Plant Hydration and Weather Integration System

ECE 445 Design Document - Spring 2025

Aashish Chaubal, Jaeren Dadivas, Iker Uriarte

Project # 76

Maanas Agrawal

Contents

1 Introduction

1.1 Problem and Objective:

In today's rapidly evolving agricultural landscape, striking the perfect balance between optimal plant hydration and sustainable water conservation has never been more critical. Research shows that effective water management is one of the most pressing challenges facing agriculture, while some regions suffer from debilitating shortages, others grapple with the adverse effects of overabundance. Both extremes can severely damage crops, reducing yields and impacting food security. Historical and contemporary studies [1][2] underscore that managing water efficiently is not just a necessity for plant health, but also a key factor in ensuring economic sustainability and environmental resilience.

Our innovative device addresses these challenges by leveraging technology to optimize water usage. By precisely monitoring moisture levels and adjusting water delivery in real time, our solution ensures that every drop is used to its fullest potential. This proactive approach not only supports robust plant growth but also conserves water; a resource that is increasingly precious in many parts of the world. The system's smart integration of sensors and automated controls means that plants receive just the right amount of water, avoiding the pitfalls of both under and over watering. This results in healthier crops, reduced waste, and a more sustainable agricultural practice overall.

Beyond its technical merits, our device represents a paradigm shift in agricultural water management. It embodies a commitment to innovation and environmental stewardship, making it a crucial tool for modern farmers and agriculturalists. By addressing the nuanced challenges of

water management with precision and adaptability, our solution paves the way for a future where agricultural productivity and sustainability go hand in hand.

Our device relies on a soil moisture sensor as its main component to determine whether plants need watering. The plant will be watered only if the moisture reading is low and the weather forecast does not indicate rain. To access the weather forecast, the device will briefly connect to Wi-Fi and retrieve data from AccuWeather. This ensures that the plant never becomes dehydrated and minimizes the frequency of watering from both the pump and rain. Figure 1 illustrates the basic idea of the design.

Figure 1 is a very basic way to describe the complexity of this design, but it describes the main function of this system which is saving and utilizing water wisely. This diagram doesn’t include our rain detection subsystem or our data collection subsystem.

1.2 Visual Aid:

Figure 1: Visual aid of a simple design

The system is designed as an automated irrigation control unit which will physically consist of a central control box branching out soil moisture sensors and a water pump from an ESP32 microcontroller. At the core of the system is an insulated and waterproof enclosure to house the main electronic components. The casing is crucial to protective and shield the internal circuitry from external environmental factors such as temperature and weathering. The ESP32 acts as the brain of the design and mains the responsibilities of processing sensor data and making the decisions on when to activate the water pump to optimize water usage.

Extending from the protected control box we will have multiple soil moisture sensors connected with insulated wiring to further protect the system from water intrusion and environmental factors. Each sensor will be strategically placed in a plant's soil to continuously monitor moisture levels to transmit back to the ESP32. Coupled with the real-time weather data from a third party weather application requested via API, the ESP32 takes the information from

the soil moisture sensors to determine when additional watering is necessary for the plant's wellbeing. If the system deems additional water necessary, then the ESP32 will activate the water pump which is an external tubing system that directs water from a container into the soil. The power supply for the entire system will be controlled and operated by the microcontroller and located inside the housing to ensure safety and proper operation.

1.3 High-Level Requirements:

For our system to be considered effective and complete, it must meet a set of high-level requirements. If the project fails to satisfy these requirements, it will not adequately address the problem it was designed to solve. The high-level requirements for this project are:

1. The system must be able to measure the amount of water saved without forecast-based communication. The collected data must either be displayed on a website or stored on an SD card. If the timeline allows, both options should be implemented.
    1.1. The weight sensor subsystem shall be used to count how many times our plant gets water from rain. The times where it detects weight change must be recorded and then analyzed to study if it was caused by rain or another random event.
2. Our system must allow easy adjustment of the required moisture levels based on the specific needs of the plant.
3. The system must water the plant when soil moisture levels are critically low based on the specified plant parameters. It will also adjust the amount of water provided according to the precipitation percentage from the forecast.

3.1. If the forecast predicts a 50% chance of rain, the system will provide ⅔ of the recommended water amount, these parameters will be adjusted as we collect data.

2 Design

2.1 Functional Overview:

The design of this project is unique, as shown in Figure 2, which presents the block diagram. Each subsystem is largely self-sufficient and does not rely on many external elements, making it a modular design. The most complex subsystem is the weight sensor subsystem, as it requires multiple components to function.

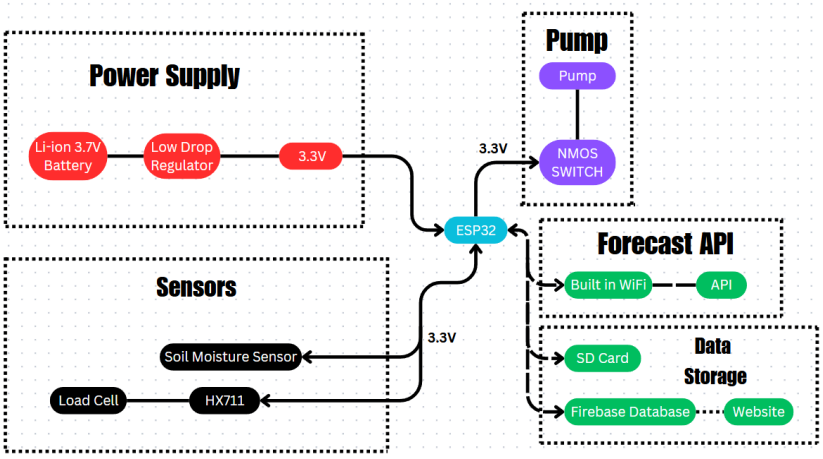

Figure 2: Block Diagram

ESP32: The ESP32 is perfect for this project because there is a vast amount of documentation for this microcontroller and we can use standard software such as Arduino IDE to program our PCB. The ESP32-WROOM is perfect as the average operating current needed is 80 mA and can operate at a reasonable 3.3V. [4]

Soil Sensor: The sensor we are planning on using is the 101020008, the specifications of this sensor can be found in Table 1. This sensor fits this project perfectly because it has a low maximum current usage and allows us to connect it to our ESP32-WROOM-32 without needing to adjust voltage.

Item	Condition	Min	Typical	Max	Unit
Voltage	-	3.3	-	5	V
Current	-	0	-	35	mA
Output Value	Sensor in dry soil Sensor in humid soil Sensor in water	0 300 700	-	300 700 950	-

Table 1: Soil Sensor Specifications [3]

Low Drop Regulator: The regulator selected to provide 3.3V is the LM1084. This regulator is a good choice due to its relative low voltage drop and high current output. Since our system has minimal current requirements, it is a perfect fit. The regulator can supply a maximum current of 5 A, but to achieve 5 A output, the battery voltage must be at least 4.8V to compensate for the voltage drop. If we use five AA batteries we can power it with a total of 6 V, which is enough to get the full load. The typical application of this regulator can be seen in Figure 3.

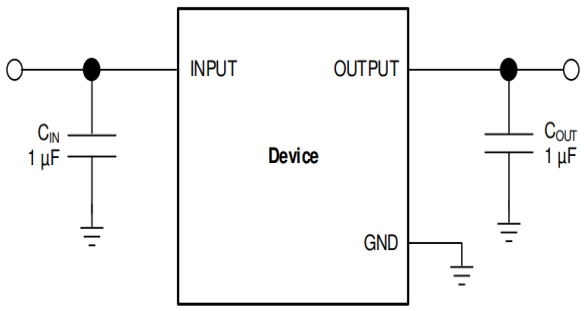

Figure 3: Typical Application [5]

Load Cell and HX711: The load cell and HX711 is a subsystem that requires two components to build. The load cell is in charge of measuring change in weight and passes the data to the HX711 that converts the analog signal from the load cell into a digital signal that goes into ESP32. The load cell we intend to use is the SEN-14728. This sensor operates with any voltage below 6V and will draw approximately 3.5 mA, which does not impose any constraints on our maximum current available. Figure 4 visually shows how a load cell connects to a HX711 and then to a DevBoard.

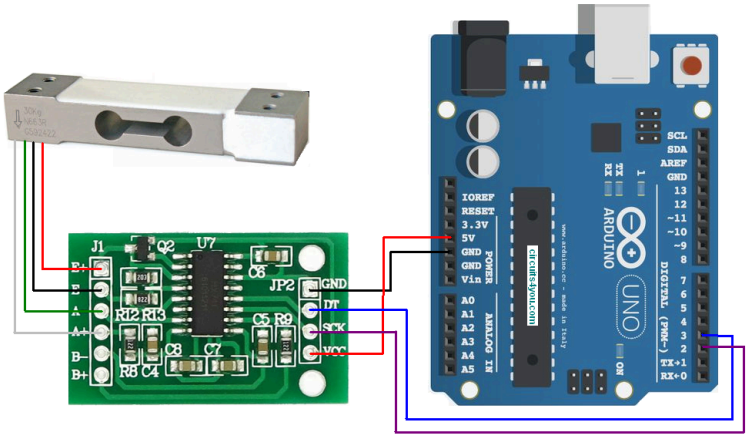

Figure 4: Load Cell + HX711 [6]

Pump: The pump we want to use is a submersible 3V DC water pump. The problem with this pump is basically a motor, so unlike our sensors we can't control what the pump does. We will connect a mosfet that will act as a switch and we will gain control of the pump. [7]

2.2 Subsystem Design and Requirements:

2.2.1 Power Supply Subsystem

The Power Supply Subsystem provides stable operating voltages to all components of the Plant Hydration & Weather Integration System. Five 1.2 V rechargeable batteries [???] will serve as the primary energy source. The water pump will operate at 6 V. Simultaneously, the ESP32 microcontroller and sensors operate at 3.3 V, so a LDO converter regulator will provide a stable 3.3 V output. Decoupling capacitors and similar elements help maintain low ripple and protect against voltage spikes, especially when the pump switches on or off.

Table 1.1: Power Supply Subsystem - Requirements & Verification

Requirements	Verification
1. The subsystem must provide a stable 6 V (±5%) for the pump under peak load conditions.	1. Connect the boost converter output to an oscilloscope (Scopy) with a programmable load simulating the pump's maximum current (eg. 800 mA). 2. Verify the output stays between 5.6 V and 6 V at full load.

	3. Measure ripple to confirm it does not exceed 100 mV p-p.
2. The 3.3 V rail must remain within ±3% of nominal to power the ESP32 and sensors.	1. Supply the LDO from the battery. 2. Measure output voltage at no load and at ~300 mA load with an oscilloscope. 3. Ensure voltage remains in the 3.20-3.40 V range during normal operation.
4. The ESP32 must be able to go to sleep mode when not active to extend the battery's life cycle.	1. Fully charge the battery and run the system. 2. Record time to low-voltage alert. 3. Verify operation for at least 24 hours.

2.2.2 Sensor Subsystem

The Sensor Subsystem collects environmental data, specifically soil moisture levels and rainfall accumulation, that the system uses to make watering decisions. A capacitive soil moisture sensor (SEN0114) is inserted near the plant's root zone and powered at 3.3 V, outputting an analog voltage proportional to volumetric water content. Calibration involves recording sensor output in both dry and fully saturated soil, then mapping the intermediate voltages to a 0-100% moisture scale. Meanwhile, a load cell with an HX711 amplifier detects the weight of rain collected in a

small cup. This high resolution measurement (±1 g or better) allows the system to detect even light rainfall, prompting it to delay or cancel watering events if enough water accumulates. Both sensor readings are fed into the ESP32's ADC or digital input pins at scheduled intervals.

Table 1.2: Sensor Subsystem - Requirements & Verification

Requirements	Verification
1. Soil moisture sensors must measure volumetric water content within ±5% accuracy across 0-100% range.	1. Insert sensor into dry soil and note ADC reading. 2. Saturate soil fully and note ADC reading. 3. Test intermediate moisture levels (25%, 50%, 75%) using a weigh-and-water method. 4. Confirm calibration curve yields ±5% accuracy across repeated trials.
2. Load cell + HX711 must detect rainfall weight changes at ±1 g resolution.	1. Calibrate by placing known masses (1 g, 5 g, 10 g) in the rain cup. 2. Record ADC output and verify it distinguishes each weight within ±1 g.

3. Sensor data must remain stable (±2% drift) over 24 hours in static conditions.	1. Keep the soil sensor in a controlled environment (constant moisture) for 24 hours. 2. Leave the rain cup empty for 24 hours. 3. Log sensor data periodically and confirm drift remains under ±2%.
4. Sensor power consumption must be minimized to support battery longevity.	1. Measure current draw when sensors are actively powered vs. switched off by the ESP32. 2. Confirm that sensor off-state current is below 1 mA.

2.2.3 Control Unit Subsystem

The Control Unit Subsystem consists of the ESP32 microcontroller, local data storage (SD card), and any user interface elements directly attached to the board (status LEDs or control buttons). The ESP32 orchestrates sensor polling, data processing, and pump activation, all while managing Wi-Fi connectivity to retrieve weather forecasts and upload sensor logs to Firebase. A microSD card module, connected via SPI, stores offline data in CSV or JSON format whenever Wi-Fi is unavailable, preventing data loss. The ESP32 also reads battery voltage from the Power Supply Subsystem, logs system events (“pump on,” “rain detected,” “battery low”), and provides real-time or scheduled updates to the external user interface. If desired, an onboard LED or small display can indicate system states, such as charging, watering in progress, or low battery. The

Control Unit thus forms the brain of the system, integrating information from all other subsystems to execute efficient, automated plant care.

Table 1.3: Control Unit Subsystem - Requirements & Verification

Requirements	Verification
1. The ESP32 must poll sensors at configurable intervals and log data locally if Wi-Fi fails.	1. Disable Wi-Fi or simulate a network outage. 2. Verify that the ESP32 continues to read sensors and store logs on the SD card. 3. Re-enable Wi-Fi and confirm that any missed data is eventually uploaded to Firebase.
2. The SD card module must reliably store data (CSV or JSON) without corruption.	1. Operate the system for 24 hours, logging data every ~15 minutes. 2. Inspect the SD card contents for missing or malformed entries.
3. The control unit must provide a user-visible status (LED, display, or button feedback) for at least one system event.	1. Configure an LED to blink when the pump is active or when the battery is below a threshold.

	2. Observe LED behavior during normal operation and confirm that it correctly indicates the event.
4. The MCU must avoid brownouts or resets when the pump activates.	1. Monitor the 3.3 V rail with an oscilloscope while the pump is switched on. 2. Ensure voltage drop is <5% of nominal and the ESP32 does not reset or lock up.

2.2.4 Activation Subsystem

The Activation Subsystem handles the physical watering action through a DC pump and optional solenoid valves. The pump, typically rated for 5 V, draws current through a dedicated driver transistor or MOSFET that isolates high current loads from the microcontroller's GPIO pins. In more sophisticated designs, solenoid valves can divide water flow into multiple zones, each controlled by a separate transistor or relay. The ESP32 sends digital control signals to the driver circuitry, turning the pump or valves on and off according to sensor feedback (soil moisture, rainfall detection) and external weather data. A flyback diode or transient suppression network may be added if the pump or valves are inductive loads. In practice, the subsystem operates for short intervals, commonly a few seconds, to avoid overwatering. The Activation Subsystem's design aims to be robust enough to handle rapid on/off cycling while maintaining system stability and ensuring minimal water leakage or backflow when the pump is inactive.

Table 1.4: Activation Subsystem - Requirements & Verification

Requirements	Verification
1. The pump must deliver a stable flow rate (±10%) at 5 V under typical loads.	1. Supply the pump at 5 V from the boost converter. 2. Measure water flow into a graduated container over 30 s. 3. Repeat multiple times and ensure flow rate is within ±10% of the expected value.
2. The subsystem must minimize water leakage or backflow when the pump is off.	1. Pressurize the tubing by running the pump for 10 s. 2. Deactivate the pump and observe if water continues to flow. 3. If leakage occurs, add check valves or ensure solenoid valves are fully sealed in the closed state.

3. The driver circuit must tolerate pump inrush current without damaging components or resetting the MCU.	1. Monitor MOSFET gate and drain voltages during pump startup using an oscilloscope. 2. Confirm the inrush current remains within MOSFET and wiring limits. 3. Check that the ESP32 supply does not experience excessive voltage sag leading to brownouts.

2.2.5 External Cloud Subsystem

The External Subsystem manages all off-board communication and user interactions beyond the immediate control unit. Primarily, it involves a Wi-Fi connection from the ESP32 to a cloud-based database (like Firebase) and a user facing web interface. The ESP32 periodically requests realtime weather data from an online API, parsing information such as rainfall probability and temperature to refine watering decisions. The system also uploads sensor readings and log entries to Firebase, allowing the user to monitor soil moisture, rainfall accumulation, and battery status from any internet connected device. If the network is unavailable, data is queued locally on the SD card and synced when connectivity resumes. A user website can display real time charts, water usage statistics, and configurable thresholds. This external link thus enables remote monitoring and control, turning the system into a fully integrated, cloud aware plant hydration solution.

Table 2.1: External Subsystem - Requirements & Verification

Requirements	Verification
1. The system must fetch weather data from an API at least every 10 minutes (or a user defined interval).	1. Connect the ESP32 to a known Wi-Fi network. 2. Log timestamps of each successful API call. 3. Verify that the average interval between calls matches the configured schedule (±1 min).
2. Sensor data must be uploaded to Firebase whenever Wi-Fi is available.	1. Simulate a temporary network outage, allowing data to accumulate on the SD card. 2. Reconnect to Wi-Fi and confirm that all pending sensor logs are successfully uploaded.
3. The user must be able to view moisture levels, rain data, and battery status via a web interface.	1. Access the system's web dashboard from a browser or mobile device.

	2. Confirm that displayed data (moisture, rainfall, battery voltage) matches real time or recently logged values.
4. The subsystem must allow user-defined parameters (like moisture thresholds) to be updated remotely.	1. Implement a settings page on the web dashboard that writes updated thresholds to Firebase. 2. Verify that the ESP32 retrieves and applies these new values within a specified period (around 60 s).

2.3 Hardware Design

2.3.1 Operating Voltage & Regulation

Our Plant Hydration and Weather Integration System is powered by four. Because our system components operate at different voltage levels, we have implemented two primary voltage rails. The first is a 6 V rail, generated via the battery, which is connected to a voltage regulator that steps the battery's nominal voltage down to 3.3 V. This rail powers the water pump, which typically requires around 6 V at up to 220 mA during startup. The second is a 3.3 V rail for the ESP32 microcontroller and the soil moisture sensors. A low-dropout (LDO) regulator maintains a stable 3.3 V output. These converters must handle transient events, particularly when the pump switches on, to prevent voltage dips that could reset the microcontroller.

In typical operation, the ESP32 polls soil moisture sensors and the load cell at regular intervals (e.g, every 10-15 minutes). During idle periods, power consumption is relatively low, and the battery voltage remains fairly constant. However, when the pump activates, the current draw on the 5 V rail can spike. To mitigate noise and voltage sag, we have placed decoupling capacitors near the regulator output and at the pump's power input. We also feed the battery voltage to an ADC pin on the ESP32 via a resistor divider, enabling real-time monitoring of battery health.

Stepping down from 6 V to 3.7 V inevitably incurs some conversion losses, leading to our maximum output current of the regulator being 5 A. While the ESP32 at 3.3 V is relatively low-power, the pump can draw 130 - 220 mA when active, creating the most significant load on the battery. To maintain a battery life of 48+ hours under moderate watering schedules, we minimize unnecessary pump activations and put the ESP32 into light-sleep or deep-sleep modes whenever possible.

2.3.2 Water Pump & Sensor Integration

For pump we will be using a DC 2.5-6 V submersible water pump, operating at around 6 V. It is responsible for delivering water to the plants. Its flow rate (typically 100-200 L/h) is sufficient for small gardens or potted plants, and it can be driven by a MOSFET-based driver that switches the 6 V rail under the control of a GPIO pin from the ESP32. A flyback diode or snubber circuit may be added if the pump exhibits inductive characteristics, preventing voltage spikes that could propagate back into the boost converter or microcontroller. We also incorporate a current sense resistor in some prototypes to measure pump current and detect potential blockages or dry-run conditions.

We employ 101020008 soil moisture sensors, which are inserted into the soil at a depth of 5-10 cm to measure volumetric water content (VWC). Each sensor is powered at 3.3 V and outputs an analog voltage proportional to the moisture level. The ESP32 reads these values via its internal ADC, typically with 12-bit resolution. Calibration involves recording sensor outputs in both dry soil and saturated soil, then mapping these readings to a 0-100% moisture scale. Because capacitive sensors are less prone to corrosion than resistive designs, they offer improved longevity for outdoor applications. We do, however, add a protective conformal coating around any exposed metal pads, as repeated exposure to water or fertilizers can degrade sensor performance over time.

2.3.3 Rain Detection Mechanism

In addition to soil moisture sensing, our system employs a Load Cell + HX711 amplifier to detect rainfall. A small, weather-resistant cup sits atop the load cell, collecting precipitation. The HX711, which offers 24-bit differential measurement, interfaces with the ESP32 via a two-wire serial protocol, providing high-resolution weight readings (±1 g or better). If the load cell indicates accumulated rainfall above a certain threshold (eg, 5-10 ml), the system will temporarily halt or reduce pump activation, saving water by letting the plants absorb natural rain. Periodic tare or zero calibration compensates for drift caused by temperature changes or minor mechanical shifts. Additionally, we may filter the load cell output with software algorithms to mitigate noise from wind or small debris.

2.3.4 Mechanical & Environmental Considerations

Because this system is intended for outdoor use, all critical electronics such as the ESP32 board, voltage regulator and load cell amplifier, must be sheltered from direct exposure to rain, extreme temperatures, and debris. Waterproof housings or enclosures with IP65+ ratings are recommended to ensure longevity. The soil moisture sensors are designed for partial burial, but care must be taken to seal any wiring junctions. The load cell should be mounted in a stable position, ideally shielded from strong winds, with a funnel or cup that efficiently collects rain while preventing excessive evaporation or splashing. We also encourage the use of rust-resistant or stainless steel fasteners for any mechanical fixtures.

2.4 Software Design

2.4.1 Irrigation Control, Including Rain Delay and Water Savings Calculation

At the heart of our system is the ESP32 firmware, which runs a finite state machine cycling through states such as SENSOR POLLING, WEATHER FETCH, DECISION EVALUATION, and ACTUATION. Soil moisture data is collected from capacitive sensors at regular intervals (e.g every 10 minutes), while rainfall accumulation is measured using a load cell (via the HX711). The moisture sensor outputs an analog voltage (V_{sensor}), which we map to a percentage using:

$$\text{Moisture } (\%) = \frac{V_{\text{sensor}} - V_{\text{dry}}}{V_{\text{wet}} - V_{\text{dry}}} \times 100$$

Here, (V_{dry}) and (V_{wet}) are the sensor outputs for completely dry and saturated soil, respectively. Meanwhile, the load cell readings W(t) are filtered with a moving average to reduce noise:

$$\overline{W}(t) = \frac{1}{N} \sum_{i=0}^{N-1} W(t-i)$$

Any significant change in weight over a certain timeframe,

$$\Delta W = \overline{W}(t) - \overline{W}(t - \Delta t),$$

indicates recent rainfall, prompting the system to delay watering if natural precipitation is sufficient.

Once all sensor data is acquired, the firmware checks real-time weather data (see Section 2.4.2) and decides whether to WATER or RAIN-DELAY. If the soil moisture is below 30% and the rain probability is less than 70%, the system enters a WATERING state and activates the pump for a duration:

$$t_{\mathrm{pump}} = k \times \big(30\% - \mathrm{Moisture}(\%)\big)$$

where k is an empirically derived constant that translates moisture deficit into pump-on time. Conversely, if there is a high chance of rain or the load cell already indicates significant rainfall, the system transitions to a RAIN-DELAY state. After each watering event, the firmware computes the approximate water savings by comparing soil moisture before and after irrigation:

$$\Delta M = \mathrm{Moisture}_{\mathrm{after}} - \mathrm{Moisture}_{\mathrm{before}}$$

This metric is logged to both the SD card (locally) and Firebase (cloud) for later analysis, allowing users to see the cumulative effect of rain-based water conservation.

2.4.2 Weather API Integration

Our firmware periodically queries a third party weather API (AccuWeather) to obtain parameters. These data points are retrieved via HTTPS GET requests every ~10 minutes, then parsed from the JSON response.

If rain probability (P_{rain}) exceeds 70%, the system defers watering to avoid redundancy. When the API fails or returns incomplete data, the firmware reverts to its last known valid data or relies solely on sensor-based decisions. This modular approach allows for seamless switching between different weather providers and ensures robust operation under varying network conditions.

2.4.3 Firebase Communication & Data Logging

For remote monitoring and historical tracking, our ESP32 firmware sends sensor readings, battery voltage, rainfall amounts, and irrigation events to a Firebase cloud database at regular intervals (typically every 15 minutes). Each data packet is formatted in JSON and timestamped with the ESP32's internal clock. In case of a Wi-Fi outage, the system stores logs on an SD card and later synchronizes them with Firebase once connectivity is restored.

All transmissions use secure HTTPS connections, and we optionally apply checksums to the data payload to guard against corruption. Users can review real-time and historical trends via the web dashboard, which fetches data directly from Firebase. This setup allows them to observe the correlation between weather events, moisture levels, and pump activation over time.

2.4.4 Web UI & User Interface

The web interface, developed using modern frameworks, provides an intuitive dashboard that displays soil moisture trends, rainfall accumulation, battery status, and watering events. Graphs are updated in near-real-time using Firebase's Realtime Database or Firestore, and the UI allows users to adjust system parameters, such as the moisture threshold or watering duration on the fly. Any changes are written back to Firebase, where the ESP32 periodically checks for updates.

Responsive design principles ensure that the interface is accessible from desktops, tablets, and smartphones. Alert notifications are also integrated to warn of low battery, sensor errors, or network failures. By combining user friendly visualization with secure authentication, the system offers both transparency and control, enabling users to fine tune their plant hydration strategy while relying on automated, data driven irrigation decisions.

2.5 Tolerance Analysis

We have identified the rain detection subsystem using a load cell and HX711 ADC as the most challenging component to implement. The main reason for this is the sensitivity of the load cell to environmental factors such as wind, debris, and temperature fluctuations, which can introduce errors in weight measurements. Unlike digital sensors, load cells require precise calibration and may experience drift over time, leading to incorrect readings. The system must differentiate between actual rainfall and other factors that may affect the weight of the cup. **The load cell should detect weight changes as small as 1g to ensure rain is accurately measured.** We will also be manually testing (placing known weights) this sensor routinely.

The load cell outputs analog signals proportional to weight which is then amplified and digitized by the HX711 ADC module then read by the ESP32 microcontroller. We can begin calculating the relative accuracy and resolution of our load cell by analyzing the relationship between rainfall volume and weight:

$$W_{rain} = \rho_{water} V_{water\ collected}$$

Next we can calculate the volume of the water captured in the rain gauge. 1 mm of rainfall corresponds to 1 liter per square meter. For example, a small rain gauge with an area of 10cm^2 would accumulate the following amount of water volume.

$$V_{water\ collected} = 10cm^2 * 0.1cm = 1cm^3$$

With the same rain gauge example, this calculation then leads us to find that the weight of 1 mm of collected rainwater would result as the following:

$$W_{rain} = 1cm^3 * 1g/cm^3 = 1g$$

Therefore, our load cell needs to be able to detect weight changes as small as 1g resolution in order to return reliable measurements of rainfall. Our load cell has a capacity rating of up to 1.5kg and an output sensitivity of ~1mV/V while our HX711 ADC module has a 24-bit resolution. However, the real-world noise and system constraints will likely reduce the actual resolution. The excitation voltage that our system will use is 3.3V. Given this, we can calculate the raw sensor resolution as the following:

$$3.3V/(2^{24}) = 0.197\mu V$$

From here, we can continue on and calculate a theoretical minimum value of the detectable change in weight for the sensor system:

$$\frac{1kg}{3.3*10^{-3}}(0.197\mu V) \approx 0.06g$$

This shows that theoretically, our weight sensor system should be able to detect weight changes smaller than our required 1g resolution, which means that our sensor system is feasible to detect rainfall. It is important to note that we should expect sources of error that may change the feasibility of this sensor system. In the real world, the load sensor is subject to temperature changes, electrical noise, and mechanical stability which all may affect the measured readings from the sensor. In order to ensure the feasibility of the sensor we must manage some techniques for error mitigation such as: periodically calibrating the sensor to eliminate drift, using software to filter out noise, and adding a mechanically sound and stable mount for the sensor to reduce physical disturbances in the sensor's readings.

3 Cost and Schedule

3.1 Cost Analysis:

We can expect a salary of about $38/hr per team member. Accounting for the time it should take us to complete this project and the fact that there are three team members, we can calculate the total labor cost:

$$\$38/hr * 3 * 2.5 * 60 = \$17,100$$

The total cost for the main elements used in this project can be visualized in Table 3. Before shipping, the total cost for parts amounts at least $72 dollars.

Description	**Manufacturer**	**Part Number**	**Quantity**	**Price ($)**	**Link**
Load Cell	SparkFun	SEN-14728	1	12.5	Link
HX711	SparkFun	SEN-13879	1	10.95	Link
Voltage Reg	TexasIns	LM1084-3.3	2	4.98	Link
SDCard Conn	Digilent Inc	410-380	1	7.55	Link
MicroC	Espressif	ESP32-WROOM	1	4.2	Link
Programming Header	SparkFun	CH340C	1	9.95	Link
Soil Sensor	Seed Techn	101020008	3	9.9	Link
Pump	Gifkun	DC 2.5-6V Pump	3	11.98	Link

3.2 Schedule:

The general schedule we are trying to follow for this project can be seen in Table 4. Everything will be assigned according to each member's availability. We have been practicing this method since the beginning of the semester and we have been working well and fairly together. We already own most of the equipment we will be using.

Week	**Objective**
3/10 - 3/16	- Sensor subsystem functioning on our DevBoard - Pump subsystem functioning on our

	DevBoard - Create code for ESP32 to use SD Card - Advance with API.
3/17 - 3/23	- Have a full functioning DevBoard - Work on PCB optimization for third round PCBway Orders.
3/24 - 3/30	- Solder the PCB, debug if necessary. - Try the code from the DevBoard, debug if necessary
3/31 - 4/6	- Work on PCB optimization for fourth round PCBway Orders, if necessary.
4/7 - 4/20	- Find features we could add. - Extra time in case of falling behind
4/20 - 5/7	- Demo - Final Papers/Presentation

4 Ethics & Safety

Our project provides innovative solutions for sustainability and more efficient resource consumption. Our Plant Hydration and Weather Integration System aligns with the IEEE Code of Ethics by prioritizing public welfare, environmental sustainability, data security, and more responsible engineering practices. The main goal of this project is to reduce water waste and innovate solutions for sustainable irrigation. By utilizing soil moisture level monitoring and real-time weather data, our system will help users conserve water usage, aligning closely with IEEE's Code of Ethics Section I.1 emphasizing public safety, health and welfare. Although our system is meant to improve efficiency, we must acknowledge that our system will not entirely remove the need for human oversight as our design is limited by sensor and weather prediction

accuracy/variability. Our commitment to transparency with the user correlates with IEEE's Code of Ethics Section I.5 of maintaining honesty and integrity with the user and our system. It is also important to recognize that our system does store and transmit sensor data using a Firebase cloud, and so we understand the importance of responsible data management. We maintain that our system will avoid unnecessary data collection to respect user privacy and ensure secure user access to prevent unauthorized data manipulation. This further correlates with IEEE's Code of Ethics Section I.1 in relation to the importance of protecting user privacy.

We develop this project with a broad user base in mind. So, we ensure accessibility for various applications such as agricultural, home gardening, or other research purposes. We do not promote or approve of discriminatory practices based on race, gender, disability, or economic status in alignment with IEEE Code of Ethics Section II.7.

There remain a few points of concern in regards to safety which must be addressed as our project is planned to provide solutions to outdoor plant life involving electrical components. Electrical safety will be followed as we are working with an ESP32 microcontroller, lithium batteries, and water-based subsystems. We must ensure that components are properly insulated and waterproofed to prevent the possibility of shorting circuits and other electrical hazards caused by damage from moisture, dirt, and temperature variability. Most notably, our lithium-ion batteries pose a potential fire hazard which requires our group to maintain careful management preventing overcharging, overheating, and thermal runaway. To follow safety guidelines, our system implements a battery management system to monitor the power system ensuring safe charging and discharging cycles and more to prevent potential fire hazards. Mechanical safety will be followed as well. We must take into consideration the stability of the water pump system and the weight sensor system to prevent things such as tipping hazards.

5 References

[1] K. Sentlinger, “Water Scarcity and Agriculture,” *The Water Project*, 2023. https://thewaterproject.org/water-scarcity/water-scarcity-and-agriculture

[2] “Water Risks to Agriculture: Too Little and Too Much | Newsroom,” *Ucmerced.edu*, 2024. https://news.ucmerced.edu/news/2024/water-risks-agriculture-too-little-and-too-much

[3] “Grove -Moisture Sensor.” Accessed: Mar. 03, 2025. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/980/Grove_Moisture_Sensor_Web.pdf

[4] espressif, “ESP32-WROOM-32 Datasheet,” 2023. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

[5] *Ti.com*, 2025. https://www.ti.com/general/docs/suppproductinfo.tsp?distId=10&gotoUrl=https%3A%2F%2Fwww.ti.com%2Flit%2Fgpn%2Flm1084 (accessed Mar. 07, 2025).

[6] “Use load cell setup without tare on each use,” *Arduino Forum*, Jan. 07, 2021. https://forum.arduino.cc/t/use-load-cell-setup-without-tare-on-each-use/690267

[7] Adafruit Industries, “Submersible 3V DC Water Pump with 1 Meter Wire - Horizontal Type,” *Adafruit.com*, 2020. https://www.adafruit.com/product/4546?gQT=1 (accessed Mar. 03, 2025).