GONZAGA UNIVERSITY
School of Engineering and Applied Science
Center for Engineering Design and Entrepreneurship

PROJECT PLAN
10/02/2024

Smart Watering System with IoT
CS25-09

Prepared by:

Signed by:
Arvand Marandi
36E7574D256646D...
Arvand Marandi

Signed by:
[illegible]
293FC9DECC85486...
Madison Spink

Signed by:
[illegible]
14C70FFDEE1F452...
Caleb Lefcort

Signed by:
Matt Nguyen
D99CADB61DF942E...
Matt Nguyen

Reviewed by:

DocuSigned by:
[illegible]
B1E07857B76E4C6...
Rafael Pozos
Faculty Project Advisor

Signed by:
Roberto Valenti
91E499B2DF254BD...
Roberto Valentini
Project Sponsor/Liaison

Signed by:
Kayla Larson
DE7395ED35BD4B4...
Kayla Larson
Design Advisory Board Member

Signed by:
Harish Chintakunta
964F0E2CE2524FB...
Harish Chintakunta
Project Sponsor/Liaison

Project Overview

1.1 Project Summary

Water conservation is a critical challenge in agriculture. Traditional irrigation methods often waste resources and even harm plants due to inefficient water distribution. Our project aims to leverage technology to optimize indoor irrigation systems, ensuring plants receive the precise amount of water they need at the right time with minimal human involvement.

To accomplish this, we are developing a smart watering system that combines IoT sensors and MathWorks tooling to enable real-time data monitoring and analysis of indoor gardening. The IoT devices will capture key environmental conditions such as soil moisture, while ThingSpeak and MATLAB process this data. A state machine will then determine if the plants require watering. If they do, the system will send a notification, allowing the user to react by starting the watering strategy or aborting. The application may have other possible actions the user can choose from which would incorporate additional industry-relevant technologies. This solution aims to minimize water consumption and optimize grow conditions around Gonzaga.

1.2 Project Objectives

Of the desired outcomes, this project's primary business objective is to show the utility of MathWorks' tools in developing an IoT-enabled smart watering system. In application, this will be done using MathWorks' ThingSpeak plugin to facilitate communication between IoT devices and MATLAB to process the produced data. Conclusions will then be drawn from this data using a state machine.

In conjunction with the latter, another objective of this project is to optimize the system. Given that the project will be hosted in an indoor gardening setting, this means our system will produce the most growth it can (sensory data being the limiting factor) while using a minimal amount of water.

1.3 Project Stakeholders

Irrigators Development Team
The Irrigators aim to gain technical and industry experience and develop project planning and management skills.

MathWorks
MathWorks, through liaison Roberto Valenti, aims to promote MathWorks' ThingSpeak while solving industry leading problems such as efficient water usage in agriculture.

Project Advisors
The project advisor's goals are to promote the success of the project and facilitate the learning process within the Irrigators team, as well as provide extra support in communication with the sponsor.

DAB Members
The members of the Design Advisory board are here to facilitate the design process and execution of the project.

Target Users
Our target users for this product are gardeners and small farmers aiming to optimize water usage by automating irrigation. We are focusing primarily on indoor gardening with indoor irrigation systems, what may receive more water than needed with an automatic irrigation system that cannot respond to relevant elements like soil water retention and weather conditions when on a timer system.

1.4 Project Deliverables

System Simulation
The main deliverable is the smart watering system that uses a state machine model to manage plant irrigation. Initially, the system will use mock sensor outputs, simulating soil moisture values based on real values. The sensors will transmit this data to ThingSpeak for processing, and the system will determine an appropriate watering strategy using the state machine. That watering strategy, alongside other relevant data, will be displayed in a web application connected to ThingSpeak that the user can interact with.

(a) **Mock Sensor Data**
This deliverable will simulate the garden's soil moisture conditions in software, allowing the team to test different watering strategies without relying on physical sensors and budget. The mocked values will be sent to ThingSpeak via it's API.

(b) **State Machine**
This deliverable will house the logic determining when and how to water the plants based on the mocked sensor data. The state machine will ultimately decide the watering strategy and will be written in MATLAB.

(c) **ThingSpeak Integration**
This deliverable will involve setting up and configuring ThingSpeak to collect, process, and analyze environmental data from the mocked sensors. ThingSpeak will communicate with the system's state machine to determine a watering strategy based on the processed data.

System Analysis
The deliverable will define how the system responds to the state machine's output. To do this, we will develop a web application that is connected to ThingSpeak. This application will display the output of the state machine, alongside a dashboard of real-time sensor data, estimated water usage, previous watering history, and ability to actuate/abort the watering system.

Hardware Deployment
This deliverable focuses on implementing the system on physical hardware, if time and budget allows. It will include integrating real sensors, a microcontroller, drip-irrigation system, ventilation, pump, and anything else deemed necessary. This stage transitions from the simulated environment to practical application.

Documentation
The project will include user and developer documentation. The user documentation will detail how to set up and operate the system, while the developer manual will explain the technical architecture, state machine logic, and integration with ThingSpeak and other technologies used. The team will provide both documents in PDF format in the project repository.

1.5 Project Scope

Much of this project will be handled in-house as most of the components fall within the scope of the project. While we will be using MathWorks tools to build this project (MATLAB) and collected data will be sent to and processed on MathWorks's ThingSpeak we are not considering this a separate service because tools for data aggregation, analytics and visualizations will be developed in house on ThingSpeak.

The primary external sources of data will be the user, who we expect to input relevant crop data. The primary output of the smart watering system will be a command to turn on or off an irrigation system either to the system itself or to a user operating the irrigation system. We also plan to use ThingSpeak's data visualization tools to provide reports of the analyzed sensor data to the user.

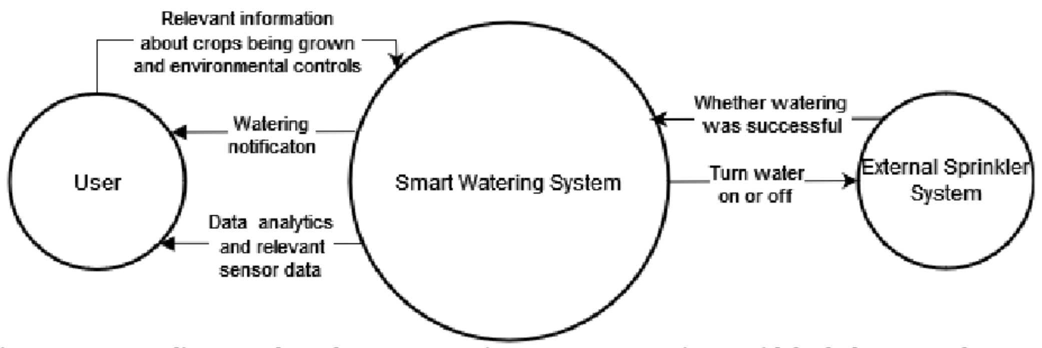

Relevant information about crops being grown and environmental controls
Watering notificaton
Data analytics and relevant sensor data
User
Smart Watering System
Whether watering was successful
Turn water on or off
External Sprinkler System

Figure 1: Context diagram where the smart watering system communicates with both the user and external sprinkler system. The Smart watering system requests and receives data from the user and sends commands to and receives success or failure notifications from the sprinkler system.

1.6 Related Work

There have been multiple implementations of smart irrigation projects. Some of these include a smart watering system for strawberries in a greenhouse, a Nigerian low-cost watering system and information relay to farmers, and an automated watering system for green walls. Many of these projects take a similar approach to our design first sensor data is collected then sent to the central system via various channels (Wi-Fi, GSM, LoRa, MQTI). The central system then aggregates the data and decides based on the information received. Some projects also include weather data when informing their decisions. Once the decision is made the system either messages the user that the plants need watering (via Email or SMS), or the system sends a message to a pump system initiating a watering cycle.

The most similar approach to our design was conducted by a group of researchers at CVR College of Engineering. They collected moisture, temperature, and humidity information. This was then sent to a NodeMCU microcontroller unit via Wi-Fi and a decision is made based on certain thresholds. The data was also sent to a mobile app for monitoring sensor data. Finally, the decision was relayed to a pump system that waters the crop. The general design of this project is similar to our own with the main difference being the use of a mobile app for relaying information. Our project will likely use a web interface instead. The other major difference is our use of the MathWorks ecosystem of tools. Much of our project will be built with these tools so the underlying technologies will be different. We are also developing an indoor system which differs from their outdoor system. This will allow us to have more control over the environment and affect more variables.

Project Requirements

2.1 Major Features

Table 1: Major Features

Feature	*Description*
System Simulation	This feature involves creating a simulation that includes mocked sensor data, a state machine for watering decision-making, and integration with ThingSpeak will allow for communication between IoT modules and data analysis. The system simulation allows for the testing of system behavior in a test environment, ensuring that the watering logic is effective and reliable before deploying to actual hardware.
Hardware Deployment	This feature focuses on implementing the system on physical hardware. It includes integrating real sensors, a microcontroller, and other necessary components to deploy the system. This stage transitions from the simulated environment to practical application. Hardware deployment is essential for validating the system's performance and usefulness in real conditions.
System Analysis	A web-based application will be developed to allow users to monitor sensor data and configure parameters such as soil moisture, and eventually, temperature and humidity. This interface provides real-time insights and control over the watering system. This improves usability and ensures users have control over their watering needs.
Documentation	The project will include user and developer documentation. The user documentation will detail how to set up and operate the system, while the

	developer manual will explain both the hardware and technical architecture of the system, state machine logic, and integration with ThingSpeak and other technologies used. Detailed documentation supports the usability and maintainability of the system, and therefore, is a major feature.

2.2 Initial Product Backlog

Table 2: Initial Product Backlog

Requirement	*Description*	*Major Feature*	*Priority*	*Estimate*
Gather Mocked Sensor Data	Gather or generate mock sensory data and upload it to ThingSpeak. Mocked data should be a realistic reflection of data produced from accessible sensors for the project.	System Simulation	High	5-7 days
Send Mocked data	The mocked data will be sent to ThingSpeak.	System Simulation	High	2-4 days
State machine built in MATLAB	The state machine will take the data provided by ThingSpeak and produce an adequate response.	System Simulation	High	3-5 weeks
Send decision	The decision from the state machine will be sent to the action hardware.	System Simulation	High	1-2 weeks
Create budget	Decided which pieces of hardware will fit in the budget as well as cost of running the hardware (electricity and water).	Hardware Deployment	High	5-7 days
Purchase Hardware	If budget allows purchase hardware for the project.	Hardware Deployment	Low	2-4 days
Deploy to Hardware	Once the system simulation is in-place and robust, deploy to hardware for real-world use cases.	Hardware Deployment	Low	4-7 weeks
Environment Setup	Access MATLAB, ThingSpeak, and any other necessary tools.	*System Simulation*	High	2-4 days
User Interface	Users will be able to configure the system and monitor sensory data.	*System Analysis*	Medium	4-7 weeks

2.2 Additional Features

Table 2: Additional Features

Feature	*Description*
Incorporate industry relevant technology	To tailor the project to our interests in systems programming, we want to gain experience with other industry-relevant technologies like Rust. Incorporating these technologies into the project can demonstrate MathWorks' ability to integrate with other popular tools, while also allowing us to build valuable

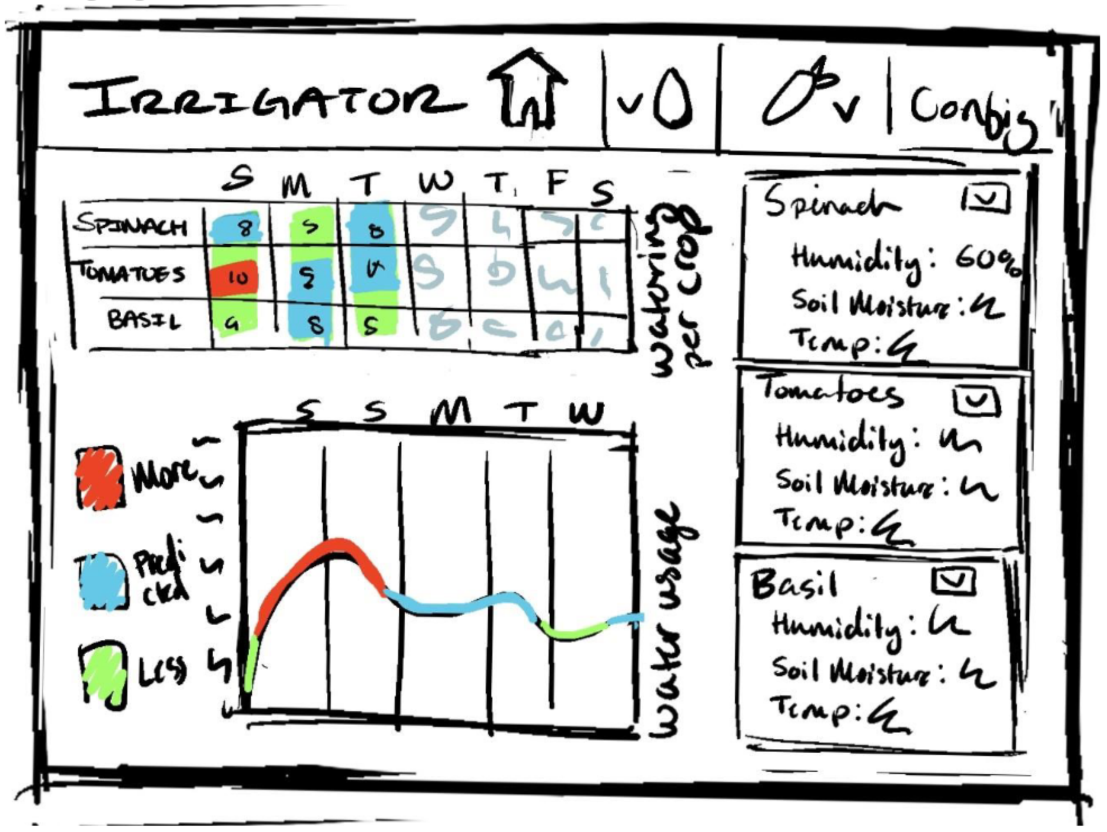
	expertise. This could be useful in the user-interface or "system analysis" component of the project, while the fundamental parts would remain in MathWorks tooling.
Preload profiles for common crops.	For the initial implementation the user will set the desired measurables for the grow environment. A nice feature to add would be pre-loaded profiles that would set these measurables based on the plant the user selects. The advantage of this would be less for the user as they would not need to know

Design Considerations

3.1 Initial User Interface Design

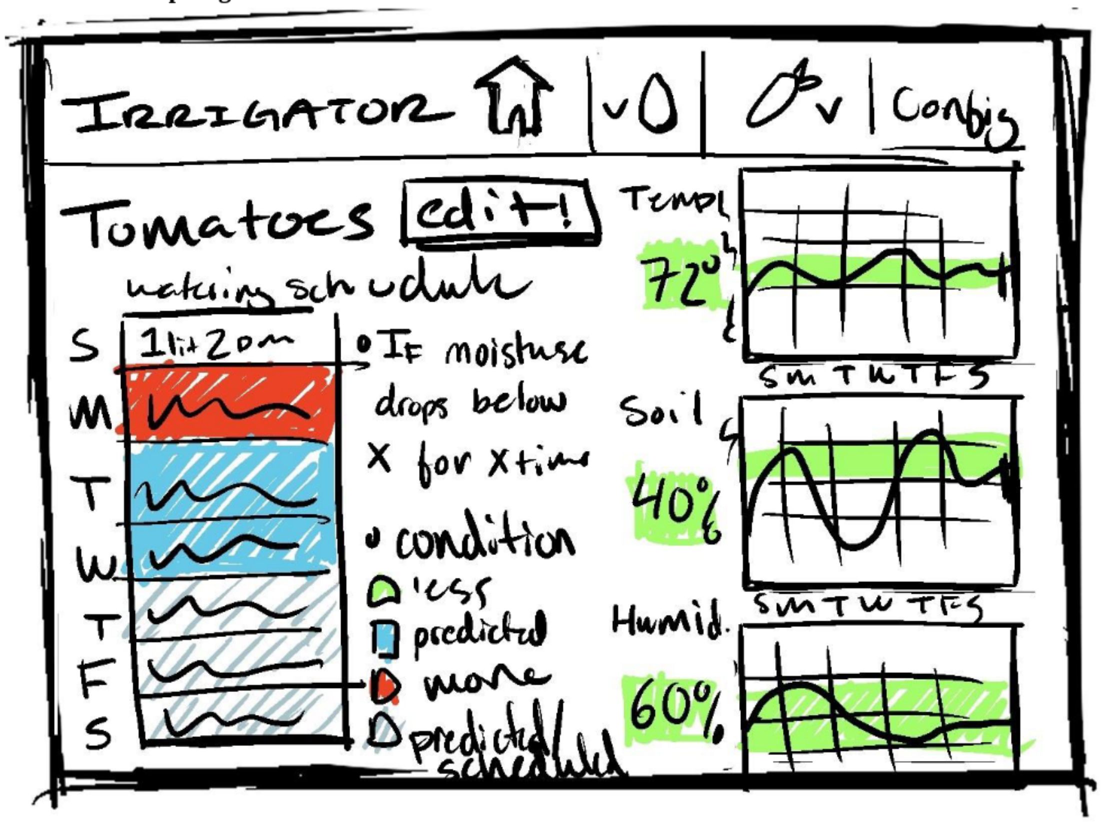
Although we plan to have most of the system controlled by the state machine, we still want to enable the user to make changes to the watering schedule and be able to veto any decisions made by the machine in the case of an exception. An interface for the initial input of crop data is also needed, such as what the soil moisture levels must be and how much they can fluctuate. Because this is an IOT device, we hope to have a web page accessible by the user to interact with the watering system.

Landing Page



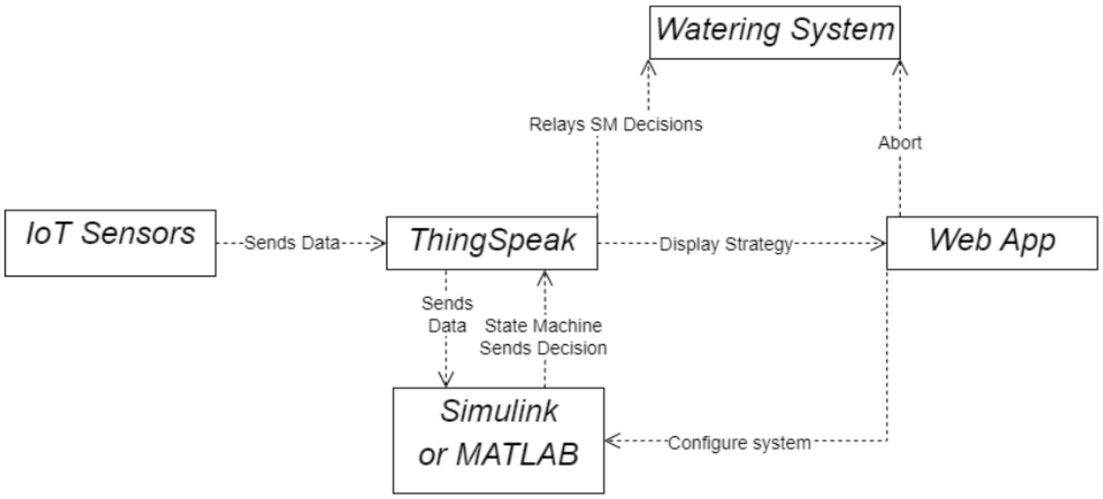
The above mock-up of the homepage has a navigation bar at the top, with options for home, viewing and editing the current water schedule and parameters, as well as viewing data for individual crops, and a page to edit any configuration settings in the system. On the left is the watering schedule, showing a weeklong calendar with the amount each crop was watered each day. For previous days it shows whether the amount of water used was over or under the amount previously scheduled in the strategy, and for future days it shows how much water is predicted to be used based on the current watering strategy. Below the calendar is a graph showing the water usage over time. To the right shows the current sensor data of all crops,

Individual Crop Page



The above page shows a page for all data for the tomato watering system. It includes the navigation bar previously described. The page has a title labeled ”tomatoes” and to its left is a button that takes the user to the page to edit any parameters or edit the watering schedule itself. Below the title is a vertical calendar showing the watering schedule. For each day there it shows the time(s) the plants were watered, as well as how much water was used. As with the homepage for future dates, it shows the scheduled times and amounts of water. In the middle, all explicit parameters for watering are displayed, such as desired soil moisture. To the left are a series of graphs displaying current sensor data as well sensor data over time.

3.2 Initial Software Architecture



Architecture Description:

- IoT sensors read data from watering system and send data to ThingSpeak.
- ThingSpeak stores sensed data and relays it to a state machine.
- The state machine, hosted on ThingSpeak, is developed in either MATLAB or Simulink.
- Watering strategy is decided by state machine based on sensor data and user configurations.
- Decisions are sent from ThingSpeak to the watering system; adjustments are made accordingly.
- ThingSpeak sends decisions and relevant data to the web app, which will be displayed to the user.
- Via the web app, users can decide to pause/abort the watering program.
- Users can configure system parameters via web app; preferences are sent to the state machine.

3.3 Development Environment, Tools, Languages, and Libraries

3.3.1. Hardware Components

- **Raspberry Pi**: Main controller responsible for collecting sensor data and controlling the irrigation system.
- **Sensors**: For detecting environmental parameters (e.g., soil moisture, temperature, humidity).
- **Actuators**: For controlling the irrigation system (e.g., pumps, valves).
- **Connectivity**: Wi-Fi module for internet connection.

3.3.2. Software Components

<u>A. Programming Languages</u>

- o **Python**:
  - Used for writing scripts on the Raspberry Pi to interface with sensors.
    - Communicating with ThinkSpeak API (HTTP requests).
    - Triggering actions (e.g., opening or closing irrigation valves).

- o **MATLAB/Simulink**:

    - For building the state machine model and simulating the logic for decision-making based on sensor data received from ThinkSpeak.
    - Auto-generating C code from Simulink models, if necessary.

B. Libraries and Frameworks

- **Raspberry Pi GPIO Library**
    - To control GPIO pins and interface with sensors/actuators.
- **Requests Library (Python)**:
    - To make HTTP requests to the ThinkSpeak API for sending sensor data and receiving aggregated data.
- **Simulink Toolboxes**:
    - Simulink Stateflow: For building state machines.
    - Simulink Coder: For converting Simulink models into C code if needed.

3.3.3. Development Tools and IDEs

- **Simulink/Matlab IDE**:
    - For creating and simulating state machines and generating response logic.
- **Visual Studio Code (VS Code)**:
    - Can be used as a lightweight editor for Python development with Raspberry Pi.

3.4 Initial Software Test Plan

Initially, we will test our project in a simulated environment to validate the functionality of the state machine logic, ThinkSpeak integration, notification system, and web app. If deemed useful, we will use unit and integration tests to further compartmentalize and evaluate that the system works as expected. This approach allows us to focus solely on the software components of the smart watering system without the complications introduced by hardware. Once we have thoroughly verified that the simulated mock-up operates as intended and meets design criteria, we will proceed to transition the system to hardware deployment. This step will involve adapting the validated software to work with actual sensors and microcontrollers, ensuring successful integration of the simulated logic with real-world conditions. This strategy minimizes risks and maximizes production deployment success if/when we get to hardware.

4 Project Risks

Poor communication between IoT components
If the IoT components are unable to communicate properly, the wellbeing of the plant may be compromised (e.g. over/under saturation, too much light). To prevent this from happening, testing will occur to ensure that connection is established. The team will actively monitor what data is read and what decisions are produced by the state machine. If a decision isn't met with its associated action, the team will investigate the matter and find a fitting solution.

Component malfunction
If an IoT component malfunctions, the wellbeing of the plant may be compromised (e.g. over/under saturation, too much light). To prevent this from happening, the team will initially ensure that the component produces proper responses. Throughout the lifetime of the project, team members will monitor each component, ensuring that proper actions are completed. If a decision isn't met with its associated action from a component, the team will investigate the matter and find a fitting solution.

Poorly written software
If a program is written improperly, the wellbeing of the plant may be compromised (e.g. over/under saturation, too much light). To prevent this from happening, the state machine will undergo testing on a set of accurate mock data. The Arduino programs will be tested by ensuring that proper actions are produced in accordance with the state machine's decisions. Once the Arduino programs are written correctly, there isn't a need for monitoring. Throughout the lifetime of the project, the state machine decisions will be monitored by the team. If incorrect decisions are produced, the state machine will be adjusted accordingly.

Oversaturation/slow drainage
If the plant becomes oversaturated, the roots might not receive enough oxygen. Additionally, too much water in the soil creates an environment for fungal growth. These both will kill the plant, if these issues are unattended to. To prevent this from happening, the team will place a rock bed at the bottom to ensure that there are air pockets, and that water has a place to drain. The risk will be monitored using a soil moisture sensor. If the soil is too moist, the Arduino will reduce the amount of water the plant receives.

Budget unable to meet system requirements
If the team is unable to meet budget requirements, necessary components of the system might be absent, resulting in the incompletion of the project. To prevent this from occurring, the team will compose a compelling budget, detailing the need for each component. If our budget cannot be approved, we will file a complaint and work with what we have (mock data) till the second round of budget applications.

Unsatisfactory mock data
If our mock data is poor, the state machine's decisions might not be applicable to the physical system, which could compromise the wellbeing of the plant. To prevent this from occurring, the team will undergo a literature review phase, learning what sort of data is to be expected. We will then either search Kaggle or other dataset websites to find an appropriate dataset or generate a dataset of our own given what we learned. The risk of an unsatisfactory dataset will be monitored throughout the lifetime of the

project by observing decisions produced by the state machine. If the state machine consistently produces incorrect decisions, we will interpolate from the data produced by the system and adjust the state machine accordingly.

Unauthorized access to physical system

If unauthorized personnel are given access to the watering system, they might introduce variables unbeknownst to the team or intentionally tamper with the system; this could compromise the wellbeing of the plant. To prevent this from occurring, the chosen environment for the room will be accessible only to team members. The system is held in Hughes Hall, and only team members have access to the room. The risk will be monitored by maintaining audit logs of room entry, ensuring only team members are permitted. If an unauthorized person accesses the room, we will investigate the matter and ensure that the event cannot be repeated.

Initial Product Release Plan

5.1 Major Milestones

Table 3: Major Milestones

Milestone	*Description*	*Target Completion Date*
Initial ThingSpeak communication	Set up the basic information flow between the thingspeak server, state machine logic, and the mock sensor system. First to ensure we have mocked data entering ThingSpeak that we can run logic over.	2nd week of October
Basic state machine	Minimal functioning MATLAB state machine that makes a watering decision solely based on mocked soil moisture data. This is the first milestone after initial communication because it is the central component of the project and other systems will be reliant on it.	2nd week of November
Basic temperature control logic	MATLAB logic to regulate tent temperature through controlling light and fans. After watering logic, we need to add other logic critical to the plant’s vitality before introducing a dashboard.	2nd week of December
Web based dashboard	Web based interface for the user to monitor sensor data and set desired settings. After watering and temperature regulation logic are in place, want a “administrator” dashboard where the user can view all relevant information surrounding watering, temperature, etc.	2nd week of February
ThingSpeak web app Integration	Connect the web app to ThingSpeak for monitoring and setting field values.	1st week of March
Initial hardware setup	Physical connection of hardware components, establishing internet connection, and converting mock sensor script to run on Arduino and make ThingSpeak API calls. Once the initial software has been written we will start implementing the hardware.	4th week of March

Rely on real hardware	Convert all mocked data to collect from real soil-moisture and temperature sensors, actuate/halt drip irrigation system, and fan/light control. Here we move away from mocking anything. At this point, the entire system is working, and we now just need to take values from real hardware.	2nd week of April
Verification of system functionality	Ensure all components are fully integrated and working together. After all logic in software is functional and we deployed to hardware, the team will perform end-to-end testing to ensure the system works in a real-life use case.	Week before final demo

5.2 Initial Sprint Releases

Table 4: Sprint Release Plan

Sprint Date	*Spring Goal*	*Backlog*	*What we will demo*
1st and 2nd weeks of October	*Initial ThingSpeak communication*	• *Environment Setup* • *Gather Mocked Sensor Data* • *Send Mocked data*	We will show the basic information flow within ThingSpeak
1st and 2nd week of November	*Basic state machine*	• *State machine built in MATLAB* • *Send decision*	We will show that the system can make a "correct" watering decision based on mocked data.
1st and 2nd week of December	*Basic temperature control logic*	• Research desirable growing temperatures • Send decision	We will show that the system can initiate actuation/halting of components to regulate temperature. Note: no hardware is set up at this point, so the system will output a decision for turning on/off fans and light.
1st and 2nd week of February	*Web based dashboard*	• *User Interface* • *Purchase Hardware*	Will show the basic functionality of the web app and allow for comments on design decisions.
1st and 2nd week of March	*ThingSpeak web app Integration/ start hardware setup*	N/A	Demo the web app integrating with ThingSpeak
1st and 2nd week of April	*Rely on real hardware*	• *Deploy to Hardware*	Demo full functionality of the system
4th week of April and 1st week of May	*Verification of system functionality*	• *Test the system in a real use-case*	Demo full functionality of the system

Maintenance Considerations

Regarding the IoT device (the Arduino and attached sensors) the maintenance will have to be provided by the owner of the physical system. As this project's scope does not include a product release, the Irrigators team are considered the sole owners of the device, and all maintenance for the IoT device is left up to us. The technical requirements for maintaining the Arduino are knowledge of how to load and execute code onto the device from an external computer and how to maintain the electrical components and sensor connections in the face of water damage.

All maintenance of the front-end webserver and the ThingSpeak channels is left up to the Irrigator development team. We do not expect that developing a system for future maintenance or upkeep will be necessary, as the project will not be further developed outside of the scope of the project outside of any personal implementations that may arise from the repository being published publicly to GitHub.

Project Management Considerations

The team will meet regularly online with the team advisor on Mondays, barring any future scheduling conflicts. We are currently using email to update all involved parties of project changes and updates, and once the project gets underway, will have semi-regular meetings with project sponsors. We plan to move most of our communications to Microsoft Teams when updates are more regular.

The project is sectioned so each member owns and partially directs the development of that aspect but is not the sole member working on it. Currently the project is split into the physical IoT device, owned by Arvand Marandi. The Arduino development, owned by Matt Nguyen. The ThingSpeak Channels and State Machine, owned by Caleb Lefcort. Water usage prediction owned by Arvand Marandi. And the front-end webserver and website owned by Madison Spink.