected in a day. A 4 kilobyte memory module will be able to hold a little more than 7 days worth of data. This amount of storage on the PCB for data should suffice for emergencies. In the end, we decided to have the PCB system only remember the last set of values, so storage was not an issue.

Next, the storing of data was considered on the phone application. On the cell phone, power consumption is less of an issue than on the PCB. It also contains the computational power to do data analysis. The PCB part of the system and the phone interface could become independent of the web server using the ad hoc feature of Wi-Fi. These reasons make it a better long term storage than the PCB, but there are still some drawbacks. The application would have to be running in order to execute commands and to request data. This would mean the alert system would have to be located on the PCB to warn the user about certain conditions. These communications between the phone interface and the PCB would be unpredictable due to the runtime of both. The third option of storing the data long term is to store the data on a server. Data servers have very high uptime, so the PCB system and the phone interface would be able to access it any time. The calculations and storage can be done on the server, which will have the most real time data of the three methods discussed. It also minimizes the computations the PCB system will need to use, which will decrease the amount of power it needs to use. The phone interface can also be simpler and more compact; taking up less of the phone's storage. This would also make it easier to update the commands to the PCB. The request to change settings would go from the phone to the server, and the next time the PCB system initiates a TCP request, the server can send the command update during that communication. The system would at most have to wait 15 minutes for an update, but the scale of the project, the time to update does not pose any

significant threat to the health of the plants. In conclusion, the best way to integrate the data storage of these systems would be to have long term storage on a webserver and to have at least 4 KiloBytes of storage. This was achieved with a database.

8.3 System Casing

This system has a casing to protect the electrical components from physical abrasions and potential water contact. A few constraints will have to be considered first before choosing the materials for the casing. First is the placement of the sensors. If a light, temperature, and moisture sensor are implemented, the placement of these sensors will affect casing design. The moisture sensor needs to be in the soil to operate, so this sensor will not be in the casing. This means that there will have to be an opening in the case that allows the moisture sensor wire to still be connected to the PCB, which will reduce the waterproofing. The temperature and the light sensor will most likely be implemented on the PCB. This means that both sensors will be measuring the microclimate of inside the casing and a few adjustments will have to be made to ensure that the climate inside and outside the casing is equivalent. At least the top of the casing will need to be clear to reduce the amount of light distortion. The case will also have to be breathable to allow the temperature inside the case and outside the case to equalize. With these constraints in mind,

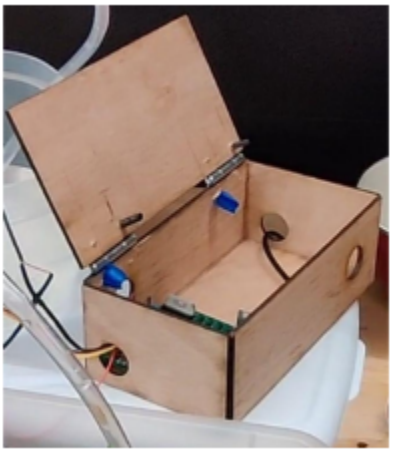


Figure 47: Final Container

The casing materials we considered would be 3D printed plastic, laser cut acrylic, or laser cut wood as seen in Figure 44. A 3D printed casing will allow for total control of design and cheap printing cost, but it is also not waterproof and is not translucent. This would have to be fixed by treating the case with some waterproofing material The second option would be to use laser cut acrylic, which would only need to be waterproofed at the seams and has translucent sheets. The disadvantage is that the casing will have to be designed with the limitation that acrylic comes in sheets, which will limit cutting to basic shapes. This does have the added benefit of being able to make clean side panels as seen in Figure

43, which is much harder to do while 3D printing. The advantage of using wood would be the cheap material costs, but the issue with wood is that it would be damaged by long term contact with water.

The battery will have to be accounted for as it makes sense to allow for the PCB and battery to fit in the same water resistant container to allow for overall implementation simplicity.

8.4 Heat dissipation

Heat dissipation is a key issue with electronics. “According to Newton’s law of cooling, the heat dissipation rate is proportional to the temperature difference between the body (electronic device) and the surroundings.” (Heat Dissipation in Electronic Devices, Cadence System Analysis) The parts of this project that will consume power need to have their heat dissipation considered. The main parts include the power system, sensors, MCU, and Wi-Fi module.

The MCU’s power draw will come from its high frequency oscillators which are required for I2C and SPI communications as well as timing 15 seconds between sending data to the server in a non testing environment. The current draw will increase 135uA/Mhz. These communications do not have to be particularly fast or accurate, so the RC clock (which uses a lot less power) can be used for the 15 minute timer while the more powerful oscillator can be used for communications, which should last less than a minute. Generally, the MCU will not need to operate continuously, and will not have to use a lot of computing power to solve complicated issues. The maximum operating temperature of the MCU is 105C, which is well above any ambient temperatures. If some features remain unused or are designed around, low power modes can be utilized for the entire runtime or switched to when necessary.

The sensors ended up drawing less than 100uA. Since they only operate for a short time and with so little power, the heat produced by them should dissipate naturally without any additional heat sinks. The sensors chosen all operate under ambient temperatures, so a really hot day will not cause acute damage to the sensors.

The Wi-Fi module that was selected for heat analysis was the ESP-12F. This module will operate at a voltage of 3.3V and depending on the Wi-Fi protocol, will draw different amounts of current. At most, to transmit with protocol 802.11b, will typically take 170mA of current, and at the least to transmit with protocol 802.11n will typically take 120mA of current. This means at most this module will consume 0.56W of power at most. The use of this module needs to be considered. This module will communicate with the server every 15 minutes, so according to the data sheet, most of the time, the module should be in “Deep-Sleep”, which typically draws 1uA. The module will only transmit for maybe 0-3 seconds since the system will typically only transfer a few bytes during the interval. Also, the operating temperature of this module is 125C, which is way above ambient temperatures, so the environmental temperatures should

not be an issue. With these considerations in mind, the Wi-Fi module will not need any additional heat dissipation.

The wires connecting mobile components should have the appropriate AWG to prevent overheating from the cable being too small. If the AWG is up to standards, then the wires should not overheat. In the event of a short circuit, unwanted current can find its way through the circuit and start heating up components.

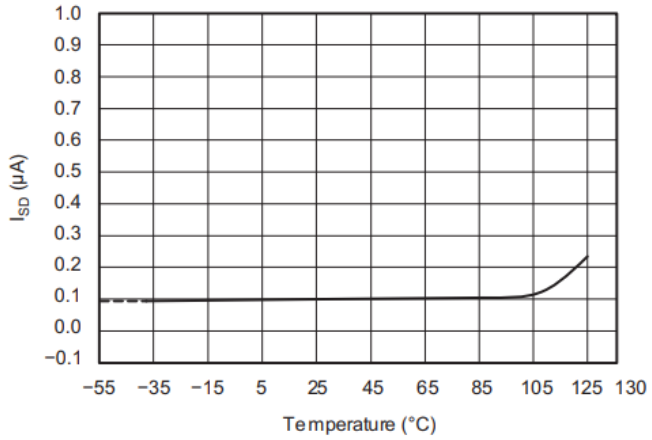


Figure 48: Shutdown current VS Temperature for the TMP275

Some of the components, like the TMP275 as seen in Figure 45 will actually shut off when approaching their maximum operating temperature. For this component, the damage caused by the increase in temperature will be minimized by these safety features, but the PCB connections have little to no protection from heat caused by a short circuit. The addition of small current limiting fuses will limit damage from too much current to just the fuses. This will not only prevent damaging short circuits, but also from overheating due to a malfunction.

8.5 Electrical Device Protection

There was an issue with connecting the sensor system with the power system. All of the sensors and the Wi-Fi module were tested with power from the Arduino Uno's 3.3V power pin. This was able to provide just enough current for the system to work. Once this system was connected to the power system, which consisted of the 12V battery and a buck converter that converted the 12V to 3.3V, the connections to and internal components of the Wi-Fi module were fried. These were traced back to the Vcc+ of the module. The Buck converter steadily read 3.3V, so the issue was most likely caused by excess current due to improper grounding. To prevent vital parts of the system from being fried, protective devices will be used on the PCB to prevent damage. The devices that are needed to be protected are DC and low voltage. These include the Wi-Fi module and the MCU. The Wi-Fi module at the most will operate at no more than 500mA and the MCU will at most operate at 2mA. If required, that means a 250V 500mA fuse can be implemented for the Wi-Fi module and a 125V 2mA fuse can

be implemented for the MCU. These are usually expensive and should be used as a last resort. Most fuses are more expensive than the components that need to be replaced, so financially fuses do not make much sense. Another good way to protect devices would be to ensure proper grounding, which should be implemented in the PCB design phase. Spare parts are required for testing and making sure there are back ups when faults do happen.

8.6 Project Operation

This section will give users an in-depth look into their autonomous plant watering system. From describing ideal water pump placement, tube placement, and software layout the consumers will have everything they need to operate LeafIt!

8.6.1 Introduction

This automatic irrigation system should contain all of the necessary apparatus to water various amounts of indoor and outdoor plants. No programming, electrical work such as soldering, or construction is required for the function of this automatic irrigation system. The only preparation required for use of this automatic irrigation system is adjusting the length of the vinyl tubing to the desired length of the user as well as downloading and operating the required application for control of this system through their Android phone. This automatic irrigation system contains 15 meters of vinyl tubing, a power and control box containing electronics necessary for the function of this system, moisture and temperature sensors necessary to gather data about the health of the user's plants, and a water pump to submerge into an independent source of water for the user's plants.

Note that the automatic irrigation system will not function properly unless the water pump is submerged in an adequate amount of water; most failures in the system will be due to inadequate amounts of water from the water source. Failures in the system are most likely due to low power or compromised power electronic components or input lag due to a faulty Internet connection slowing down the transfer of control signals between the user's Android phone and the automatic irrigation system. Less likely but more catastrophic failures in the system will mostly likely be due to moisture damage in the compromised power and control box or damaged sensors and water pump.

8.6.2 Setup/Placeholder figures of pictures depicting proper setup

1. Fill a container with water that can adequately submerge the system's water pump. The wire connections between the power and control box, the sensors, and the water pump are purposefully long. Though the power

and control box has adequate insulation and moisture protection, it is advised for the user to use the longer wire connections to prudently place the water pump, power and control box, and plants with submerged sensors a fair distance away from each other to minimize the risk of moisture and environmental factors damaging the system.

2. After determining the desired number of irrigated plants, use scissors to cut the adequate lengths of vinyl tubing needed to irrigate these plants. One end of the tubing will be secured to the water pump while another end will be aimed at the user's plants for irrigation. Take adequate precaution to secure one end of the vinyl tubing to the desired plant. Note that though this system is designed to support a large number of the user's plants, the strength of the flow of the water will be affected by the length of the vinyl tubing used for irrigation. If the chosen length of the vinyl tubing is too long, this may negatively affect the strength of the flow of water required for the function of this system.

Figure 49: Installation Example

8.6.4 Control Instructions/Display/Chart of Water Settings

1. After completing the previous setup steps, plugging the power control box into a wall outlet, and downloading the necessary app to control the system, the next step necessary for function is to actually control the automatic irrigation system through the user's phone. The system's phone application will give three different options with customized care settings for three different types of indoor or outdoor plants. The three different options and their function are depicted in the chart below. Controls and preferences can easily be canceled and changed by the user through the onscreen prompts. However, to avoid software issues, the user is advised to not change the user preferences for their plant's irrigation when the

system is in the process of watering the plants. Please wait for the system to finish its current process before changing the control preferences.

2. During ideal functionality of the system, the system will automatically irrigate the user's plants when the sensor finds less than ideal moisture levels required for the health of the type of plant entered by the user. The system will also alert the user of less than adequate temperature levels required for the health of their plant.

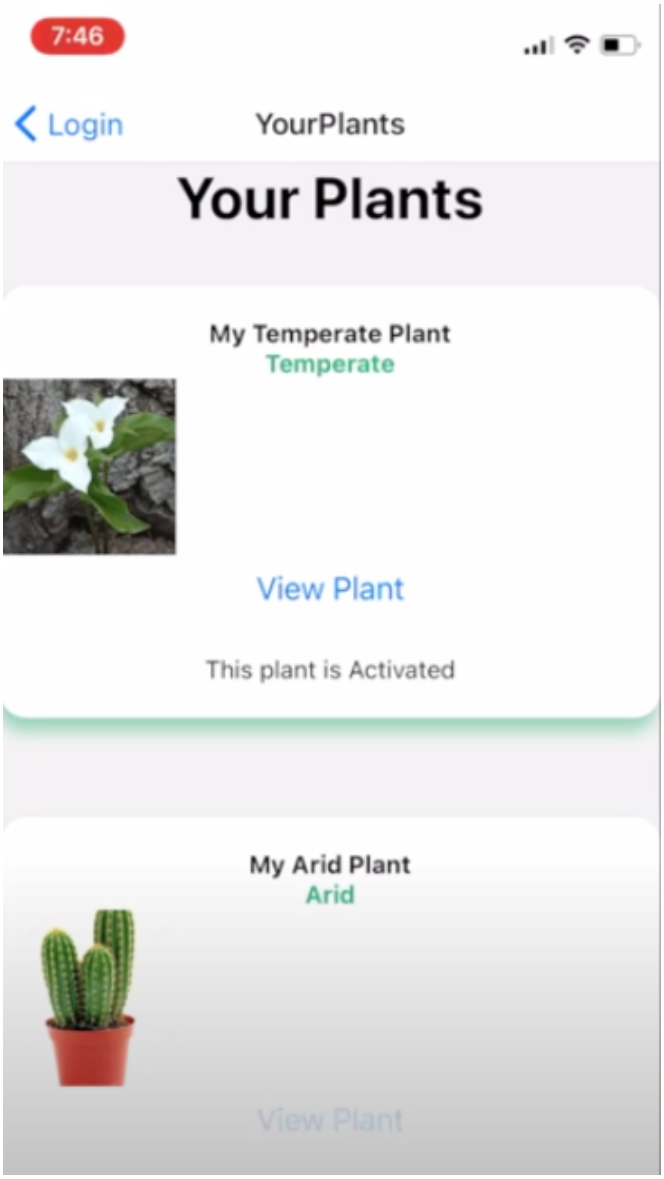


Figure 50: User Dashboard Example

User Water Control	Moisture and Temperature Settings Dependent on Type	Watering Frequency	Sample Types of Plants
Tropical Plant	High moisture and high temperature	Frequent	Palms, Elephants Ear, Bromeliad, Arthurium
Temperate Plant	Middle moisture and middle temperature	Less frequent	Ferns, Snake Plants, Orchids, Evergreens
Cactus/Succulent	Low moisture and high temperature	Infrequent	Jade plants, Aloe, Cactus

Table 32: Plant Type Watering Settings

8.7 Overall Schematic

This section details a flowchart of the overall schematic detailing the operation of this prototype automatic irrigation system. It also serves as a summary of the overall system.

The table below details the parts chosen for the prototype.

Selected Part or Function	Purpose
MERN	Cross platform development stack
Vercel	Hosting platform
ATmega	MCU
L298N	Motor controller
Tenergy 12V	Battery
Valefod Buck Converters	Manage voltage
TMP275	Temperature sensor
Songhe Sensor	Soil moisture sensor
ESP-12F	Wi-Fi Module
6V Mini Brushless Pump	Water Pump

Table 33: Chosen Parts for Prototype

The figure below is the flowchart representing our overall schematic of every component present in our design. The three main modules are the Control Software and Hardware, UI Software and Communication Hardware, and Power Module. The first module is functionally responsible for the control of the actual function of the automatic irrigation system. The second module is responsible for the communications and initial transfer of data between the user and the system itself. The last module is required for actually powering the unit as a whole.

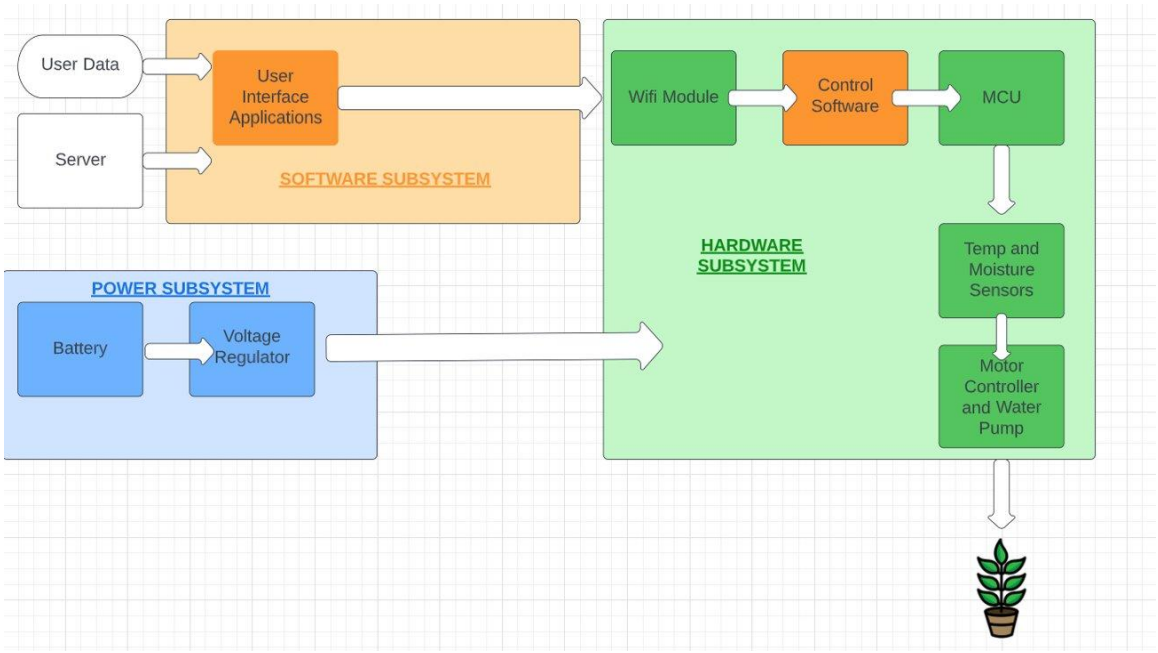


Figure 51: Schematic Flowchart

Each of these sub modules are responsible for fulfilling the basic requirements that we set out in the beginning of this project.

Control Software and Hardware

- The system has two sensors to monitor soil moisture of the plant and temperature.
- The system used a water pump to draw water from a reservoir of water outside of the system i.e., a bucket or pot of water.

UI Software and Communication Hardware

- The system automatically waters one plant using a water pump with amounts of water based on one of the three possible types of plants that the user inputs, the temperature, and the moisture present in the soil of the plant.

- The program of the system was able to compile and correctly output the expected actions deemed by the user input.

Power Module

- The power system was capable of delivering at least 12V DC.
- The power supply was capable of powering the sensors.
- The power system has current leakage protection.

9.0 Administration

The concluding section details administrative sections needed for the understanding of this project. This section contains project development timelines, budget and finance tables, division of work tables, and an explanation to important project design issues.

Below is the tentative timeline of the project:

	Task	Start	Due
Senior Design 1			
	Initial Divide and Conquer	May 18, 2022	June 3, 2022
	Updated Divide and Conquer	June 8, 2022	June 17, 2022
	60 Page Draft	June 15, 2022	July 8, 2022
	100 Page Report Submission	July 9, 2022	July 22, 2022
	Final Document	July 23 2022	August 2, 2022
Senior Design 2			
	Assemble Prototype	9-29	11-21
	Testing and Redesign	9-29	11-21
	Finalize Prototype	10-31	11-21
	Peer Report	11-30	12-6
	Final Documentation	11-30	12-6
	Final Presentation	11-3	11-30

Table 34: Project Timeline

Below is a tentative timeline of the software development of the project:

Software Development Timeline		
Task	**Start Date**	**Finish Date**
Deploy basic website to ensure hosting is working properly	July 9, 2022	July 15, 2022
Design/mockup website and mobile pages	July 5, 2022	July 31 , 2022
Set up database on MongoDB	August 3, 2022	August 10, 2022
Code core pages of the website	August 10, 2022	August 30, 2022
Code core pages for the mobile application	August 20, 2022	September 5, 2022
Refine pages for both web and mobile	September 5, 2022	September 15, 2022
Thoroughly test and ensure all code is covered	September 15,2022	September 25, 2022
UI update / UI touch up	September 26, 2022	October 5, 2022
Finalize web and mobile application	October 6, 2022	October 21, 2022

Table 35: Software Development Timeline

9.1 Budget and Finance

Below is an estimate of the budget of this project:

Subsystem	Item	Quantity	Vendor	Estimated Cost
Power Subsystem	12V rechargeable Battery	1	Amazon	$22
MCU Subsystem	TBD	TBD	TBD	TBD
Water Reservoir	Tupperware	1	Walmart	$10
Tubing	Aquarium tubing	1	Amazon	$2
Temp Sensor	Temp sensor probe	1	GikFun	$3
Moisture Sensor	Soil Sensor	1	Almocn	$3
Water Pump	12V DC Motor	1	Amazon	$16
Voltage Step down	Buck Converter	6	Amazon	$11
Pump Control	Pump Controller	1	Qunqi	$7
Pump and tube combo	pump and tube combo	4	Amazon	$12

Table 36: Proposed Budget

Below is the actual bill of material, which keeps track of spending:

Part Name	Description	Order Date	Receive Date	Cost
LM2596	Buck Converter	6/26/2022	6/28/2022	$10.99
L298N	Motor Drive Controller Board	6/26/2022	6/28/2022	$6.99
Tubing	Penn-Plax Standard tubing	6/26/2022	6/28/2022	$1.73
Tenergy NiMH Battery	Battery: 12V 2000mAh	6/26/2022	6/28/2022	$21.99
Pump	12V Mini Brushless AC Water Pump	6/26/2022	6/28/2022	$15.96
ATMEGA328P-PU	Microcontroller Chips	10/24/2022	11/1/2022	$19.50
Pump and Tubing	Combo pump and tubing kit (3-5V pump) (4 Pumps)	6/26/2022	6/28/2022	$11.39
12V Power Supply		9/21/2022	9/22/2022	$8.89
Switch		10/16/2022	10/17/2022	$7.99
Songhe Camp MS	Soil Moisture Sensor	7/6/2022	7/7/2022	$10.99
ESP8266-12F	Wifi Module	7/6/2022	7/7/2022	$8.99
TMP275AIDRG4	Temperature sensor	7/11/2022	7/14/20-22	$11.84
OPT4001YMNR	Light sensor	7/11/2022	7/14/2022	$11.14
MSP430FR2475TRHA	MCU	7/12/2022	7/15/2022	$17.93
TPS561201DDCR	3.3V voltage regulator	9/28/2022	10/3/2022	$2.07
NLCV32T-3R3M-EFR	Inductor	9/28/2022	10/3/2022	$1.08
CON-SOCJ-2155	Power Jack	9/28/2022	10/3/2022	$1.00
1N4007RLG	Diodes	9/28/2022	10/3/2022	$4.44
shipping				$7.99
L7805CV	5V voltage regulator	10/24/2022	10/27/2022	$3.45
PJ-102A	Power Jack	10/24/2022	10/27/2022	$3.50
TB002-500-02BE	Fixed Terminal Blocks	10/24/2022	10/27/2022	$3.90
shipping				$7.99
PDSE1-S12-S3-D	3.3V voltage regulator	10/31/2022	11/3/2022	$15.60
shipping				$7.99

Pin Header		10/24/2022	10/26/2022	$5.00
ESP Dev Board		10/23/2022	10/25/2022	$28.00
Capacitors		11/3/2022	11/5/2022	$6.00
Crystals		11/3/2022	11/5/2022	$5.50
TMP275AIDRG4	Temperature sensor	10/7/2022	10/12/2022	$25.74
PCB 1		9/15/2022	9/29/2022	$40.02
PCB 2		10/10/2022	10/17/2022	$11.05
PCB 3		10/23/2022	10/30/2022	$40.09
PCB 4		11/3/2022	11/10/2022	$56.10
Total Cost				$442.83

Table 37: Bill of Materials

9.2 Division of Work

The table below breaks up the project into different tasks and lists a Lead Designer, and an Integrator. The Lead Designer is responsible for the overall design, and the Integrator looks past just the design and takes other systems and or subsystems into consideration and will recommend adjustments to the lead designer's work, in order to ensure proper implementation with the system as a whole.

Task	Lead Designer	Integrator
Power Supply	Kyle	Francis
Designing Applications	Adam	Adam
PCB Design	Matthew	Matthew, Adam
Sensor integration	Francis	Kyle
MCU logic design	Matthew	Matthew, Adam
Coding Applications	Adam	Adam, Matthew
Power distribution	Kyle	Francis
Hardware Testing	Kyle, Francis	Matthew
Code Unit Testing	Adam	Adam, Matthew
PCB to Website Communications	Francis	Adam

Table 38: Division of Work

9.3 Issues to overcome

This section will go in-depth with any issues we have as we go through the implementation stages. It will touch upon software issues, hardware issues, and or unforeseen constraints that were not taken into consideration earlier on in the project design.

9.3.1 Project Design Problems

All design processes will experience several problems, may it be in the initial design process itself, the implementation stage, or the testing stage. The problems below have been documented as a record of process for the project as well as to demonstrate an understanding of issues that may have been created and how the team learned from them.

9.3.2 Initial Design Process Issues

Due to irrigation being a relatively simple problem, one of the main issues dealt with was the potential issue of over design. As engineers, it is very easy to approach this with a “cool feature” mindset, or focusing the design of our system on a wish list of features that we as a team may find enjoyable to implement. However, through market and industry research, we began to understand the danger of potentially over designing a concept for a problem that is relatively simple. The complexity of the design should really only be proportional to the complexity of the problem because over design can lead to issues like difficulty in implementation in comparison to the complexity of the problem as well as an unreasonable increase in overall cost.

Another potential issue is the implementation of the chassis design. Though the use of a chassis containment unit was utilized in the DIY models presented in the industry research, the actual implementation of a prototype unit demonstrated how a chassis design may not be preferable functionally, though it enhances the aesthetic value of the overall automatic irrigation system. This is because it greatly weakens the strength of the stream coming from the water pump since the vinyl tubing is hung from the rafters of the chassis containment unit. Therefore, the more modular “Snip N Drip” design will be used for this automatic irrigation system. However, if the implementation of the current system is on time and if the implementation is not too difficult, a chassis containment unit could possibly be created for this design in order to add more aesthetic value to the automatic irrigation system as an optional add on to the system. However, the vinyl tubing will have to be placed in such a way so that it does not disrupt the function of the water pump. Another DIY model prototype related issue is the potential issues between the voltage needs of the water pump and the voltage requirements of the MCU PCB. The DIY model prototype that was utilized acknowledged the vast differences in these two component’s voltage requirements. The water pump needed a voltage of about 12V but the Arduino could only tolerate 5V. Therefore, the kit company provided a “shield” to attach to

the Arduino that could bridge the gap between these two components. By acknowledging the solution that the kit company implemented in their design, our team will have to overcome this obstacle through a design of our own. Similar to the Arduino which is essentially both a power and MCU PCB, our design utilized one single PCB with a power sub-system and MCU sub-system.

Another issue that arose in designing our project was the possibility of water from the pump getting on to the PCB. This was fixed with the implementation of our storage box, laser cut in the UCF TI workshop. This allowed us to ensure all sensitive electronics were kept in a container that would minimize the exposure to water from the pump. This box was designed with holes in every side to ensure adequate heat sink, as well as for sensor and pump connections to exit from the PCB.

9.3.3 Testing Stage Issues

A common issue when testing was dealing with components that are too small to use a breadboard with. The proper way to deal with this would be to buy a board that lets the user solder the piece onto it, and then it has leads that connect it to breadboard compatible testing sites. This would require soldering tools and the part to pcb converting piece to operate. A work around was devised that was cheap and simple for testing parts. The way to test small parts, such as the Wi-Fi module, was to take a piece of insulation foam and place the module on top. The connection wires would then be fed through the connection leads of the Wi-Fi module and into the foam to keep them in place. This method was successful when testing out the Wi-Fi module, but had limited success when testing out the temperature sensor. The temperature sensor was smaller, had little surface contact with the foam, and did not have holes to put the wires in. An additional piece of sticky adhesive was added to keep the temperature sensor in place and the wires were placed inside the foam touching the leads of the sensor. This method was not consistent, but eventually it worked enough to establish communications with the sensor. The light sensor is too small for this method and it will be a requirement to develop another method of testing.

Another issue when testing was short circuits, which caused massive amounts of current to pass through a device which melted the internal parts, rendering the part inoperable. This was caused by the solutions presented in the previous paragraph. This happened with the Wi-Fi module when connecting the power system to the sensor system. The short existed before when testing with the Arduino, but it could not supply enough power, so the short went undetected. This issue did not result in any setbacks, as there are three additional spare Wi-Fi modules that could be used. This affects our design process by making sure that the communications and sensor part of the PCB will have limited power going to it as to prevent device failure. New testing methods will need to be implemented, which include soldering connections to the part and bypassing any foam testing structures.

The first major design change occurred when we switched from the MSP430 chip to the ATMega chip. This was done to save large amounts of development time. The ATMega and the ESP-12E were developed on the same platform which allowed for quick integration. This freed up time to work on feature creation. Another issue with the MSP430 chip was it was too small to solder by hand, which we would have had to do with our PCB manufacturer. The bootloader and software uploading technique was also untested, but was done with the ATMega chip already. Another design aspect that changed several times was the numerous changes to the 3.3V system. Our first design had a part that was out of stock until late December. Our second design had a voltage regulator that was too small. With size and availability as important specifications, we switched to our final 3.3V regulator.

In the last week of our project we received our final iteration of our PCB. This 4th generation PCB was the final possible PCB design and we were confident it would work. We soldered all of our components to the PCB and plugged all of the sensors in. We connected power and heard the pump run initially. Soon after this trial run, the PCB stopped working and we thought we had wrong traces or connections. We learned that the motor controller was supplying too much current due to the current inrush when the motor would turn on. This caused our Crystal oscillator to be fried, and thus our ATMEGA chip would not function properly. Once we switched to a good Crystal oscillator and from a 12 volt pump to a 5 volt pump our PCB worked flawlessly.

10.0 Conclusion and Summary

Our project provides amateur gardeners with an easy to install watering system to their already existing garden to automate the watering process and to record the climate the plants are in as well as control how much water their plants get. This will allow the user to take extended vacations and not have to worry about forgetting to water their favorite plant. The analysis tools that will be provided can be used to troubleshoot potential issues with the plant, such as them not receiving enough sunlight or water.

This project contains an apparatus that waters plants and senses the environment, with an included website and phone app that focuses on long term health of the plants. A record of the environment as well as actions taken by the system are logged and analyzed. The user can adjust the type of plant they are using with our system. Below is the completed system:

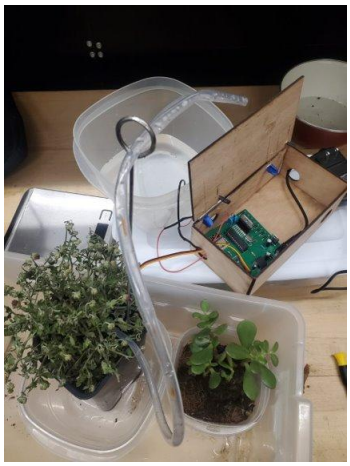


Figure 52: Final System

All basic goals were able to be reached. The PCB can successfully measure the soil moisture and temperature. This data was able to be processed by the MCU and was able to be sent and stored on the website. The user was able to create a new account and create and add plants on the website or mobile application. The data was successfully sent to the right user and plant.

11.0 Appendix

References

1. *Angular*, https://angular.io/docs.
2. Author Cadence System Analysis, et al. “Heat Dissipation in Electronic Devices.” *Cadence*, 3 Feb. 2022, https://resources.system-analysis.cadence.com/blog/msa2022-heat-dissipation-in-electronic-devices.
3. “Different Wi-Fi Protocols and Data Rates.” *Intel*, https://www.intel.com/content/www/us/en/support/articles/000005725/wireless/legacy-intel-wireless-products.html.
4. “Getting Started.” *React*, https://reactjs.org/docs/getting-started.html.
5. “I2C Temperature Sensors Derived from the LM75.” *I2C Temperature Sensors Derived from the LM75 - Arduino Reference*, https://www.arduino.cc/reference/en/libraries/i2c-temperature-sensors-derived-from-the-lm75/.
6. “Introduction: Vue.js.” *Introduction | Vue.js*, https://vuejs.org/guide/introduction.html.
7. “IP Ratings.” *IEC*, https://www.iec.ch/ip-ratings#:~:text=Electric%20and%20electronic%20equipment%20deteriorate,are%20widely%20used%20throughout%20industry.
8. “Microcontrollers - MCU.” *Mouser*, https://www.mouser.com/c/semiconductors/embedded-processors-controllers/microcontrollers-mcu.
9. “New Products.” *Analog | Embedded Processing | Semiconductor Company | TI.com*, https://www.ti.com/.
10. “Next.js.” *Getting Started*, https://nextjs.org/docs/getting-started.
11. Rolyon. “Azure Documentation.” *Microsoft Docs*, https://docs.microsoft.com/en-us/azure/?product=popular.
12. Santos, Rui, et al. “Best ESP8266 Wi-Fi Development Board - Buying Guide 2020.” *Maker Advisor*, 31 May 2021, https://makeradvisor.com/best-esp8266-wi-fi-development-board/.
13. Tstef, and Instructables. “ESP-12E (ESP8266) with Arduino Uno: Getting Connected.” *Instructables*, Instructables, 25 Mar. 2019, https://www.instructables.com/ESP-12E-ESP8266-With-Arduino-Uno-Getting-Connected/.

14. *Website Redesign Checklist: 5 Trends to Consider*, https://topdesignfirms.com/web-design/blog/website-redesign-checklist.

15. “Xamarin: Open-Source Mobile App Platform for .NET.” *Microsoft*, https://dotnet.microsoft.com/en-us/apps/xamarin.

16. “Components and Props.” *React*, https://reactjs.org/docs/components-and-props.html.

17. “The Developer Data Platform.” *MongoDB*, https://www.mongodb.com/.

18. “Firebase Documentation.” *Google*, Google, https://firebase.google.com/docs.

19. “Flutter (Software).” *Wikipedia*, Wikimedia Foundation, 19 July 2022, https://en.wikipedia.org/wiki/Flutter_(software).

20. “Flutter Documentation.” *Flutter*, https://docs.flutter.dev/.

21. Inc., Apple. *Swift.org*, https://www.swift.org/documentation/.

22. *Ipřiro*, https://www.ipriro.com/products/sPlant-Big-Power-Automatic-Drip-Irrigation-Kit-Indoor-Plants-Self-Watering-System%40Tex%20Watson%20%EF%BF%BC%20you%20dont%20get%20this.

23. Jacobs, David. “Compare Low-Power Wi-Fi Protocols and Their Roles in IOT.” *SearchNetworking*, TechTarget, 16 Oct. 2020, https://www.techtarget.com/searchnetworking/tip/Compare-low-power-Wi-Fi-protocols-and-their-roles-in-IoT.

24. Maker, Orion, and Instructables. “IOT Automatic Plant Watering System.” *Instructables*, Instructables, 5 Nov. 2017, https://www.instructables.com/IoT-Automatic-Plant-Watering-System.

25. “MySQL Documentation.” *MySQL*, https://dev.mysql.com/doc/.

26. R, Kamal. “Bluetooth vs. WIFI - Which Is Better for Connectivity for IOT Development?” *Top Mobile App Development Company*, https://www.intuz.com/blog/bluetooth-vs-wifi-connectivity-for-iot-development.

27. “React Native.” *Wikipedia*, Wikimedia Foundation, 29 July 2022, https://en.wikipedia.org/wiki/React_Native.

28. Vercel. “Introduction to Vercel.” *Vercel Documentation*, Vercel, https://vercel.com/docs.

29. “What Is Mysql? & Why It Is the World's Most Popular Open Source Database.” *MySQL Tutorial*, 11 Apr. 2020, https://www.mysqltutorial.org/what-is-mysql/.

30. “What Is React Native? Complex Guide for 2022.” *What Is React Native? Complex Guide for 2022*, https://www.netguru.com/glossary/react-native.

31. “NiMH Battery Charging Basics.” *NiMH Battery Technology, How to Charge Nickel Metal Hydride Batteries Tutorial for Design Engineers, as Well as NiMH Chargers.*, 6 June 2022, https://www.powerstream.com/NiMH.htm.

32. “Houseplant Statistics in 2022 (Incl.. Covid & Millennials).” *Garden Pals*, 9 May 2022, https://gardenpals.com/houseplant-statistics/.

33. “97/204158 DC : Draft Apr 1997.” *SAI Global Store*, 23 Nov. 2012, https://infostore.saiglobal.com/en-us/standards/97-204158-dc-draft-apr-1997-253987_saig_bsi_bsi_589595/.

34. “IEC.” IEC 60896-11:2002 | IEC Webstore | Battery, Energy Efficiency, Smart City, 4 Dec. 2002, https://webstore.iec.ch/publication/3849.

35. “IEC.” *IEC 62485-2:2010 | IEC Webstore | Rural Electrification, Energy Storage, Battery, Energy Efficiency, Smart City*, 16 June 2010, https://webstore.iec.ch/publication/7091.

36. “IEEE SA - IEEE Recommended Practice for Sizing Lead-Acid Batteries for Stationary Applications.” *IEEE Standards Association*, 5 June 2020, https://standards.ieee.org/ieee/485/6726/.