ECE 445

Spring 2026

Soilmate

Ysabella Lucero (ylucero2), Emma Hoeger (ehoeger2), and
Sigrior Vauhkonen (sigrior2)
Team Number: #32
Date: 02/27/2026
TA: Zhuchen Shao
Instructor: Arne Fliflet

Table of Contents

Section 1: Introduction

1) Problem……....(4)
2) Solution……(4)
3) Visual Aid…….…....(5)
4) High-Level Requirements…….…....(6)

Section 2: Design

1) Block Diagram…….…(7)
2) Physical Design…….(7)
3) Subsystems and Requirements
    a) App Configuration Subsystem……...…(9)
    b) Sensors Subsystem……..……..(12)
    c) Microcontroller Subsystem……...(15)
    d) Power Subsystem……(18)
4) Tolerance Analysis…….…(19)

Section 3: Cost and Schedule

1. Cost Analysis
    a. Labor…….……(20)
    b. Parts……..…….……(20)
    c. Total…….…….……(22)

2. Schedule……(22)

Section 4: Societal Impact, Standards, Ethics, and Safety

1. Societal Impact……...……(23)
2. Engineering Standards……...……(25)
3. Ethics…….……(26)
4. Safety…….……(27)

Section 4: References……...……...……(29)

Introduction

1. Problem

Many houseplant owners struggle to take proper care of their plants. In fact, “60 percent of millennials said that what worries them most is ensuring plants get the proper amount of sunlight. Other anxieties include: watering (56 percent of respondents), [and] keeping the plants alive (48 percent)” (Castillo). It can be difficult to keep track of when to water them and where to keep them, based on the species of plant and stage of life. Since all plants require water at different frequencies and amounts, it’s also easy to forget to water the plants on time and meet their different schedules. This causes many plants to die, and many owners become disheartened and avoid taking care of plants altogether.

2. Solution

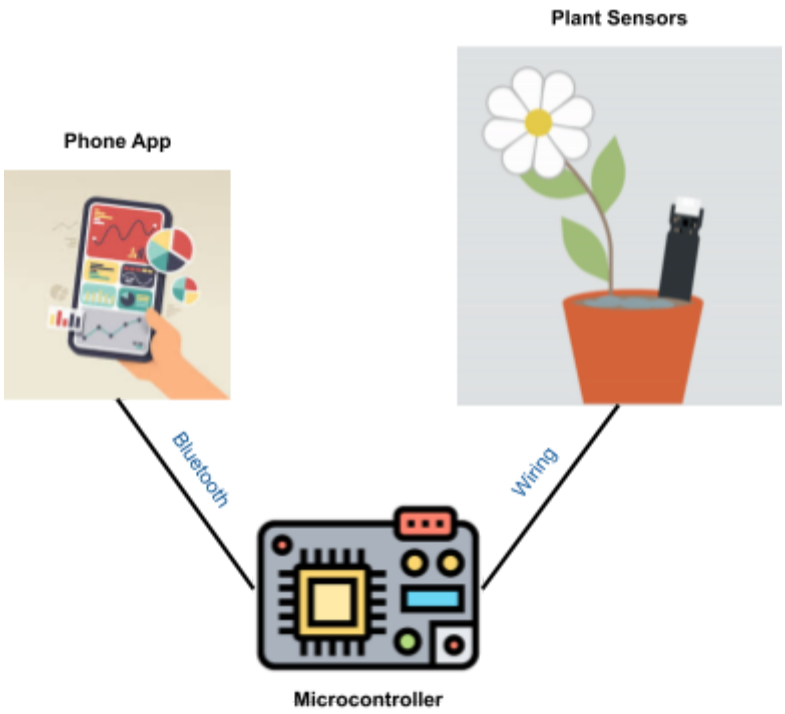
Our solution is to create a notification system to inform houseplant owners of when they should water their different plants. It will also monitor and notify the owner of the conditions of the plant based on various sensors. This will be done by creating an app that the owner can download on their phone, where they will be able to enter their type of plant. There have been many apps created to act as a reminder to water plants; however, the majority of them rely on a schedule rather than live data gathered from the plant. Also, those that do have live data from the plant do not track the weather. Our app will use the location of where a plant is originally from and use the weather patterns in that area to determine when it should be watered (ie. when it’s raining). In addition, there will be a soil moisture sensor, humidity sensor, light sensor, and temperature sensor. The soil moisture sensor acts as a

failsafe to prevent dangerously dry or damp conditions by alerting the owner to water the plant if the moisture is very low, and prevent overwatering of the plant if the moisture is very high. The humidity sensor will alert the owner when the humidity is dangerously too high or too low for the plant, which is especially useful for tropical plants in a non-tropical environment (many houseplants are of a tropical background). The temperature sensor will alert the owner when the room temperature is not in the optimal range for the specific plant. The light sensor will ensure that the plant is in a place with the correct light exposure, and will inform the owner when it is not.

With the integration of software and hardware subsystems, this effective plant notification system will make taking care of houseplants much more convenient for both beginner and experienced plant owners. Beginner plant owners will find it easier to learn about and keep track of the demands of their plants, preventing the most common mistakes that result in their death. Many experienced plant owners can have upwards of 20 plants, and this notification system would make it much simpler to keep track of when to water them all and meet their different needs.

3. Visual Aid

Our design will combine the plant sensors, the user's phone via Bluetooth, and the ESP32 Microcontroller to create this useful tool, which can be seen from our visual aid diagram below.



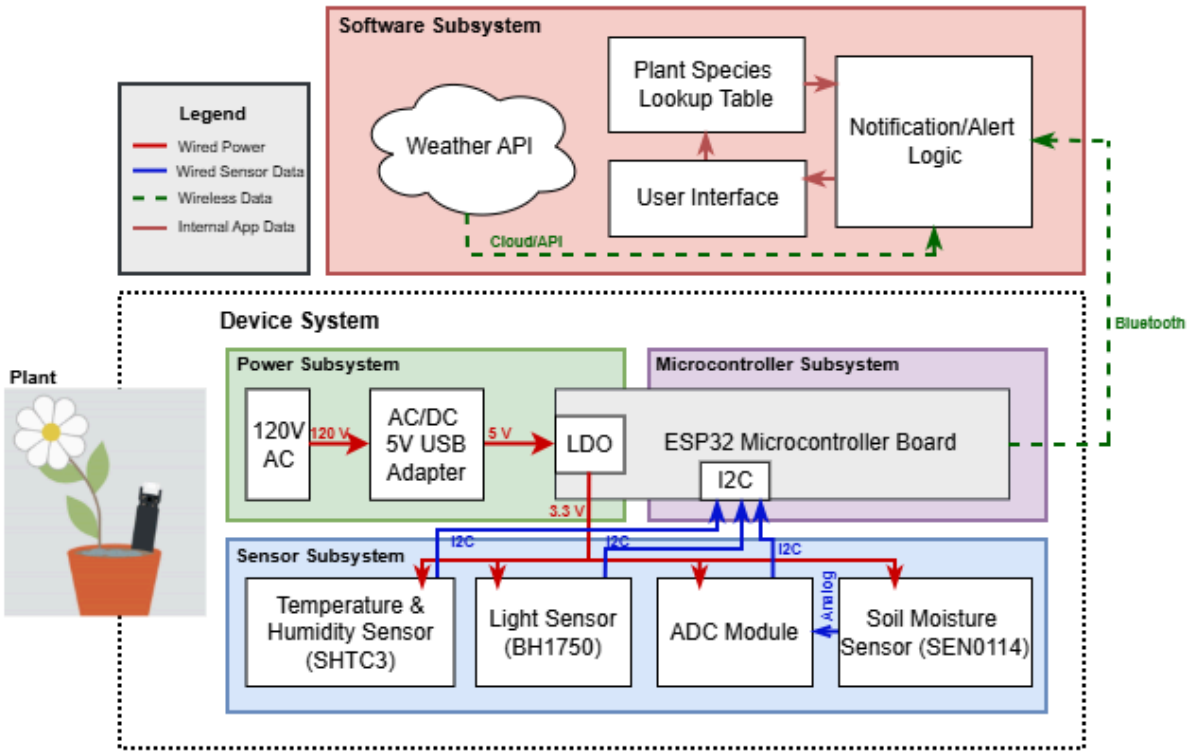
4. High-Level Requirements

1) The ESP32 microcontroller must be able to gather accurate data from the sensors. This will be done by having a PCB with our microcontroller and the majority of the sensors, excluding the soil temperature sensor, since this has to be placed inside the soil. The PCB will be placed near the plant and will gather the environmental data of the plant. It will then communicate the data to the microcontroller, which will transmit it via Bluetooth to the user's phone app.
2) Another basic requirement is to gather weather data from the plant's original region. This will be done using a weather API that will update in the app once a day. If there is a chance of rain, then the app will know when the plant should be watered.
3) Lastly, the app that we create must send notifications to the user when to water the plant or to notify them when the plant's environment must change in combination

with the sensor and weather API (ie. the humidity is too high/low). This will be done using Flutter, a software that is used to create apps.

Design

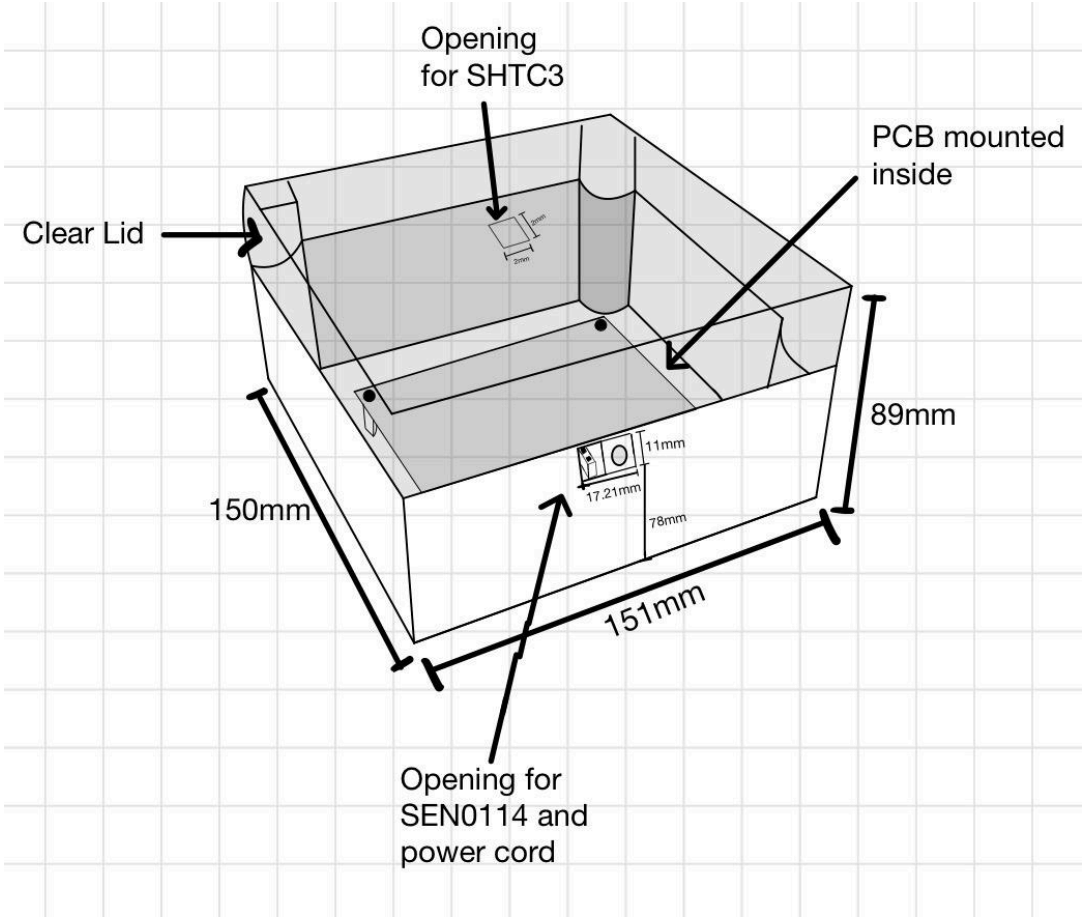
1. Block Diagram



2. Physical Design

Our physical design will consist of a clear waterproof box where our PCB will be kept. This is so it remains protected from the possible water spills that an owner may make when watering the plant. It is especially crucial since it has to be placed in close proximity to the plant because our soil moisture sensor will not physically be on our PCB. The sensor has prongs that

must remain in the soil; therefore we will have a cable that plugs into the sensor which will then be attached to a connector on our PCB. The PCB will have connectors for the sensor, the power cord, and the USB-C. Since these cords must be plugged into the PCB, we will have to cut holes in our box so they can go through. Also, we have two sensors that will be on the board, the humidity/temperature sensor and the light sensor. The light sensor requires us to use a clear plastic box so that it can still detect the light. The humidity/temperature sensor however must be exposed to the open air, in order to gather accurate readings. Because of this, we will have to mount the board close to the top of our clear lid container and create a small opening, ideally only exposing that one sensor. The safety of this design will be discussed in the Societal Impact, Standards, Ethics, and Safety Section. Below is a drawing of our physical design.



3. Subsystem Overview and Requirements

a. App Configuration Subsystem

This subsystem consists of the user interface and the decision engine for our system. It will collect the base environmental measurements of the plant: soil moisture, temperature, humidity, and light, as well as weather conditions from the plant's place of origin. With this data, plant care decisions will be made using pre-existing thresholds for the species of plant.

The app will be implemented using Flutter and Android Studio. There will be four separate types of data to consider. The first is the species lookup table- the app will contain a plant database of predefined data, which contains optimal soil moisture(very dry to very moist), an optimal temperature range (in Celsius), an ideal humidity range (in %Relative Humidity), an ideal light range (dark to bright light), and the origin region of a number of plant species. This data will be obtained from existing data that is well-researched. Next is the added plant data, which is the data that is stored (locally) when a user adds a plant for monitoring. When adding a plant, the user will select the plant species and pot planting date (or the date the soil was last changed). With this information, using the aforementioned lookup table, the app will store the target ranges for soil moisture, temperature, humidity, light, and origin location for the specific plant. In the event that a plant species is unknown to the app (not in the lookup table), the user is able to manually add this information. We also have the sensor data, which is the data received from the sensor device via Bluetooth Low Energy (BLE). This will contain the current soil moisture, temperature, humidity, and light exposure of the plant the sensor device is by. This amount of data is well within

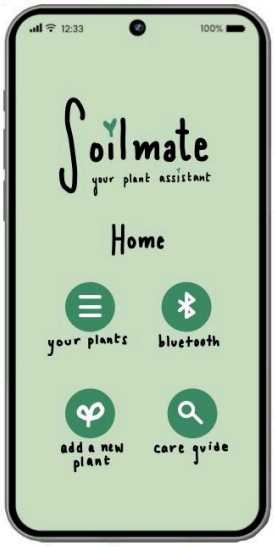
the bandwidth of BLE- each data value will likely be 4 bytes of data (a float), so 16 bytes total. Finally, there is the weather data for each plant's origin location. Once per day, the app will call a weather API (OpenWeatherMap API) using the stored origin location of a plant to obtain precipitation probability.

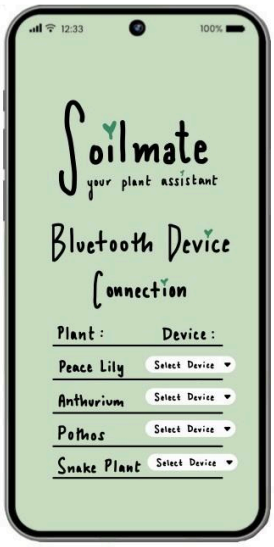
The data contained and collected will be used to make decisions to notify the user. To send a watering notification to the user, the current weather data and soil moisture sensor data will be consulted. There are two cases: the first is if the precipitation probability of the plant's origin region is above a threshold of 50%, AND the soil moisture reading is below a threshold high moisture value, and the second is if the soil moisture reading is below a threshold low moisture value. Essentially, the plant watering notification will be sent if the chance of rain at the origin is over 50%, making sure the soil isn't oversaturated, and if the soil moisture is dangerously low. A notification to lower temperature will be sent if the temperature sensor reads a value above the recommended maximum temperature, and a notification to increase temperature will be sent if the temperature sensor reads a value below the recommended minimum temperature. A notification to increase ventilation will be sent if the humidity sensor reads a value above the recommended maximum humidity, and a notification to mist the plant will be sent if the humidity sensor reads a value below the recommended minimum humidity. A notification to move the plant to a brighter window (perhaps a South-facing window) will be sent if the light sensor reads a value below the recommended light exposure, and a notification to move the plant to a dimmer location will be sent if the light sensor reads a value above the recommended light exposure.

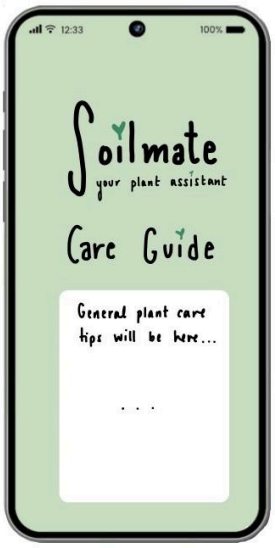
With the data from the soil moisture sensor as a supplementary failsafe, a decision will be made on whether to water the plant or not. If the plant should be watered, a notification will be generated to inform the user. The data from the temperature, light, and humidity sensor will also generate notifications if the temperature and/or humidity is out of the recommended range, informing the user that the environment is too hot or too cold, or too moist or dry. It will give recommendations to either turn down/up the temperature, place the plant in a different facing window (north, east, south, west), mist with water if too dry, or open windows if too humid. This will make the app much more beginner plant owner friendly.

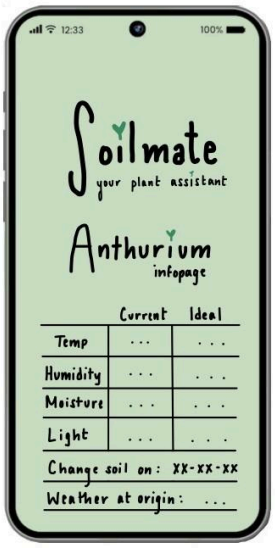
The app will be made up of a home screen that leads to the other pages, a page that allows the user to add a new plant to monitor, a device connection page to connect a specific sensor device to an added plant, a page listing the plants added which leads to infopages for the plants, a care guide page with some general tips on caring for plants, and infopages for each plant that display the current conditions of the plant compared to the ideal conditions.

Below is a graphic that depicts the app design and the aforementioned pages.









Requirements	Verification
• The app must allow user entry of new plants for both plant species existing in the lookup table and plant species that are not	• Test adding a new plant for an existing species and confirm that correct corresponding values are stored in local database • Test the same for a species not listed

• The app must be able to establish and maintain a BLE connection with the ESP32 microcontroller within a room	• Initiate a bluetooth connection to the ESP32 from the app, and measure the time it takes for the connection. Repeat 20 times at various locations within the room and confirm that there are 95% or more successful connections
• The app must be able to store at least 5 different plant profiles	• Add five different plants and verify that the app is able to store separate plant data for all of them
• The app must receive BLE data from the ESP32 microcontroller without corruption	• Send known test values from the ESP32 and confirm that these values match in the app UI
• The app must only send a watering notification when the correct conditions are met, and not otherwise. The app must not be able to send a watering alert more than once a day.	• Insert the soil moisture sensor into bone-dry soil and ensure that the app generates a watering alert • Insert the soil moisture sensor into waterlogged soil and ensure that the app does not generate a watering alert even when it is raining in the origin location • Insert the soil moisture sensor into normal soil within the threshold and ensure that the app generates a watering alert when it is raining in the origin location
• The app must generate environmental alerts within 1 minute of receiving sensor data that would result in a notification	• Measure the time between BLE packet reception and the user alert.

b. Sensors Subsystem

The sensor subsystem will use a resistive moisture sensor (SEN0114), a temperature and humidity sensor (SHTC3), and a light sensor (BH1750). All of these sensors, except the SEN0114 which requires an ADC module, will use an I2C interface that is compatible with our microcontroller (ESP32). The sensors will send their measurements to the microcontroller to be interpreted and relayed through the

app. Our power subsystem will supply the correct voltages to the rated amounts of the sensors.

The I2C interface consists of a SDA and SCL. The SDA is our bidirectional data line that will transfer data from the sensors to the microcontroller. The SCL is the common clock reference that is determined via our microcontroller. The microcontroller has two available I2C pins however we have three sensors. Since it is a bus, we will connect all of our sensors in parallel to the same I2C pin. We determined that this was possible since all three sensors have different addresses dedicated to them so one will not overwrite the other. The SHTC3 sensor will be at x70, the BH1750 sensor will be at x23, and the ADC module that is translating SEN0114 will be at x48.

Also, we will have to include pull-up resistors on the SDA and SCL lines to improve speed and data quality. This is because our sensors can only pull the line low and the high level is created by the pull up resistors. We determined the value of our pull-up resistors, 10kΩ, by referring to the breakout boards for our sensors as well as seeing what is common practice for I2C buses. In our schematic, we have also included a capacitor between SHTC3 power and ground pin to act as a decoupling capacitor. This will stabilize the supply voltage and filter out high frequency noise. Similarly, we have included a decoupling capacitor on the BH1750 sensor for the same reasons. We did not have to make a configuration for the ADC converter since it was purchased on a breakout board.

Requirements	**Verification**
● The SEN0114 must output voltage between 0-3.3 V to represent the	● Using an oscilloscope or DMM we can probe the analog output pin and measure

dampness of the soil	the output voltage ● Ensure that it does not reach above 3.3V, as the sensor can output up to 4.2V ● Ensure that a moist pothos plant will measure 2.48V-2.89V
● The SHTC3 must accurately determine the temperature and humidity of the room with an accuracy of at least ±5%	● We can use an ESP32 devkit or an Arduino from ECE Supply shop or 445 lab materials to connect our I2C, power, and ground pins to and run code on Arduino IDE to interpret the data ● Compare temperature reading to temperature of the room by using the thermostats often found in labs ● Compare humidity reading to the humidity of the room by using
● The BH1750 must accurately determine the brightness of the room in lux with an accuracy of at least ±5%	● We can use an ESP32 devkit or an Arduino from ECE Supply shop or 445 lab materials to connect our I2C, power, and ground pins to and run code on Arduino IDE to interpret the data ● Compare brightness reading to brightness reading of the room using an iPhone app called *Lux Light Meter Pro*
● The ADC module must accurately convert the analog signal to I2C	● We have to get a control reading with only the SEN0114 using the oscilloscope method ● Connect our SEN0114 to the ADC module as well as ESP32devkit or Arduino. Connect our I2C, power, and ground pins to and run code on Arduino IDE to interpret the data ● Compare Arduino IDE data to the control reading from just the SEN0114

c. Microcontroller Subsystem

We must be able to blend our app configuration with our live sensor subsystem to send an alert. We can do this by using the ESP32 microcontroller. It will use bluetooth

connectivity for our sensor devices to easily transfer the data to our app. It is cost-effective and has low power consumption, which will make it easy to integrate with our design. Furthermore, our group has experience with this specific microcontroller, so we are confident in its capabilities.

As described in the previous subsystem, we will only be using one of the microncontrollers I2C pins for the sensor data. We will not need to use any of the other GPIO capabilities such as the SPI or camera interface. We will include decoupling capacitors similar to the sensor subsystem to ensure there is low noise.

Also, we will include a RC circuit on our EN pin, including a 10 kΩ pull-up resistor and capacitor to ground. This is to ensure that there is a stable power-on reset by delaying the enable signal until the 3.3 V supply has stabilized.

We will also include a USB-C connection to the microcontrollers USB D+ and D− pins. This will be crucial for firmware programming and serial communication without the USB-to-UART converter. It is especially convenient since the microcontroller already contains an integrated USB peripheral and bootloader, which makes programming easier. In order to program, we must connect the GPIO0 pin to ground during reset so that the device knows it is in download mode for firmware programming. This is why we will include a simple pin header to the GPIO0 pin so that when we want to program, we short the two pins together which will effectively bring that pin down to ground. Then we will power cycle the unit to reset it, as it will be in download. During normal operation, the pin remains unconnected (internally pulled high), allowing the device to boot from flash memory.

Requirements	Verification
• The ESP32 must power on reliably at 3.3V and execute firmware under normal operating conditions	• Use multimeter to probe voltage rail and ensure that ESP32is receiving 3.3V • Connect USB-C to PC and open Device Manager to ensure a new serial Device will appear, turn ESP32 on and off a couple times to see if device continues to appear • Open Arduino IDE and run basic code that sets up baud rate and runs a loop to constantly print a message
• The ESP32 must enter USB download mode when GPIO0 is grounded during reset	• Put ESP32 in download mode by grounding pin • Connect USB-C to PC and confirm device enumerates as USB serial device in Device Manager. • Open Arduino IDE, set baud rate, and watch console to see if “waiting for download” or “connecting” message pops up • Remove GPIO0 ground and verify normal boot (same verification as requirement above)
• The ESP32 must successfully communicate with all I2C sensors on a shared bus	• Connect USB-C to PC and confirm device enumerates as USB serial device in Device Manager • Open Arduino IDE and run a simple scanner code to ensure it registers that those devices are connected at their respective addresses • Initialize SDA and SCL lines in Arduino IDE and create a TwoWire instance to create the bus. Then initialize a BME280 object with the sensors address and the TwoWire instance to then request data
• The ESP32 must successfully transmit bluetooth data to our mobile app	• Open Arduino IDE and upload firmware that establishes a two-way serial bluetooth communication • Create a loop that sends and receives data

	• Upload code and connect with iphone, should be able to type data from iphone and have it show up in Arduino IDE terminal • Alter code to request data from I2C and send it to iphone. Ensure data appears on iphone
• The ESP32 must remain operational during transmission to phone and gathering sensor data without resetting or shutting down	• With the previous requirements in place, create a loop to continuously pull sensor data at least 100 times and ensure data is expected

d. Power Subsystem

The power subsystem will deliver power to the sensors and microcontroller systems. The ESP32, temperature, humidity, moisture, and light sensors require 3.3V. The 3.3V will come from an external LDO, and we will use a 6V wall adaptor to convert the 120V AC from the bench to 6V. The LDO we will be using is the AP2112K-3.3TRG1 (SOT25) which has a fixed output of 3.3V.

Requirements	Verification
• The LDO must be able to output a voltage of 3.3V to 3.6V to power the ESP32 and all of our sensors	• Using an oscilloscope or DMM we can prode the output pin and measure the output voltage
• The wall adaptor must be able to output a voltage of at least 2.5V and no more than 6.5V, with an accuracy of at least ±5%	• Using an oscilloscope or DMM we can prode the output pin of the barrel jack connector and measure the output voltage

4. Tolerance Analysis

There are some devices within our design that could possibly risk successful completion. The first is that we need to have our soil moisture sensor translate data to our A2D module and then to the microcontroller. Having this module in between the two could create slightly skewed data when it translates analog to I2C. We will be using an ADS1115 for our converter, which provides 16-bit precision at 860 samples/second. The extra bits will allow us to be more precise with the voltage values it reads from the sensor. The module will not sample from the sensor too quickly since the sensor continuously sends out data. The module also allows us to program the adjusting gain if we notice the values varying too much.

Additionally, the soil moisture depth may vary with plants, so there can be inaccuracies in depth. Some plants need to stay consistently moist, while others may need to have the first two inches be dry before it has to be watered again. To mitigate this, we will have our app tell the owner how deep they must insert the sensor when they first get the device.

The last concern is that we will update our weather data only once a day. This can become a problem if there is no rain at that point or there is no chance of rain, yet it randomly rains later in the day. We wanted to update it only once to eliminate giving the owner multiple notifications. This is so that they do not receive conflicting notifications within a quick time span of each other. This is especially a concern in tropical environments where it intermittently rains throughout the day for a short period of time, so it could possibly overwhelm the owner.

Cost and Schedule

1. Cost Analysis

a. Labor

The average starting salary from the ECE department at the University of Illinois at Urbana-Champaign is $90,115 for electrical engineers and $103,222 for computer engineers. If we average these, we get $96,668.5 overall. If we assume a 40 hour work week for 52 weeks a year, we get an hourly rate of $46.48/hour. Since this is a four-credit-hour class, the average time spent outside of class should be about eight hours a week for an approximate 14 weeks. As a result, it should take 112 hours per person to complete the course and 336 hours as a team. Using the formula, ($/hour) x 2.5 x hours to complete, we get a total of $13,014.40 per person and $39,043.2 for the team. We also estimate 3 hours of labor from the machine shop for our physical design. If we estimate an hourly wage of $40/hr for the machine shop, we get a total of $120.

b. Parts

Description & Part Number	**Manufacturer**	**Vendor**	**Quantity**	**Unit Price**	**Total Price**	**Datasheet**
Microcontroller ESP32-S3-WROOM-1	Espressif Systems	Eshop	3	N/A	$0	Datasheet
Moisture Sensor SEN0114	DFRobot	DigiKey	3	$2.70	$8.10	Datasheet
Temperature & Humidity Sensor	Sensirion AG	DigiKey	3	$2.06	$6.18	Datasheet

SHTC3						
Light Sensor BH1750FVI	Rohm Semiconductor	DigiKey	3	$3.29	$9.87	Datasheet
ADC Module ADS1015	Adafruit Industries LLC	DigiKey	3	$9.95	$29.85	Datasheet
6V 1A Power Supply Adapter	Security-01	Amazon	1	$8.99	$8.99	Datasheet
Voltage Regulator LDO-AP2112K-3.3TRG1	Diodes Incorporated	Eshop	4	N/A	$0	Datasheet
PCB Mounting Female DC Power Barrel Jack	California JOS	Amazon	1	$6.99	$6.99	Datasheet
Capacitor - 0.1µF	YAGEO Group	Eshop	8	N/A	$0	Datasheet
Capacitor - 1µF	YAGEO Group	Eshop	8	N/A	$0	Datasheet
Capacitor - 22µF	YAGEO Group	Eshop	8	N/A	$0	Datasheet
3 Position Header Connector PPTC031LFBN-RC	Sullins Connector Solutions	DigiKey	3	$0.30	$0.90	Datasheet
Resistor - 10kΩ	N/A	Eshop	24	N/A	$0	N/A
Resistor - 0Ω	N/A	Eshop	6	N/A	$0	N/A
Tactile Switch PTS645	Littlefuse	Eshop	2	N/A	$0	Datasheet
USB Port USB4500	GCT	DigiKey	2	$0.67	$1.34	Datasheet
Light Sensor-Dev board BH1750	Adafruit Industries LLC	DigiKey	2	$4.50	$9.00	Datasheet
Temperature & Humidity Sensor-Dev	Adafruit Industries LLC	DigiKey	1	$13.95	$13.95	Datasheet

board SHT31-D						
Shipping			2	$6.99	$13.98	
Tax-DigiKey					$0.24	
Tax-DigiKey					$0.81	
Tax-Amazon					$1.48	
Total Cost					**$111.68**	
Remaining					**$38.32**	

c. Total

Team Labor Hours	336
Machine Shop Labor Hours	3
Total Labor Hours	**339**

Team Labor Hour Costs	$39,043.2
Machine Shop Labor Hour Costs	$120
Cost of Parts	$111.68
Total Cost	**$39,274.88**

2. Schedule

Week	Task	Person
February 23	Order all parts	Emma
	Design Document	All
March 2	Test & Verify Sensors	All

	Complete PCB design and order	Yzzi
	Design Review	All
March 9	Bread Board Assembly and Testing	Emma & Yzzi
	Begin App Development	Sigi
Spring Break		
March 23	Begin Soldering	Emma & Yzzi
	Contact Machine Shop for Physical Design	Emma & Yzzi
	Debug App Development	Sigi
March 30	Complete PCB and Test Sensors	Emma & Yzzi
	Test App Development	Sigi
April 6	Progress Demo	All
	Cont. Testing and Revising	All
April 13	Complete App and Program PCB	All
April 20	Mock Demo	All
April 27	Final Demo	All
	Final Presentation	All
May 4	Final Report	All

Societal Impact, Standards, Ethics, and Safety

1. Societal Impact

The motivation for our project stems from a problem in our lives that we all have in common. We are all owners of numerous house plants, but with our busy schedules as ECE students, we do not have enough time to properly care for our plants with their variety of needs. We have all had plants die from things like underwatering, overwatering, not changing the soil, and being exposed to too much sun, all caused from the lack of time we are able to dedicate to our plants.

It is well researched that having house plants provides strong benefits to people by improving indoor environments. They help with both air toxicity: "...the air indoors is as much as 30 times more toxic than the air outside. One solution, borne out by NASA research, is to bring some of the outside in: plants and associated microorganisms in the soil around them are nature's life-support system" (NASA Technology) and also mental health: "caring for plants can help reduce stress and improve mood. Interacting with plants helps suppress sympathetic nervous system activity and reduces diastolic blood pressure…the presence of plants increases concentration, memory, and productivity" (Lang). Many people, especially working people and students, believe they do not have the time, experience, or energy to take care of plants and therefore never end up having indoor plants despite the marked benefit they can make. There are those who get a plant or two and struggle to take care of them, and therefore are discouraged from owning future plants.

Our system would shoulder the mental load of taking care of plants. Keeping track of watering, worrying about light exposure and temperature, and remembering when to change the soil will all be taken care of by the device. Plant owners will simply need to follow the recommendations given by the app to keep their plants healthy. It

makes owning plants much more accessible to people of all backgrounds, whether they are experienced with plants or not, and prevents the unnecessary deaths of houseplants. Plants are important in urban environments for health and mental well-being, and our hope is that with our project, their presence will become much more prevalent. Also, economically, our design consists of widely available and inexpensive components such as the ESP32 microcontroller and common sensors, so the device would be financially accessible to both students and working individuals.

2. Engineering Standards

Our project involves wireless communication (WiFi and BLE), low-voltage power regulation, and electronic hardware. So, several engineering standards are relevant to our design. WiFi communication is not implemented in the hardware of our system- the app uses the smartphone's existing WiFi connection to access the weather API. Therefore, the standard IEEE 802.11 for WiFi communication is handled by the user's smartphone's hardware. The sensor device communicates with the app using Bluetooth Low Energy (BLE), which is defined by the Bluetooth Core Specification maintained by the Bluetooth SIG. Compliance with this specification is largely handled by our microcontroller being an ESP32 module, which is pre-certified to meet the necessary standards. Additionally, the ESP32 operates in the 2.4 GHz ISM band, which is a frequency band allowed for public use as defined by the FCC Part 15 regulations, which govern unlicensed RF devices in the US. We will also make sure to use the standard software libraries for WiFi and Bluetooth when developing the app.

As our system is powered by 120V AC mains, using an adapter, we need to comply with the UL certification standard UL 62368-1 for AC to low voltage DC

adapters. This standard ensures protection against electric shock, proper insulation between high and low voltage output, fire prevention, temperature limits, and safe design so that the high voltage is isolated. The AC/DC adapter we will use to convert 120V to 6V will be UL listed and compliant with this standard, which will make sure that our system is properly isolated from high voltage mains power.

3. Ethics

As we build our project, we will strive to follow the IEEE and ACM codes of ethics while working as a team. We will prioritize honesty and transparency within our group, making sure to maintain communication throughout the project, respect each member's contributions, and professionally resolve conflicts that may come up. We will also be responsible with the reporting of our project, ensuring that we will not falsify results, and will accurately portray the capabilities and potential limitations of our project.

Regarding the app, we are using weather forecasting data that is publicly available through a public API. We are not tracking or collecting the location of the user and using the weather forecast information of their location. Instead of this, the app stores only plant-specific environmental information. As we are not gathering any personal information of the user, there is little risk relating to the security of the user's personal information. Despite the system not handling sensitive information, we will still make sure to responsibly handle data. Any stored plant information will remain local to the app unless explicitly required. For the WiFi connectivity that will be used in API calls, we will make sure to follow the communication protocols defined by the standard libraries that are publicly available. The BLE communication in the system occurs only between

the sensor device and the app, so there is little security risk there as no sensitive information will be sent. Also, we will strive to have our UI for the app be user-friendly. Notifications should be clear and not excessively sent, the text should be in a readable font, and the app should be easy to navigate.

We recognize that inaccurate sensor readings can result in incorrect recommendations that harm the user's plant. We will do our best to make our system as reliable as possible by thoroughly testing all our sensors and understanding the values that we obtain and what conditions they correspond to, rather than relying on the metrics given by the datasheets. We will ensure that we obtain reputable and accurate data for plant health to compare to the sensor data, and if we run into any limitations, they will be clearly stated so that the user understands the extent of the device's capabilities.

4. Safety

Our design has a few aspects that pose a safety risk. To begin with, the PCB we create will not be waterproof. Thus, when the owner goes to water the plant, some drops may spill and possibly cause short circuits and ruin the board. The system operates at low voltage and low current, so the risk of injury is minimal, but it is nevertheless important to prevent damage to the components. Ideally, we would have an entirely waterproof box to enclose the PCB, but this is not possible due to the sensors we are using to monitor the plant. The soil moisture sensor needs to be inserted into the plant's soil, the temperature and humidity sensor must be exposed to the air around the plant, and the light sensor must be visible to measure the light hitting the plant- so we cannot enclose them within a box. To prevent the hazards of water meeting electronics, we are working with the machine shop to create a container for our PCB to ensure that it is minimally exposed,

with a port to wire the soil moisture sensor out of the box and with strategically cut holes to expose just the sensors that need to be visible. We plan to enclose the wires of the moisture sensor with heat shrink, and the only part of the system in contact with the soil will be the probe of this sensor.

Finally, as we develop our PCB, we will follow practices established by the course- we have completed the lab safety course needed for the course, and have learned to solder and use KiCAD with the assignments given. We will use this experience and the knowledge of course staff to make sure our design has proper trace width and spacing and correct placement of components like capacitors and resistors, and we will follow best practices for safety when soldering in the lab.

References

[1] ACM, “ACM Code of Ethics and Professional Conduct,” Association for Computing Machinery, 2018. Accessed: Feb. 24, 2026. [Online]. Available: https://www.acm.org/code-of-ethics

[2] A. Chaturvedi, “How to NOT become a plant murder 101,” Medium. Accessed: Feb. 04, 2026. [Online]. Available: https://avantikachat.medium.com/how-to-not-become-a-plant-murder-101-8d561ba9c3bb

[3] Accessed: Feb. 27, 2026. [Online]. Available: https://docs.arduino.cc/learn/communication/wire/

[4] filo, “Mobile device data statistics phone,” iStock. Accessed: Feb. 04, 2026. [Online]. Available: https://www.istockphoto.com/illustrations/mobile-phone-market

[5] Flaticon. Accessed: Feb. 04, 2026. [Online]. Available: https://www.flaticon.com/free-icon/microcontroller_2752878

[6] IEEE, “IEEE Code of Ethics,” IEEE.org, 2020. Accessed: Feb. 24, 2026. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html

[7] I. D. Castillo, “This Is How Many Houseplants the Average Plant Parent Has Killed,” *Apartment Therapy*, May 11, 2022. Accessed: Feb. 04, 2026. [Online]. Available: https://www.apartmenttherapy.com/dead-houseplants-average-37076429

[8] Lang, Kristine. “The Benefits of Having Houseplants.” *SDSU Extension*, 29 May 2025. Accessed: Feb. 27, 2026. [Online]. Available: extension.sdstate.edu/benefits-having-houseplants.

[9] “NASA Plant Research Offers a Breath of Fresh Air.” *NASA*, NASA, 2019. Accessed: Feb. 27, 2026. [Online]. Available: spinoff.nasa.gov/Spinoff2019/cg_7.html.

[10] pauljoe george, “Setup Arduino IDE to flash a project to ESP32,” Medium. Accessed: Feb. 27, 2026. [Online]. Available: https://medium.com/@pauljoegeorge/setup-arduino-ide-to-flash-a-project-to-esp32-34db014a7e65

[11] Ray, Brian. “Examining 5 IEEE Protocols - Zigbee, WIFI, Bluetooth, Ble, and WiMax.”

IoT For All, Dec. 02, 2024. Accessed: Feb. 13, 2026. [Online]. Available: www.iotforall.com/ieee-protocols-zigbee-wifi-bluetooth-ble-wimax.

[12] S. Santos, “ESP32 I2C Communication: Set Pins, Multiple Bus Interfaces and Peripherals,” Random Nerd Tutorials. Accessed: Feb. 27, 2026. [Online]. Available: https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/

[13] S. Santos, “ESP32 Bluetooth Classic with Arduino IDE - Getting Started,” Random Nerd Tutorials. Accessed: Feb. 27, 2026. [Online]. Available: https://randomnerdtutorials.com/esp32-bluetooth-classic-arduino-ide/