**AES Documentation**

*Team D*

*16/05/2023*

***Important Notes:***

* ***The descriptions in italics in this document (except for some section headings) are exemplary and explanatory and must be removed from the completed report.***
* ***Identify which section of this report was created by which team member***
* ***Your documentation should have ca. 8 pages.***

# Team members

* *Ajay Paul*
* *Doluwamu Taiwo Kuye*
* *MOHAMMAD ASHRAFUZZAMAN SIDDIQI*

# Introduction

The term "Internet of Things" (IoT) in our project domain refers to the concept of linking various devices and sensors in our smart garden system to the internet. We enable them to communicate and exchange data with one another, allowing for continuous monitoring and control of the garden.

Our smart garden system includes a "Wireless Sensor Network" (WSN). It entails placing wireless sensors throughout the garden to capture vital information like as temperature, humidity, soil moisture, and light levels. These sensors communicate wirelessly, producing a network that sends data collected to a central hub or gateway for processing.

The smart garden system, the project's intended application, would use an Arduino Uno WiFi Rev 2 and a DHT11 temperature and humidity sensor to post sensor readings to a MQTT broker located on a Raspberry Pi. This MQTT broker serves as a central point for sensor data collection and distribution. We can remotely monitor the temperature and humidity levels in our garden by connecting the Arduino to the MQTT broker from any device that is linked to the broker.

This smart garden system can be used for a variety of purposes. It, for example, enables us to monitor environmental variables in real time, allowing us to make data-driven decisions about watering, lighting, and assuring optimal growing conditions for our plants. It also allows us to receive warnings and messages when any environmental parameters fall outside of the required range, allowing us to intervene in time to safeguard the plants.

Overall, incorporating IoT and WSN technologies into our smart garden system improves its efficiency, convenience, and productivity, providing a seamless gardening experience while supporting sustainable practices through data-driven insights.

# Concept description

**Smart Garden Project Motivation**

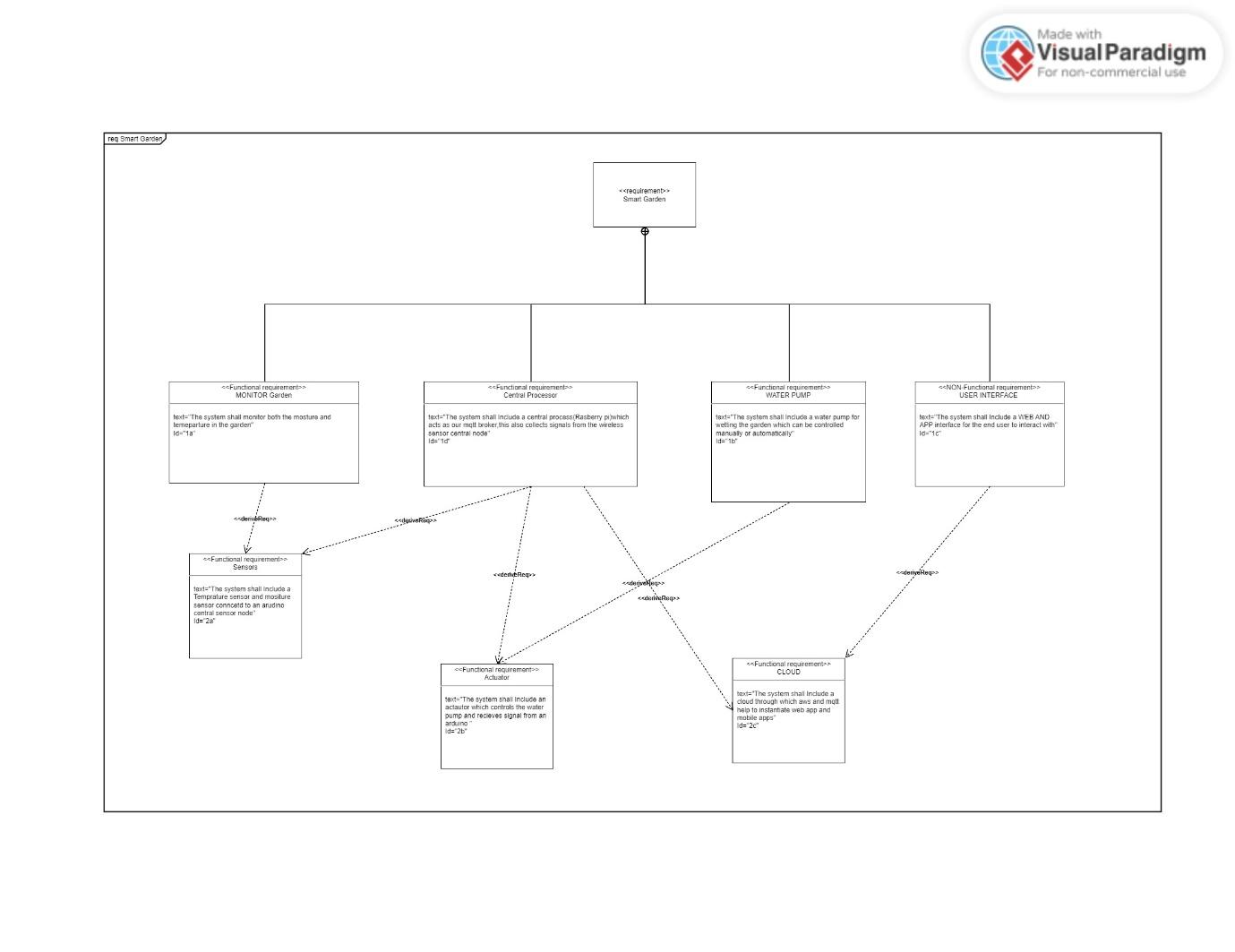
Imagine that you are a busy professional who enjoys gardening but finds it challenging to find the time to give your plants the care they require. You have a variety of plants in your garden, including veggies, herbs, and flowers, and you want to ensure that they are all getting enough water and thriving in the best conditions possible.

You can use your smart garden system to install sensors that monitor the moisture and temperature levels in your garden. These sensors can be linked to an Arduino Uno WiFi Rev 2, which feeds data to a Raspberry Pi 3 that serves as a MQTT broker. The data from the sensors can be used to regulate your water pump, which is also connected to the Raspberry Pi. If the moisture level is too low, for example, the pump can be activated to water the plants.

You may monitor the conditions in your garden in real time using an app on your phone or tablet. You can be notified if the moisture level falls too low or the temperature becomes too hot or cold. You can also create watering plans for your plants to ensure that they get the proper amount of water at the right time of day.

Overall, your smart garden system can save you time and energy while keeping your plants healthy and thriving. It can also provide you with piece of mind because you can watch and control your garden from anywhere at any time.

**Smart Garden Requirement Diagram.**



**Figure 1 Requirement Diagram for Smart Garden.**

# Project/Team management

Our project management style was the Adaptive or Flexible Management approach:

This allows for flexibility and adaptation to changing project requirements and circumstances. We structured task according to areas of specialisation and interest.

This Encourages frequent communication, collaboration, and quick decision-making.

Ideally, we all contributed to every aspect but due to unlimited hardware we took turns in implementing different parts of the system, we also made sure that every member of the group understood what was being done in each level of development.

**Some of the Tasks were:**

* Initial High-level design and Concept Drafting for Smart Garden idea.
* Sensor configuration with Arduino Wi-Fi.
* Mosquitto Installation.
* Raspberry pi Configuration and set up as MQTT Broker.
* MQTT Set up (Publisher and Subscriber).
* IOT MQTT APP Set up and configuration of water pump switch for the Actuator Arduino.
* Think speak Cloud Set up and configuration.
* Wiring of system.
* Overall system integration.

***MOHAMMAD ASHRAFUZZAMAN SIDDIQI:***

* Concept draft
* UML diagrams
* Think-speak Cloud
* Sensor configuration
* Mosquitto installation

***Doluwamu Taiwo Kuye:***

* Arduino wifi setup
* Concept draft
* Uml diagrams
* Block Diagrams
* Sensor node set up
* Subscriber Arduino code (Actuator)
* Wiring of system.
* Configuration of actuators and testing.
* Sensor data algorithm
* IOT mqtt panel setup and additional functionality
* MQTT integration for the overall system
* Thinkspeak cloud code for Raspberry pi
* Water\_pump logic
* Raspberry pi
* Testing and Validation.
* Schematic design.

***Ajay Paul:***

* raspbian os install and Setup SSH connection
* Setup the Mosquitto
* Setup the MQTT Broker and Client
* Setup the config file for external client connection
* Testing with different clients for MQTT setup
* Setup the Arduino for sensor node publisher
* Sensor subscriber for testing in Python
* Actuator Algorithm
* Actuator node implementation
* Testing of the sensor nodes and updations

*Describe which team member did which tasks.*

# Technologies

The Project involved mostly IOT technologies, we used various sensors, sensor nodes and communication protocol.

**MQTT (Message Queuing Telemetry Transport)**: is a lightweight messaging protocol used in our smart garden project to enable efficient and reliable communication between the Raspberry Pi, Arduino, and the cloud platform, allowing seamless data transfer and control commands in a publish-subscribe model. We utilized this protocol to communicate between different components of our system, sending and receiving data simultaneously.

**QOS:** We decided to use the QOS level 0 at most once due to the overhead and the constant reading of data. Since the data is not completely critical, we decided to send it just once and keep sending at close intervals. QoS level 0 is suitable for scenarios where occasional message loss is acceptable, such as sensor readings that can be easily replaced by new data.

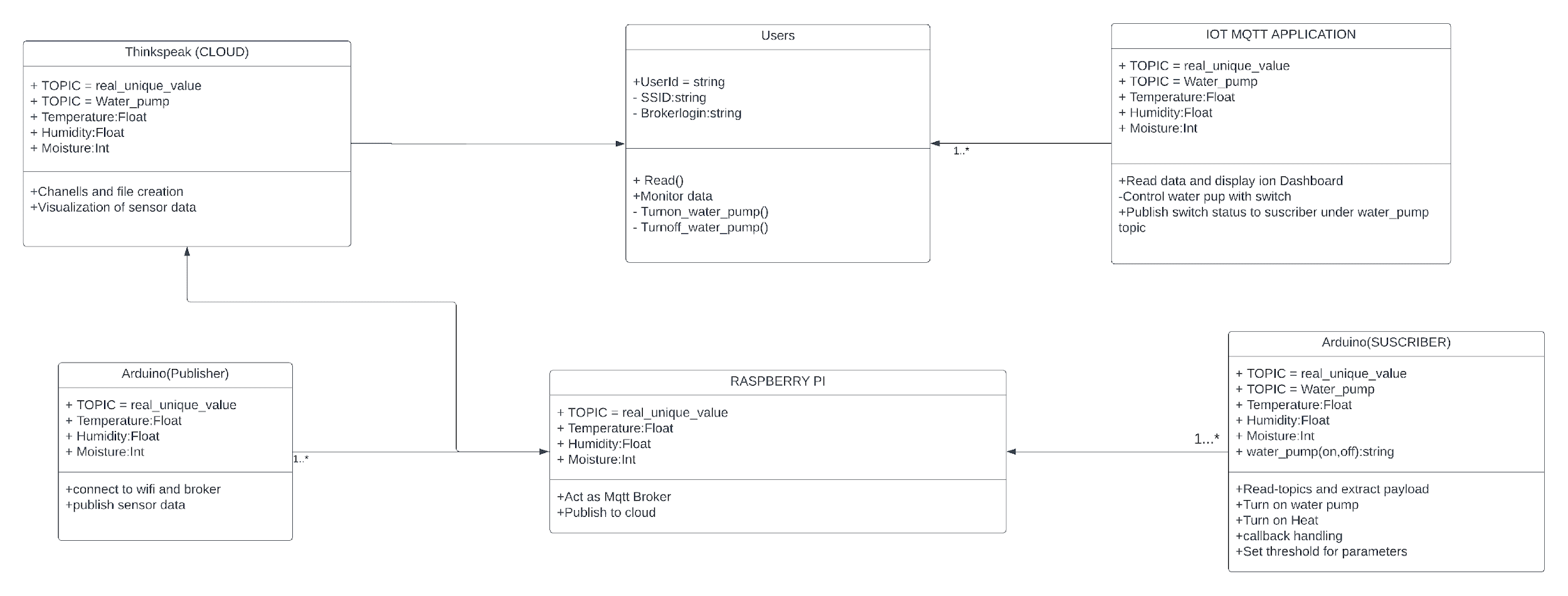
**Sensors:** We utilized an Arduino Uno Wi-Fi rev 2 as a sensor network where we had a funduino moisture sensor and DHT11 combinational sensor (temperature and humidity) connected to. Data was collected from this sensor and processed on this Arduino uno rev where we also had our publisher code running.

**Actuators:** Another Arduino Uno rev 2 acted as our actuator controller where we process data and turn water pump on based on certain conditions.

**Programming languages**: Various programming languages were utilized but they mostly included python and C, we used c to program the Arduino devices and we used python for the raspberry pi. A lot of libraries were used to achieve all the functionality from Arduino secrets to paho Mqtt.

# Implementation

* Download the Raspbian OS image, write it to an SD card, and boot up the Raspberry Pi. Access the terminal or connect a monitor and keyboard, and use the sudo raspi-config command to access the configuration menu. Connected to the Raspberry Pi via SSH using an SSH client by entering the IP address, username, and password.
* Set up an MQTT broker on a Raspberry Pi to act as a centralized messaging hub for MQTT communication. The MQTT broker serves as a middleware that facilitates the exchange of messages between MQTT clients, enabling efficient and reliable communication in Internet of Things (IoT) applications. Installed and configured the MQTT broker software, such as Mosquitto, on the Pi. This involves installing the necessary dependencies, configuring the broker settings, and securing the communication with appropriate authentication and encryption mechanisms. Once the MQTT broker is set up, it can handle MQTT connections, subscribe to topics, and route messages to the connected MQTT clients.
* ARDUNIO (SENSOR NODE): Collects data from the respective sensors DHT11 and moisture sensors and publishes this data to the Raspberry pi broker.
* Raspberry pi: acts as broker using mosquitto where other clients can subscribe to the published data from the sensor node. It is also used as a client to send data to the cloud (Thinkspeak).
* ARDUINO (ACTUATOR): Acts as a client to the Raspberry Pi broker and subscribes to sensor data topic, extracts sensor data readings from the broker and run conditional statements to turn on water pumps and heat. It also subscribes to data from the topic Water pump on the IOT MQTT panel app, where a button exists that controls the water pump through MQTT.
* Thinkspeak (CLOUD): Data being sent from Rasbeery pi (Temperature, Humidity, Moisture) is published to the smart garden channel on the cloud where it can be analyzed and visualized.
* Mobile app: IOT MQTT panel app is used to establish an mqtt connection with the system it is subscribed to the ARDUINO SENSOR NODE published sensor data and reads the data to be displayed on the screen. It also creates a topic called water pump where it sends “on ” or “off ”payload to toggle the water pump on or off.



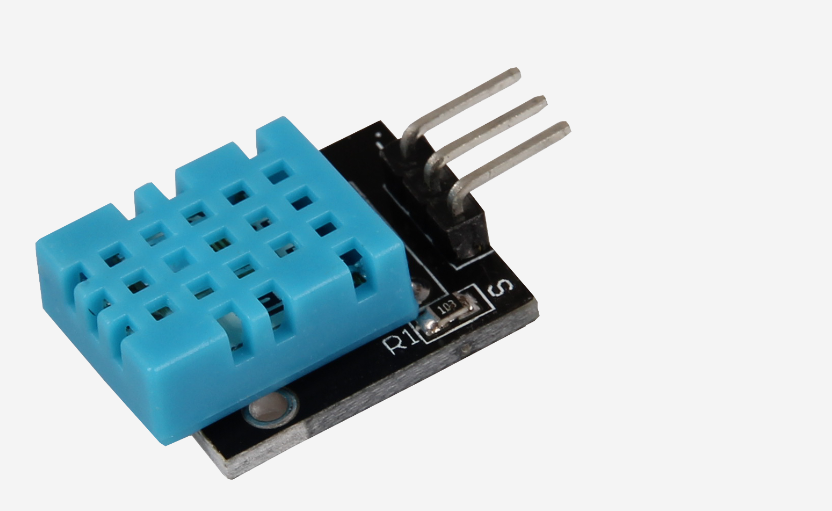
**Figure 2:Class Diagram showing the system and its components.**

The actor in this system is designed to be the USER, from the class diagram above we can see that the actions available for the user involves monitoring of the system data (Temperature, Humidity, moisture) and also turning on the water pump via the application.

**Sensor configuration:**

After setting up all the devices in the system the next stage was wiring of the different sensors needed for data collection.

**DHT11**



**Figure 3:ky-015 DHT11**

The DHT11 is a low-cost digital temperature and humidity sensor module. It provides accurate measurements of ambient temperature and relative humidity, making it popular for various applications, including environmental monitoring and home automation systems.

**Funduino Moisture sensor**

The Funduino Moisture Sensor is a module used to measure the moisture content of soil or other materials. It provides analog output that can be read by microcontrollers like Arduino, allowing for monitoring and automation of watering systems in applications such as gardening and agriculture**.**

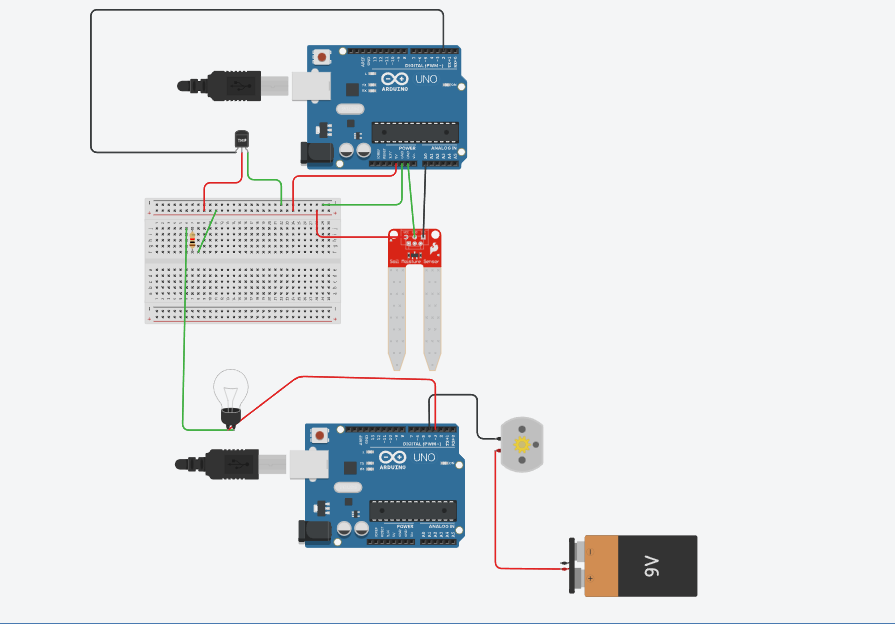
A close-up of a red and white device

Description automatically generated with low confidence

**Figure 4: Funduino moisture sensor**

**TINKERCAD representation of our wiring:**

Below we see a representation of our circuit and wiring of the sensors and the actuators.

****

A clearer understanding of our wiring can be seen in the picture below,the moisture sensor is already fit into the flower vase.



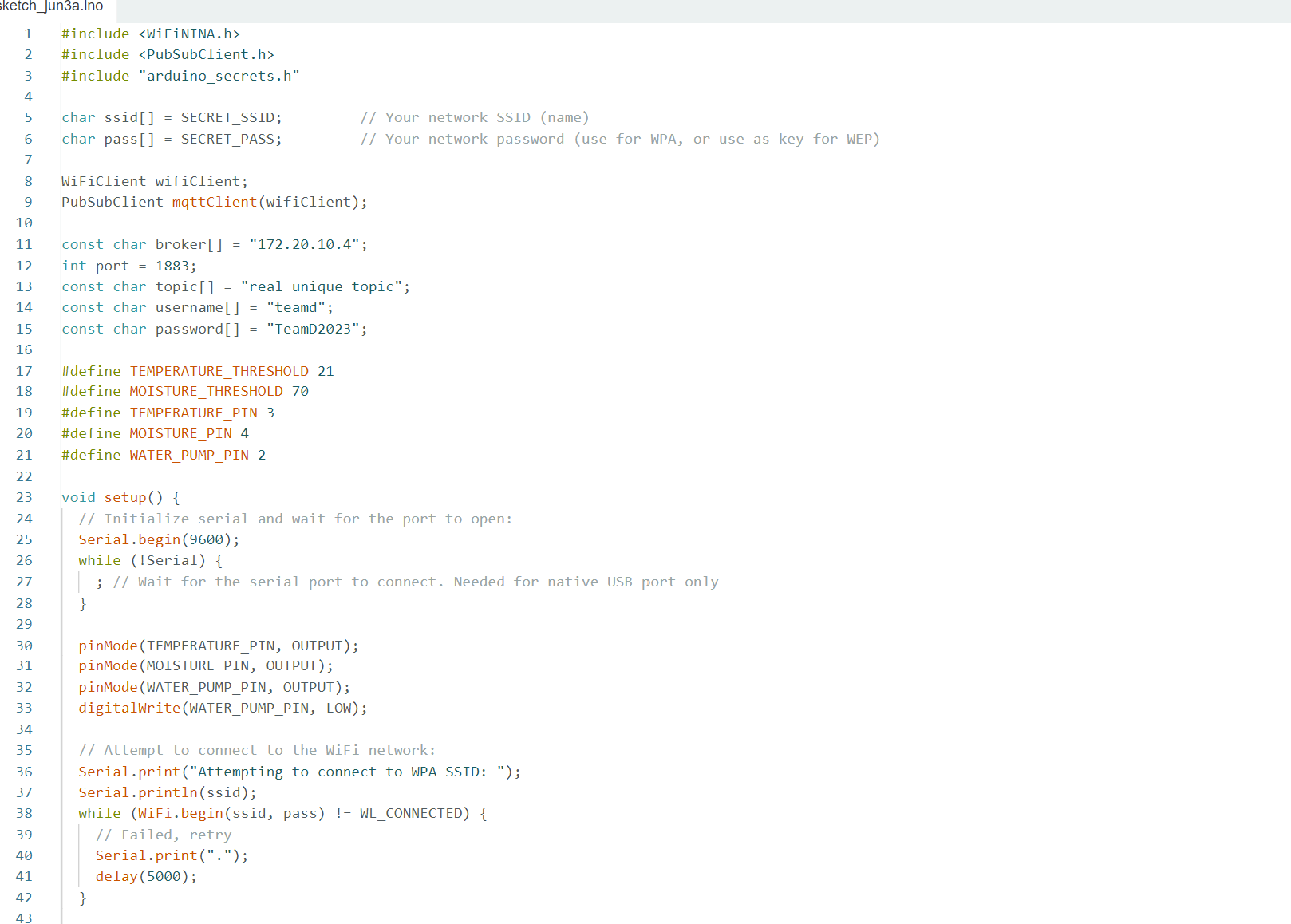
**CODE**

**PUBLISHER**

****

A screenshot of a computer

Description automatically generatedAbove is the code for publisher our sensor data from the sensor node to the raspberry pi broker, we used a lot of imported libraries, and the main functions include setting up Wi-Fi connection, setting up MQTT connection and setting up the sensors and transfer of data. We sent data to the topic real\_unique\_topic where other components of our system can subscribe too,and we always kept track of the success or failure status of each function for debugging.

**Subscriber**

**Code for MQTT Actuator**

This code creates a MQTT actuator with an Arduino board and the WiFiNINA shield. The actuator can receive MQTT messages and use them to control various output pins. It can also monitor sensor data and take actions based on preset thresholds.

**Dependencies**

This code is dependent on the libraries listed below:

* WiFiNINA: This library supports WiFi connectivity using the WiFiNINA shield.
* PubSubClient: This library enables the Arduino board to communicate with a MQTT broker as a MQTT client.

Install these libraries before running the code.

**Variables and Constants**

The code starts by defining a number of constants and variables:

* ssid and pass: The SSID (name) and password of the WiFi network to which the Arduino board should connect are stored in these variables.
* wifiClient: A WiFiClient object used for the WiFi connection.
* mqttClient: A PubSubClient object used for the MQTT connection.
* port and broker: These variables provide the MQTT broker's IP address and port number.
* The MQTT topic to which the actuator subscribes in order to receive messages.
* The credentials used to authenticate the actuator with the MQTT broker are username and password.
* TEMPERATURE\_THRESHOLD and MOISTURE\_THRESHOLD are temperature and moisture threshold values, respectively.
* The output pins TEMPERATURE\_PIN, MOISTURE\_PIN, and WATER\_PUMP\_PIN are used to control temperature, moisture, and a water pump, respectively.

**SETUP FUNCTION**

The setup function is only called once at the beginning. Its functions are as follows:

* The serial transmission is started at a baud rate of 9600.
* Sets the temperature, moisture, and water pump pin modes to OUTPUT.
* Connects to the WiFi network given by the ssid and pass parameters.
* Connects to the MQTT broker using the credentials provided.
* Subscribes to the MQTT topic and the "water\_pump" subject given by topic.
* If the connection to the MQTT broker fails, it will retry after 5 seconds.

**The Loop Function**

Following the setup function, the loop function is called repeatedly. Its functions are as follows:

* Determines whether the MQTT client is still connected. If this is not the case, it invokes the reconnect method to establish a new connection.
* Allows the MQTT client to process both incoming and outgoing messages.

**Function of Reconnection**

When the MQTT client loses its connection, the reconnect function is called. Its functions are as follows:

* Attempts to reconnect to the MQTT broker using the credentials provided.

If the connection is established, it subscribes to the MQTT subject indicated by topic as well as the "water\_pump" topic.

* If the connection fails, it prints an error message and attempts again after a 5-second delay.

**MessageReceived Method**

When a new MQTT message is received, the messageReceived function is called. Its functions are as follows:

* Prints a message confirming the receipt of a new message.
* The received payload (message) is converted to a string.
* Checks whether the received subject is "water\_pump" and takes the necessary action based on the message ("on" or "off").
* Extracts and processes temperature and moisture data from received messages.
* Temperature and moisture pins are set to HIGH or LOW based on the stated thresholds.
* Adds a 100-millisecond delay to guarantee that data processing is completed before reading additional data.

**Conclusion**

This code implements the fundamental functionality of a MQTT actuator, allowing control of output pins depending on MQTT messages received and monitoring sensor data. It connects to WiFi, connects to a MQTT broker, and handles message receiving and processing. You can alter the thresholds and add extra code to tailor the actuator's behavior.

**THINKSPEAK**

This code demonstrates an MQTT subscriber that receives messages from a specific topic and publishes the extracted sensor data (temperature, humidity, and moisture) to a ThingSpeak channel. It utilizes the Paho MQTT client library for Python and the requests library for making HTTP requests to ThingSpeak.

**MQTT Broker Settings**

* broker: The IP address or hostname of the MQTT broker.
* port: The port number of the MQTT broker.
* username: The username to authenticate with the MQTT broker.
* password: The password to authenticate with the MQTT broker.

**ThingSpeak Settings**

* channel\_id: The ID of the ThingSpeak channel where the sensor data will be published.
* api\_key: The API key for accessing the ThingSpeak channel.
* MQTT on\_connect Callback: This callback function is triggered when the MQTT client successfully connects to the MQTT broker. It subscribes to the topic where the sensor data is being published.

**MQTT on\_message Callback**

This callback function is called whenever a new MQTT message is received. It extracts the sensor readings (temperature, humidity, and moisture) from the payload and converts them to the appropriate data types. Then, it publishes the sensor values to the corresponding fields on the ThingSpeak channel using HTTP requests.

**MQTT Client Initialization and Configuration**

* The MQTT client is created using mqtt.Client().
* The MQTT client's username and password are set using client.username\_pw\_set(username, password).
* The on\_connect and on\_message callbacks are assigned to the MQTT client using client.on\_connect = on\_connect and client.on\_message = on\_message, respectively.

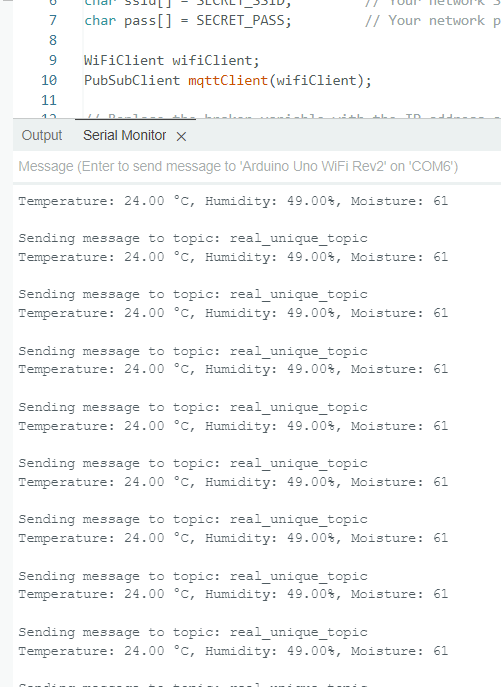
**MQTT Connection and Loop**

* The MQTT client connects to the MQTT broker using client.connect(broker, port).
* The MQTT client starts the client loop by calling client.loop\_forever(), which maintains the connection and handles incoming messages indefinitely.
* Publishing Sensor Data to ThingSpeak
* The extracted sensor data (temperature, humidity, and moisture) is published to ThingSpeak fields using HTTP requests.
* The requests.get() function is used to send the HTTP requests with the appropriate URL constructed based on the ThingSpeak API and the sensor values.
* After each publication, a delay is introduced to allow sufficient time between publishing different sensor values.
* Please note that the delays introduced in the code (time.sleep()) are used to ensure proper timing between publishing sensor data to ThingSpeak. You can adjust these delays according to your specific requirements. A screenshot of a computer program

  Description automatically generated with medium confidence

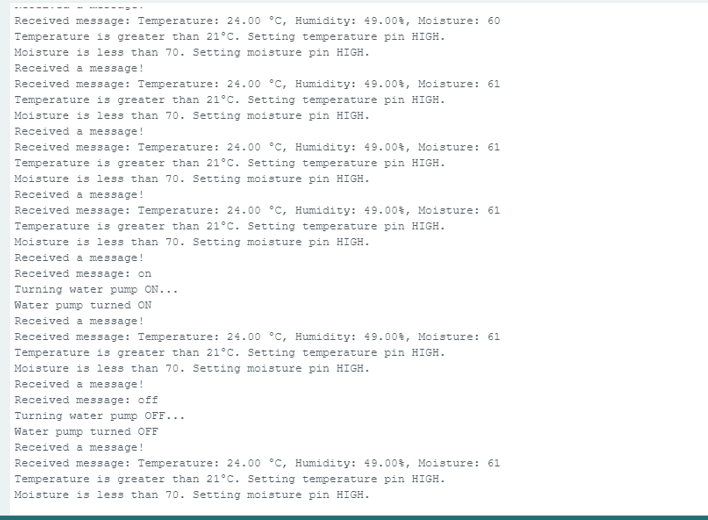
**RESULTS OF IMPLEMENTATION**

**PUBLISHER**

****

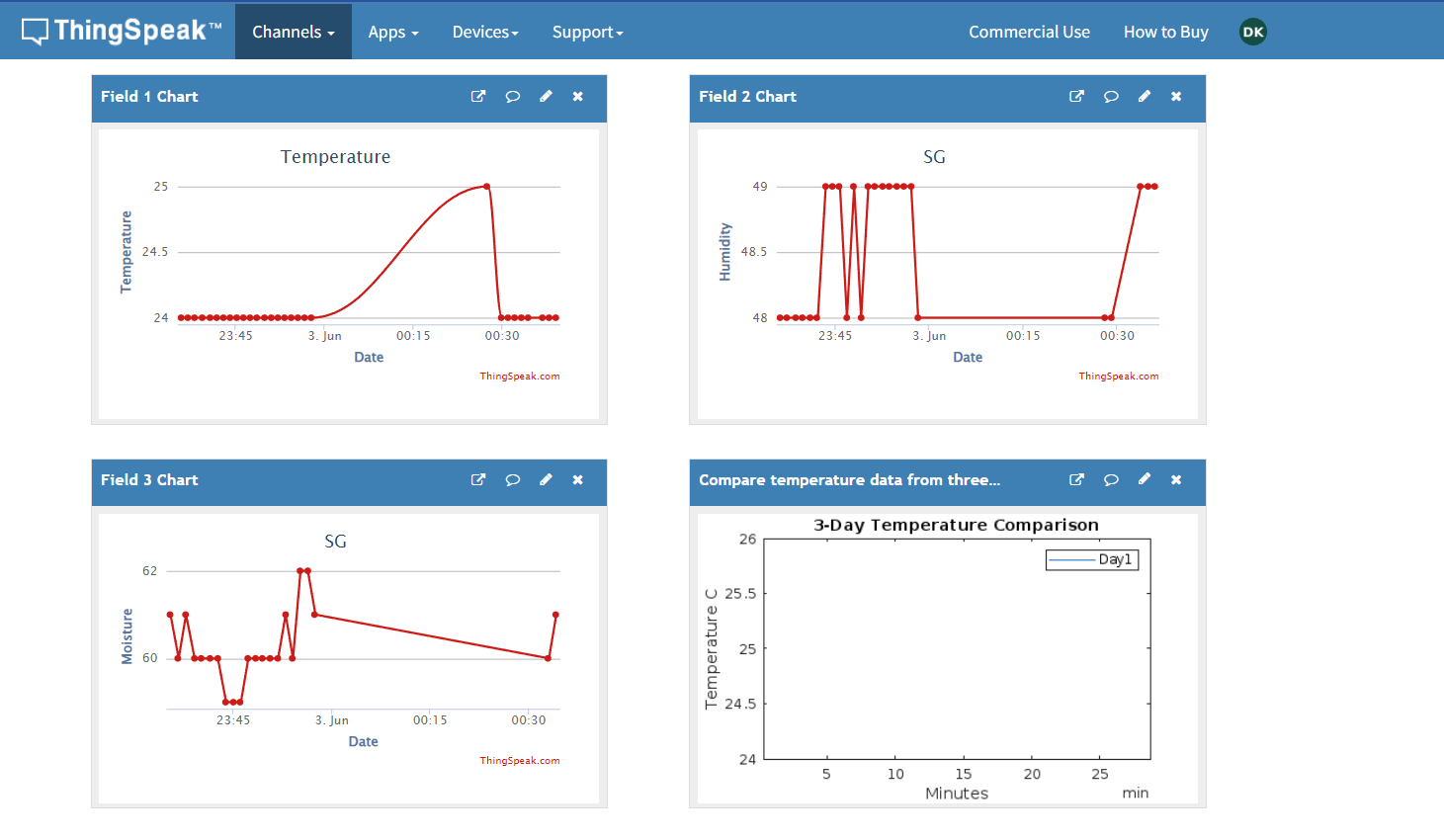
This is what shows on the serial monitor of the wireless sensor Arduino, as you can see it sends the message in this format to the top real\_unique\_topic.

**Subscriber**



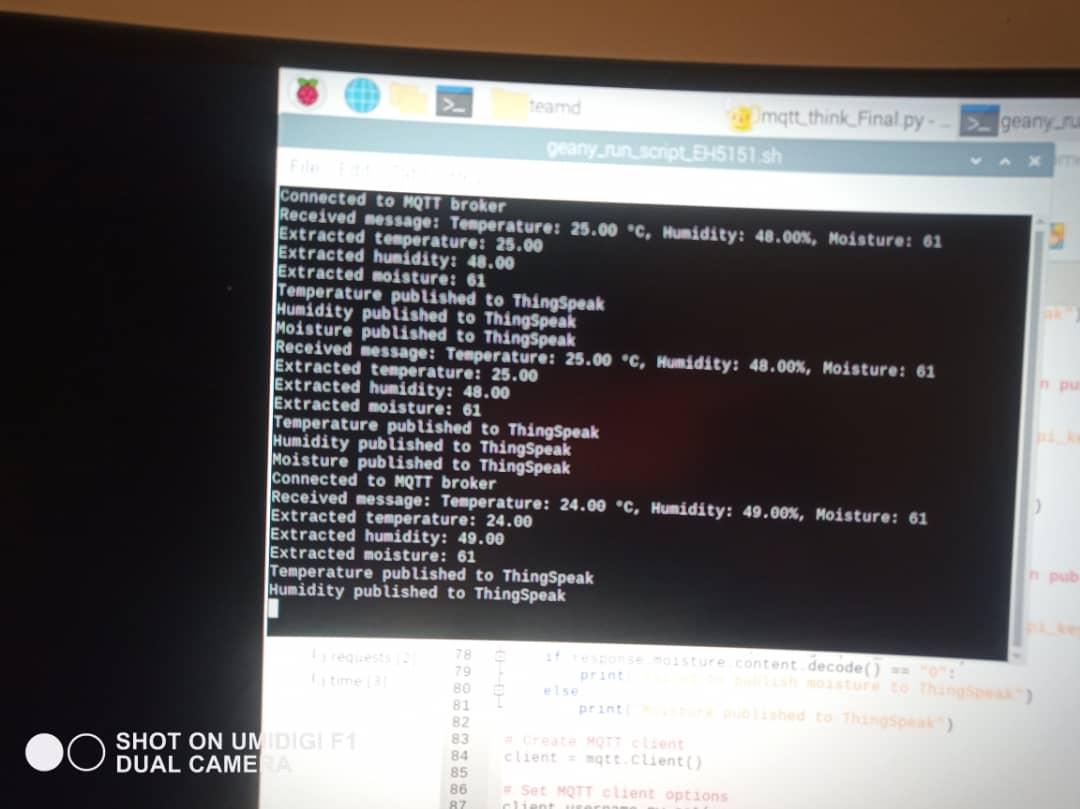
In the subscriber we see the messages being sent displayed on the serial monitor and based on this we see different conditions being executed, for example, the current condition is if the temperature is greater than 21 we set the pin high( 21 was used to ensure the functionality as the readings at this stage), also we see another condition that checks for the moisture data from the received message and if less than 70 which is our optimal moisture level we turn on the moisture pin which relates to the water pump to increase the moisture of our garden, and condition is the water\_pump topic created by the IOT panel app we check the message received for the payload on or off , in which the Arduino turns on or off the responsible pin.

**Thinkspeak Cloud**

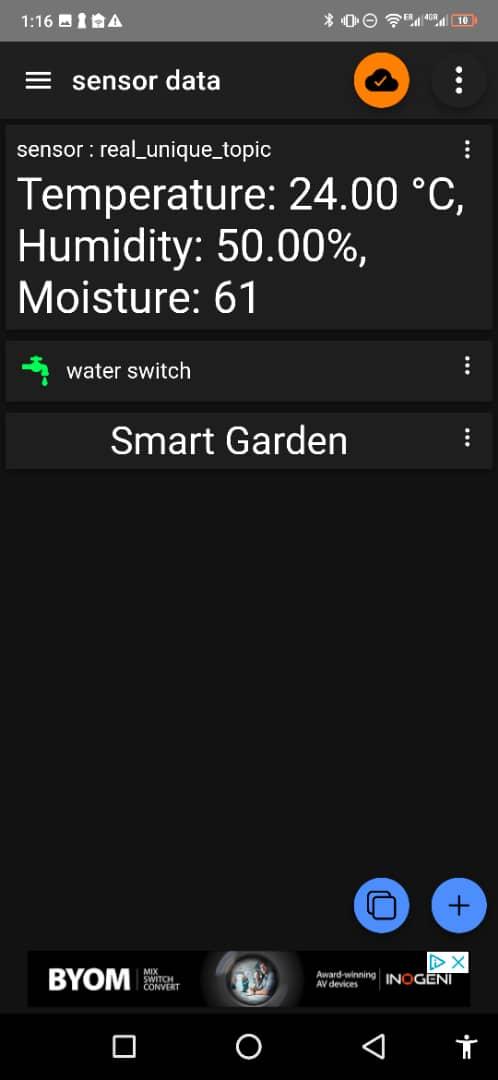
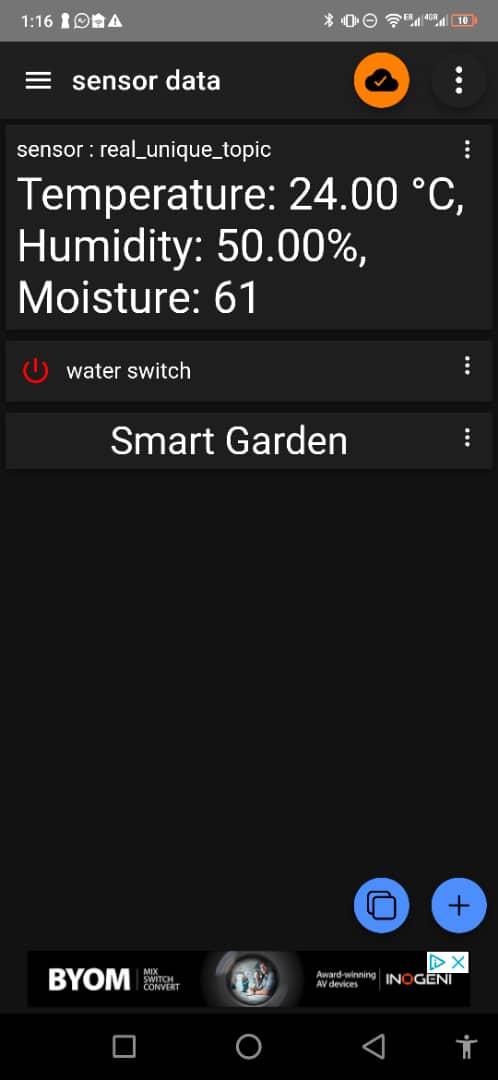


We ran the python code already shown in the code section, after setting up the raspberry pi as a client to the topic where sensor data is being published, and above is the result as shown in our Thinkspeak cloud account. This data can be accessed from anywhere in the world using this link and my log in credentials "https://thingspeak.com/channels/2164717/private\_show".

**Rasberry terminal for Thinkspeak Code**



**IOT MQTT PANEL Application**

****

The application is already subscribed to the main topic "real\_unique\_topic," where sensor data is being published. It extracts the embedded message from the topic and displays it.

Additionally, the application features a button that controls the water pump. When the button is pressed, a message is sent to the subscriber Arduino (actuator) to initiate the corresponding actions, such as turning the water pump on or off.

# Use Case

The use case of our application involves:

**Monitoring smart garden parameters**.

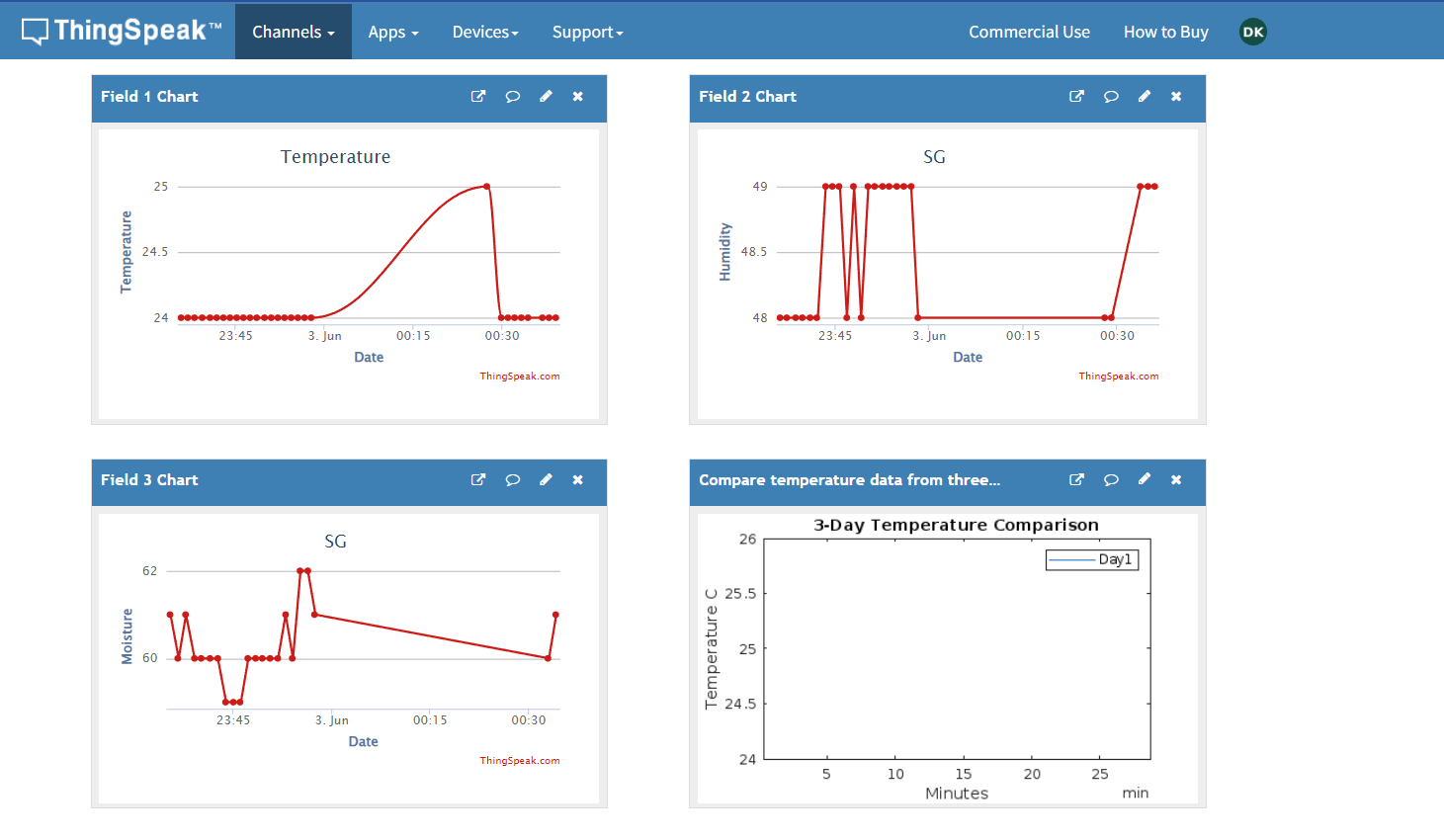
Users can read real-time data from their mobile phones while on the same network or from anywhere in the world via the cloud.

* For mobile phones, this can be achieved by simply connecting to the same network as the system and accessing the IoT MQTT Panel app. The app displays real-time readings from the temperature, humidity, and moisture sensors.
* For the Thinkspeak Cloud, users only need to log in to their Thinkspeak account where the channel is created, and data is being sent. Here, users can visualize the data and utilize various analysis tools.

**Controlling Smart Garden Components:**

Users can control smart garden components like water pump and thermostat with our system .

* As shown in the IOT MQTT APP session we have a water pump switch that the user can turn on , on demand.
* The remaining systme components like thermostats are operated automatically , there is also an automatic fucntion that turns on the water pump when moisture falls below a certain level.
* The cloud can also be used for analysis and visualization, users can check average readings or even more complex data analysis if neccesary.



**Figure 5: Thinkspeak showing visualization and analysis.**

**Sensor Node Arduino (Publisher)**

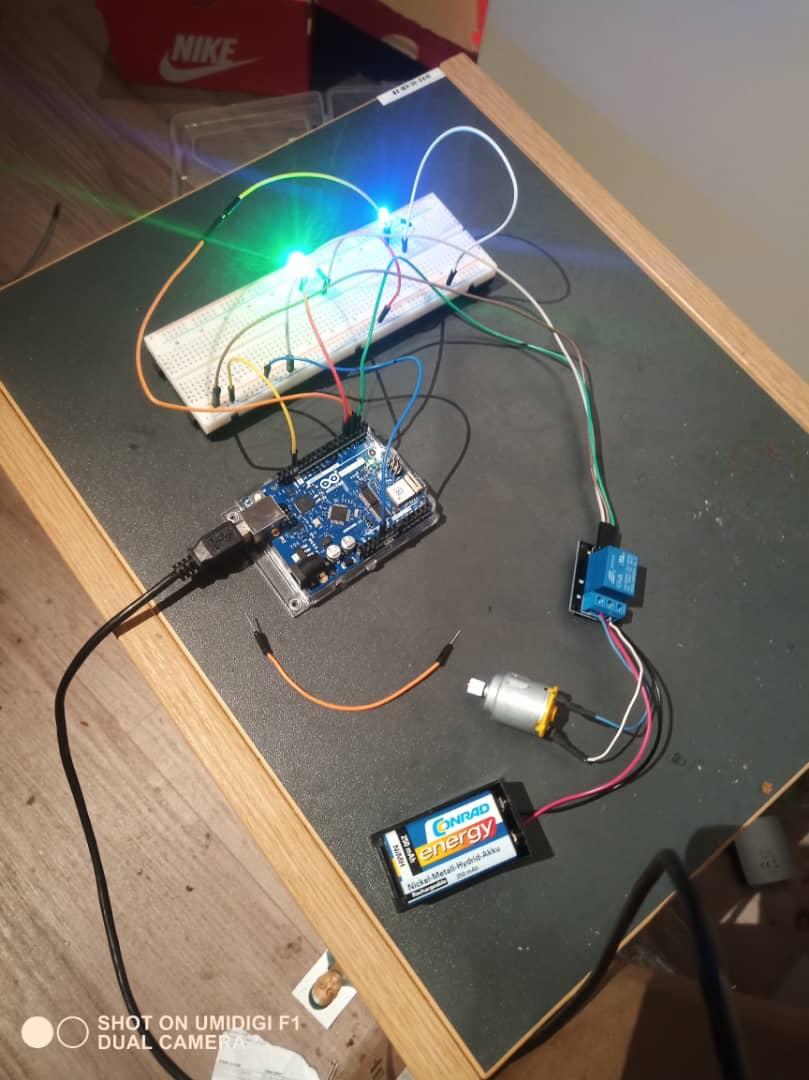
Collection of data for the system.



**Figure 6: Sensor Node Arduino collecting readings.**

**Actuator Arduino (Subscriber)**

Actuator Arduino reads messages and executes set conditions based on message values.



**Figure 7: Actuator Arduino**

# Sources/References

*Provide the sources on the technologies and algorithms you used in your project (Github).*