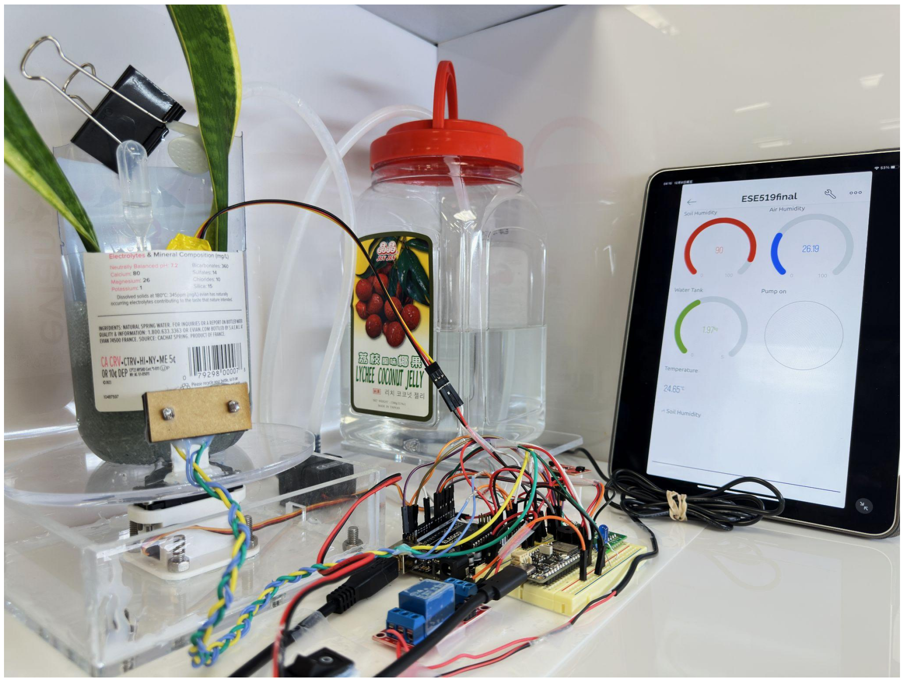
Penn Engineering

ESE5190 Final Project Report

Plant Butler

Team 27

Team members:
Lele Yang, Xueyang Qi, Xuelin Wu



Report due: December 15, 2023, 11:59 PM

Table of Contents

1 Abstract

Plant Butler tackles the prevalent issue of plant maintenance amid hectic schedules. It recognizes the challenge individuals encounter in regularly caring for their plants due to their demanding lifestyles. Plant Butler serves as a smart solution, employing cutting-edge technology to oversee user interface websites, environmental sensing, pump watering, sunlight seeking, camera recording and other essential factors. The result is a practical and user-centric system designed to streamline plant care for those leading busy lives. It is definitely successful that the efficacy of this solution is apparent in its smooth assimilation into various lifestyles, presenting a carefree method for individuals to nurture their plants.

2 Motivation

The issue at hand revolves around the disregard for plant care resulting from hectic schedules, affecting individuals who wish to incorporate greenery into their lives. The project becomes intriguing as it unites technology and nature, presenting a remedy for plant enthusiasts with varying lifestyles. The primary aim is to develop an intelligent plant care system that not only keeps tabs on plant well-being but also enriches the overall user experience. This drive aligns with the burgeoning trend of smart home devices catering to diverse needs.

3 Goals

The goal of the Plant Butler project is to revolutionize plant care by employing a sophisticated system that optimizes various environmental parameters critical for plant health. Through an amalgamation of soil moisture, light, air temperature, and humidity sensors, the project ensures a comprehensive approach to monitoring plants. The data collected from these sensors is processed and communicated to a Microcontroller Unit (MCU) using ADC, I2C, and two-line transmission methods. The processed information is then wirelessly transmitted to a user interface through an ESP32 WiFi module, facilitated by the Blynk app. To provide a visual representation of the plant's growth journey, a camera module connected to the ESP32 captures real-time footage.

The project goes beyond monitoring and incorporates actuators for precise environmental control. A servo motor, controlled by PWM signals, adjusts the light sensor, while a pump automates watering. Safety measures include a transparent small box designed for the servo motor to prevent potential hazards caused by its rotation. In cases of insufficient water storage, a buzzer triggers low water level alerts, ensuring prompt notification to users.

Encompassing diverse engineering topics such as Timers, Interrupts, ADC, and Serial/Wireless Communication, the project showcases sophistication and precision in the realm of smart plant care systems. Plant Butler seamlessly integrates technology to simplify plant care for individuals with various lifestyles, offering a comprehensive and innovative solution for enthusiasts passionate about greenery.

4 Hardware and Software Requirements Specification

4.1 Hardware Requirements Specification:

4.1.1 Overview:

Plant Butler's hardware requirements encompass various components that form the backbone of the smart plant care system. These include sensors, microcontrollers, actuators, and additional hardware for seamless operation.

4.1.2 Definitions, Abbreviations:

- HRS: Hardware Requirement Specification.
- ATmega328P: Microcontroller used as the brain of the system.
- PDV-8001: A light dependent resistor with sensitivity in the visible light region, used for sunlight seeking.
- SHTC3: Humidity and temperature sensor.
- HX711: An amplifier allows you to easily read load cells to measure weight.
- Load Cell: Measures force/load electrically, used as weighting water remaining from the tank.
- ESP32-CAM: A compact camera module equipped with the ESP32-S chip, an OV2640 camera, several GPIOs for peripheral connections, and a microSD card slot for storing captured images or serving files to clients.

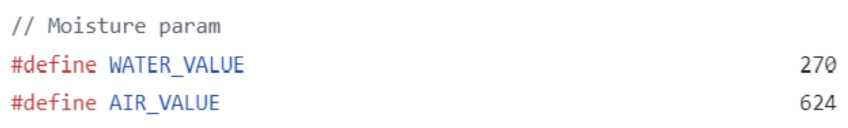
4.1.3 Functionality:

HRS 01 - Microcontroller and ESP32 Module:

- The project shall be based on the ATmega328P microcontroller, serving as the central processing unit for data acquisition and control.
- ESP32 Module shall facilitate wireless communication for seamless integration and remote monitoring and ensure data transmission between hardware and software systems.

HRS 02 - Various Sensor Implementation (Soil Moisture, PDV-8001, SHTC3, HX711, Load Cell-5KG):

- Soil Moisture Sensor: Shall monitor soil moisture levels for efficient plant watering. The calibration should be placed on both water and air each time it is used.

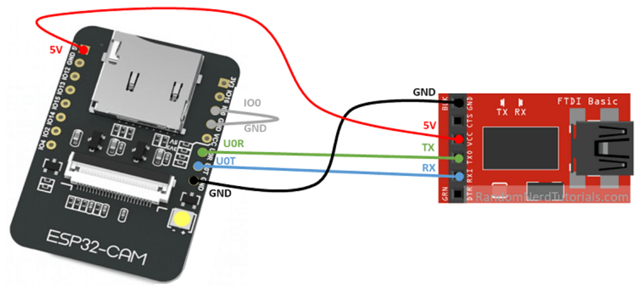


```
// Moisture param
#define WATER_VALUE                                        270
#define AIR_VALUE                                          624
```

- PDV-8001 Sensor: Shall capture real-time plant footage for visual tracking.
- SHTC3 Sensor: Shall measure air temperature and humidity for environmental monitoring.
- HX711 and Load Cell-5KG: Shall quantify and manage water tank weight for precise watering.The calibration should be placed following the 2 steps shown in the Reference[1] each time it is used.

HRS 03 - ESP32-CAM:

- Shall connect the ESP32-CAM board to your computer using an FTDI programmer. Make sure to connect pins by the following picture.



HRS 04 - Actuator Implementation (Servo Motor - MGR996, Water Pump, Active Buzzer):

- Servo motor shall rotate when the sunlight is changing, the rotate angle is 60 degrees, if you want to change the rotate angle, shall not exceed 180 degrees.
- The water pump shall automate watering with an active buzzer ringing for low water level alerts.

HRS 05 - Transparent Box for Motor Safety

- Safety measures include the use of a transparent box for the servo motor, ensuring hazard prevention during rotation.

HRS 06 - Additional Hardware for Setup(e.g., Resistors, Wires, etc.):

- Shall contribute to the overall setup for proper connections(schematic diagram shows in Appendix B) and functionality and ensure reliable operation of the system.
- The switch in 'Watering System' shall not be ignored for safety reasons. Always check the relay board is working properly before turning on the pump[5].

4.2 Software Requirements Specification:

4.2.1 Overview:

Plant Butler, a Smart Plant Care System, relies on a robust software infrastructure to ensure seamless monitoring and control of plant health indicators. The software components play a crucial role in data reception, processing, and user interface interactions.

4.2.2 Users:

The software caters to users with varying levels of technical proficiency, including plant enthusiasts, students, and professionals. It is designed to offer a user-friendly experience for efficient plant care management.

4.2.3 Definitions, Abbreviations:

- IDE: Integrated Development Environment
- UART: Universal Asynchronous Receiver-Transmitter
- I2C: Inter-Integrated Circuit

4.2.4 Functionality:

SRS 01 - Sensor Data Management:

- Shall install Arduino IDE and set up the ESP32 board manager[2][3].
- Shall ensure all dependent libraries are installed.
- Should implement UART to ensure a reliable serial communication protocol between the Arduino UNO and the ESP32. There are some other communications you can choose, for example, SPI, I2C.
- Should use 'UART_print(String)' the following function to check the real-time data from Ardunio before sending data to ESP32. Following shows an example of sensor collected data print.

```
dtostrf(shtc3_humanity,3,2,printbuff);
sprintf(String,"Humanity is: %s\n\r",printbuff);
UART_print(String);
```

- Shall ensure esp32 receive data by soil moisture, SHTC3, HX711 and Load Cell-5KG sensors as a correct order by programming logic. The following print function should be used.

```
// Print the values
Serial.print("Soil Humidity: ");
Serial.println(SoilHumidity);
Serial.print("Water Tank: ");
Serial.println(WaterTank);
Serial.print("AirHumidity: ");
Serial.println(AirHumidity);
Serial.print("AirTemp: ");
Serial.println(AirTemp);
Serial.print("PumpStatus: ");
Serial.println(PumpStatus);
```

SRS 02 - Camera Module Interface:

- Shall go to File > Examples > ESP32 > Camera and open the CameraWebServer example.
- Should log in your network with SSID and password:

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

- Shall select the correct camera module. Here we should use the AI-THINKER Model.
- Shall click tools and then Board to select AI-Thinker ESP32-CAM by choosing your usb COM port.
- After uploading the code, you should open the serial monitor at a baud rate of 115200, you should see an IP address which is your ESP32-CAM IP address[4].

SRS 03 - Blynk App Integration:

- Shall install a Blynk app to customize your personal webpage like lab 4 did[6].
- Shall include your own Wi-Fi credentials.

```
// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "XXXXX";
char pass[] = "XXXXXXXXXXX";
```

- Shall modify the Template ID with yours.

```
/* Fill-in your Template ID (only if using Blynk.Cloud) */
#define BLYNK_TEMPLATE_ID "XXXXXXXX"
#define BLYNK_TEMPLATE_NAME "XXXXXXXX"
#define BLYNK_AUTH_TOKEN "XXXXXXXXXXXXXXXXXXXXXX"
```

- Should receive real-time data from ESP32 to Blynk through wireless transmission.
- Shall set the Data type as 'Double', except 'Pump on' should be integer.
- Should use' Gauge' shape, 'Label', 'LED', 'Chart' in the dashboard. The page should be managed as pictures shows in Appendix B.
- Shall get the percentage of Soil Humidity, Air Humidity, Water Tank weight, Pump status, Air Temperature, the trends of Soil Humidity.

5 Verification

5.1 Hardware Verification:

Requirement No.	Test method	Result (Pass/Fail)
HRS 01	Check data acquisition on ATmega328P; Ensure wireless communication on ESP32.	Pass
HRS 02	Test each sensor individually for accurate readings; Verify inter-sensor communication.	Pass
HRS 03	Connect ESP32-CAM to a computer using an FTDI programmer; Confirm proper pin connections.	Pass
HRS 04	Test servo motor rotation; Verify water pump automation and low water level alert with active buzzer.	Pass
HRS 05	Confirm transparent box for servo motor safety; Test rotation without hazards.	Pass
HRS 06	Check proper setup using resistors, wires, etc.; Ensure reliable system operation.	Pass

5.2 Software Verification:

Requirement No.	Test method	Result (Pass/Fail)
SRS 01	Validate accurate data management from all sensors; Confirm real-time monitoring.	Pass
SRS 02	Check web interface accessibility; Confirm real-time footage display.	Partially Pass
SRS 03	Test Blynk app control functions; Confirm monitoring and variable control through the app.	Pass

6 Results Summary

In hardware verification, all tests were successful, affirming the robust functionality of the system. Data acquisition on ATmega328P and wireless communication on ESP32 demonstrated seamless integration. Individual sensor testing and inter-sensor communication yielded accurate readings and effective communication. ESP32-CAM connection to a computer using an FTDI programmer was flawless, and servo motor rotation, water pump automation, and low water level alert with an active buzzer operated as intended. The transparent box for servo motor safety ensured safe rotations, and the overall system setup using resistors and wires proved reliable. However, in software verification, while the camera functioned well, occasional connectivity issues with the webpage led to instability during web interface accessibility and real-time footage display tests, indicating a need for refinement in ensuring consistent performance in these areas.

7 Conclusion

- **What we learned?**

  The project provided a valuable learning experience for the team members. It allowed us to integrate various engineering topics such as Timers, Interrupts, ADC, and Serial/Wireless Communication into a real-world application. Working on both hardware and software aspects enhanced our understanding of system integration.

- **What went well?**

  In the hardware verification process, several aspects demonstrated successful outcomes. The data acquisition on ATmega328P and wireless communication on ESP32 were seamless, showcasing effective integration. Individual sensor testing and inter-sensor communication were successful, ensuring accurate sensor readings and smooth communication between components. The connection of ESP32-CAM to a computer using an FTDI programmer exhibited proper pin connections. Additionally, servo motor rotation, water pump automation, and low water level alert with an active buzzer functioned as intended. The transparent box for servo motor safety proved effective, and the overall system setup, including resistors and wires, contributed to a reliable operation. These successful outcomes highlight the robustness of the hardware components and their cohesive functioning within the system.

In the software verification process, notable successes were achieved as well. Accurate data management from all sensors and real-time monitoring were validated, attesting to the effectiveness of the software in handling and displaying sensor data. The Blynk app control functions demonstrated successful testing, confirming the software's capability to monitor and control variables through the app. These achievements in software functionality contribute to the overall success of the system, providing a solid foundation for user interaction and data management. Despite challenges in web interface accessibility and real-time footage display, the software exhibited proficiency in essential tasks, marking significant accomplishments in meeting the specified requirements.

- **What accomplishments?**

Our system embodies essential comprehensiveness. It not only considers dynamic monitoring of plant growth through a camera but also provides users with real-time data updates on soil humidity, light, air temperature and humidity, and the operational status of the water pump via a web interface. This results in the creation of a cutting-edge and user-centric smart plant care system, addressing various aspects of plant health monitoring and care.

The design of the water pump alarm system is particularly notable, significantly assisting users in avoiding the issue of forgetting to timely replace water in the tank. Additionally, the design of a separate button-style switch effectively prevents the possibility of pump malfunction leading to potential system flooding. Both of these features contribute to a safer and more reliable plant care experience for users. Furthermore, our designed automatic light tracking function maximizes the effective absorption of sunlight by plants, promoting robust and healthy growth.

- **What changed**

We did not change anything that we targeted. A change is the method to connect ESP32 with the Wi-Fi module. We were supposed to use the Arduino cloud to present the user interface. But because the Arduino board we used without the Wi-Fi module, we changed to using Blynk.

- **Obstacles**

Several obstacles were encountered during the verification process. Firstly, the servo faced limitations in free rotation due to the design constraints of the protector, hindering its intended functionality. Additionally, challenges arose in achieving real-time data transmission, as noticeable delays were observed in the data flow. The difficulty in establishing a one-to-one correspondence with the packaged data presented a significant hurdle, making precise unpacking a challenging task. These obstacles highlight areas for improvement in the design and operational aspects of the system, necessitating further attention to enhance the servo's range of motion, minimize data transmission delays, and streamline the data packaging and unpacking processes for more precise and efficient functionality.

- **Next Step:**

  Enhanced User Interaction: Implement additional features in the Blynk app for more control over environmental parameters, allowing users to customize plant care based on specific plant types.

  Machine Learning Integration: Explore the possibility of incorporating machine learning algorithms for more advanced plant health predictions and personalized care recommendations.
  Energy Efficiency: Investigate methods to optimize power consumption, potentially incorporating sleep modes for components to conserve energy.
  Expand Sensor Variety: Consider integrating additional sensors for more nuanced environmental monitoring, such as nutrient levels or atmospheric composition.
  Community Engagement: Develop a community or social aspect where users can share their plant care experiences, seek advice, and participate in a collaborative green community.

8 References

[1]https://davidegironi.blogspot.com/2019/04/hx711-load-cell-amplifier-driver-for.html

[2]https://www.arduino.cc/en/software

[3]https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

[4]https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/

[5]https://arduinogetstarted.com/tutorials/arduino-controls-pump

[6]https://canvas.upenn.edu/courses/1741662/files/folder/ESP32_Blynk_Instructions

9 Links

https://devpost.com/software/plant-butler

https://youtu.be/8zYo-L7o6YI?si=YeV9Bh4Qg3DuC6gV

https://github.com/LouisQi/ESE519_Final_Project

https://docs.google.com/spreadsheets/d/174PPIj6tu58mIZMidzgqJUK8YjlKUO3FxPmdxCbUoFw/edit#gid=0

10 Appendix A

10.1 Camera library

CameraWebServer.ino:

```
#include "esp_camera.h":
```

The code includes the necessary libraries: "esp_camera.h" for camera handling and configuration and "<WiFi.h>" for wireless connectivity.

```
camera_config_t config;
//Store configuration parameters for pin assignments, frame size, pixel format,
JPEG quality.
esp_err_t err = esp_camera_init(&config);
//Initializes the camera module based on the provided configuration.
sensor_t *s = esp_camera_sensor_get();
s->set_vflip(s, 1);
s->set_brightness(s, 1);
s->set_saturation(s, -2);
//Specific sensor settings, such as vertical flip, brightness, and saturation.
config.fb_location = CAMERA_FB_IN_PSRAM;
config.fb_count = 1;
//Specific the location and count of frame buffers
if(psramFound()){
    // PSRAM is available, adjust configuration accordingly
} else {
    // PSRAM is not available, limit frame size and use DRAM
}
WiFi.begin(ssid, password);
//Interfaces with the WiFi library to establish a wireless connection.
startCameraServer();
//Initializes and starts the HTTP servers, allowing remote control and
interaction with the camera.
```

app_httpd.cpp file:

The file initialized a HTTP server on the ESP32, allowing remote control of the camera via HTTP. The code also initializes the camera with specified settings such as gain, scale, and offset. In addition, the code offers a range of camera control functions, such as setting the resolution, adjusting brightness, contrast, saturation, etc. Also, the code supports image capture and streaming. It can capture still images in BMP format and streaming video in MJPEG format.

camera_index.h file:

The file appears to be a compressed HTML file (index_ov2640.html.gz) represented as an array of bytes (index_ov2640_html_gz). This file is intended for serving web pages, likely in the context of a web server running on an ESP32 device with a camera module.

camera_pins.h file:

This file segment provides a flexible configuration for different ESP32 camera models, allowing easy adaptation of GPIO pins based on the selected camera hardware. It is a crucial part of initializing the camera module and ensuring correct communication with the ESP32 device.

10.2 HX711 library

```
//Return the raw reading. The function waits for HX711 to become available
to read and performs a 24-bit read operation. The read data is adjusted and
returned as a 32-bit integer.
extern int32_t hx711_read();
//Return the average of the reading in "times"
extern int32_t hx711_readaverage(uint8_t times);
//Return the current reading from HX711 with tare
extern double hx711_readwithtare();
//Convert the raw readings with tare to the weight
extern double hx711_getweight();
//Set the gain
extern void hx711_setgain(uint16_t gain);
//Return the current gain
extern uint16_t hx711_getgain();
//Set the scale factor used to convert the raw ADC value to a weight unit.
extern void hx711_setscale(double scale);
//Return the current scale factor
extern double hx711_getscale();
//Set current reading as an offset value
extern void hx711_setoffset(int32_t offset);
//Return the current offset
extern int32_t hx711_getoffset();
//Set current reading as an offset value
extern void hx711_taretozero();
//Set HX711 to a low power mode
extern void hx711_powerdown();
```

```
//Wake HX711
extern void hx711_powerup();
//A calibration step to set the offset for zero weight (tare).
extern void hx711_calibrate1setoffset();
//A calibration step to set the scale factor, typically done by placing a
known weight on the sensor and adjusting the scale factor accordingly.
extern void hx711_calibrate2setscale(double weight);
//Initialize SCK as an output port with low logical level and DT as an
input. Set calibrated "gain", "scale", and "offset" parameters.
extern void hx711_init(uint8_t gain, double scale, int32_t offset);
```

The HX711 uses a two-wire serial interface for communication, which includes clock line (SCK) and data line (DT).

Initially, the MCU checks if the DT line is low, indicating that the HX711 has completed an A/D conversion and the data is ready to be read. Next, the MCU generates a clock signal on the SCK line. HX711 sends its measurements to the MCU at each rising edge of the pulse. The data is read in a 24-bit format (2's complement format), with the Most Significant Bit (MSB) first. This is done in the loop where 24 clock pulses are sent, and the bits are read and compiled into a 32-bit integer. After reading from the HX711, additional pulses are sent to set the gain and select the channel for the next reading. The read value is manipulated to convert to an integer format. Finally, the converted integer will be averaged, scaled and tared according to user calibration.

10.3 SHTC3 library (combined with I2c library)

```
//Set up the sensor by checking its availability on the I2C bus and storing
its address in the device structure.
SHTC3_RET_t shtc3_init(SHTC3_t* dev, uint8_t address);
//Send a soft reset command to the sensor
SHTC3_RET_t shtc3_reset(SHTC3_t* dev);
//Read the sensor's ID, useful for confirming communication with the
sensor.
SHTC3_RET_t shtc3_get_id(SHTC3_t* dev, uint16_t *id);
//Manage the sensor's power state by putting it into a low-power sleep mode
or waking it up for measurement.
SHTC3_RET_t shtc3_sleep_enable(SHTC3_t* dev);
SHTC3_RET_t shtc3_sleep_disable(SHTC3_t* dev);
//Fetch temperature and humidity readings from the sensor. It can read both
values simultaneously or individually.
SHTC3_RET_t shtc3_get_temperature(SHTC3_t* dev, float* temperature, uint8_t
lp_en);
```

```
SHTC3_RET_t shtc3_get_humidity(SHTC3_t* dev, float* humidity, uint8_t
lp_en);
SHTC3_RET_t shtc3_get_temperature_humidity(SHTC3_t* dev, float*
temperature, float* humidity, uint8_t lp_en);

/* Composed functions */
// start: init + sleep
SHTC3_RET_t shtc3_start(SHTC3_t* dev, uint8_t address);
// read: wakeup + measure + sleep
SHTC3_RET_t shtc3_read(SHTC3_t* dev, float* temperature, float* humidity,
uint8_t lp_en);
```

The SHTC3 sensor employs the I2C communication protocol. Each device on the I2C bus is identified by a unique address, and for the SHTC3, this address is set to 0x70. The communication process begins when the microcontroller unit (MCU), acting as the master device, initiates interaction by transmitting the SHTC3's I2C address along with a write instruction. Initially, the MCU issues a wakeup command to bring the SHTC3 sensor out of its low-power sleep mode, preparing it for active operations. Following this, a measurement command is sent. Subsequently, the MCU readdresses the SHTC3, this time with a read instruction, and awaits an acknowledgment (ACK) signal from the sensor. Once the ACK is received, the MCU proceeds to sequentially collect the temperature and humidity data provided by the SHTC3 sensor.

11 Appendix B

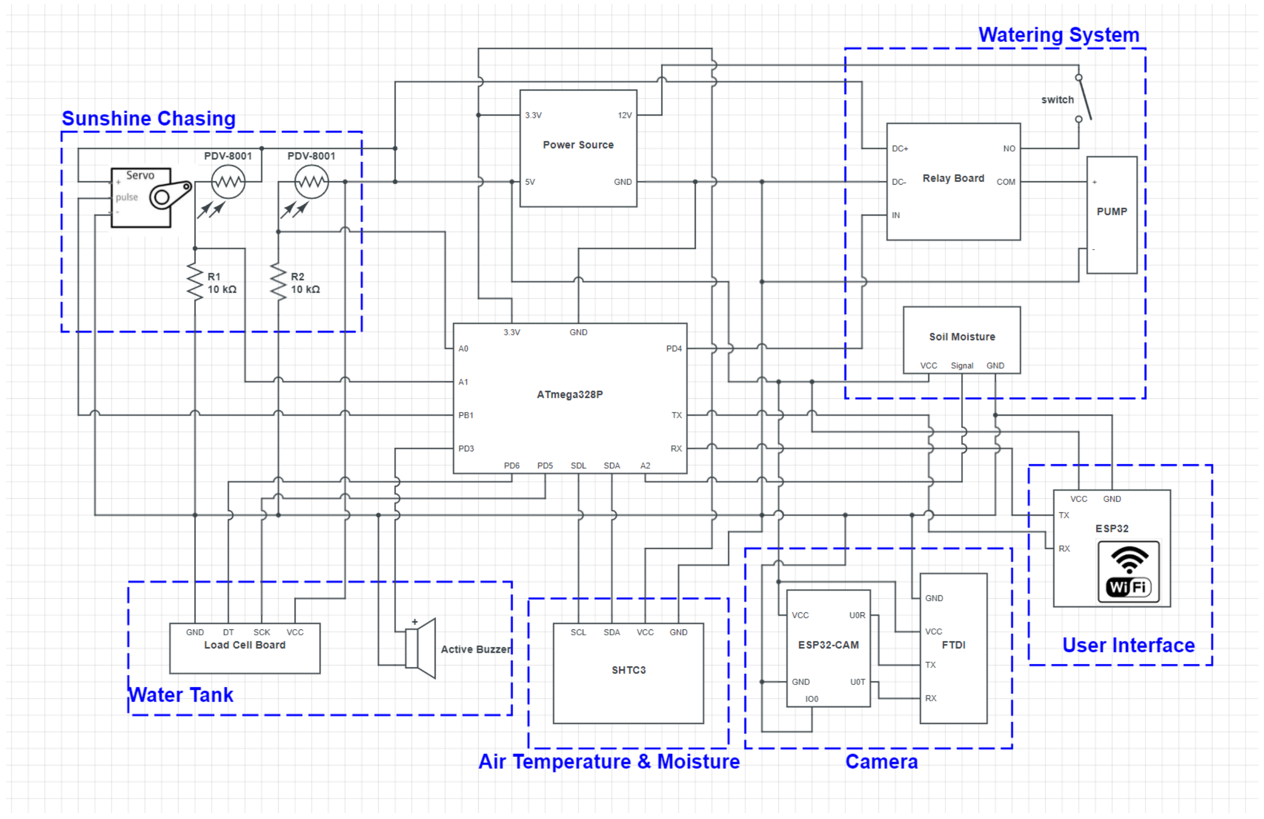
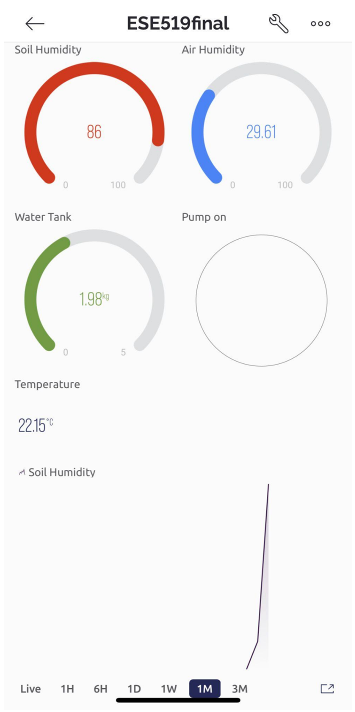
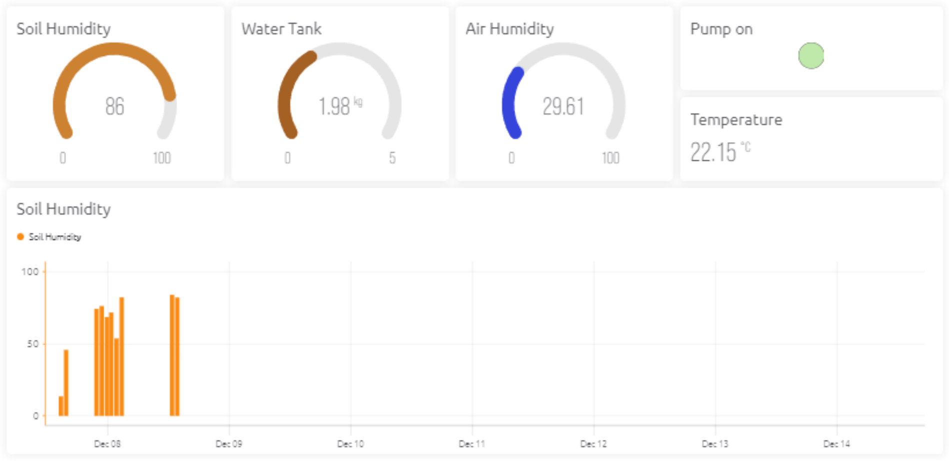


Figure of Overall Schematic Diagram





Figures of Blynk Dashboard management