**GSM MQTT WITH ultasonic**

**ABSTRACT:**

This project focuses on the implementation of a remote monitoring and control system using an Arduino microcontroller paired with a SIM800 GSM module. The primary goal is to control a buzzer and monitor distances using an ultrasonic sensor, with data transmission handled via the MQTT protocol and the Shiftr.io MQTT broker. By utilizing the GSM network, this system circumvents the need for local Wi-Fi, making it highly suitable for remote locations where traditional internet connectivity may be unreliable or unavailable. This approach showcases the potential of integrating GSM modules with IoT applications to enhance flexibility and reach.

At the heart of this project is the SIM800 GSM module, which provides robust cellular connectivity. This module communicates with the Arduino through serial connections, allowing the system to connect to the internet via the cellular network. The configuration involves specifying the Access Point Name (APN), and handling the connection process using the TinyGSM library. Once connected, the Arduino can interact with the MQTT broker to send and receive messages. This setup is crucial for applications that require reliable internet connectivity in areas without Wi-Fi access.

The MQTT protocol plays a pivotal role in this project, offering a lightweight and efficient method for data transmission. MQTT is particularly well-suited for IoT applications due to its low bandwidth requirements and real-time capabilities. In this project, the Arduino subscribes to a specific MQTT topic to receive commands for the buzzer and publishes distance measurements from the ultrasonic sensor to another topic. The PubSubClient library facilitates this interaction, managing the connection to the MQTT broker and ensuring that messages are handled appropriately. This implementation highlights the advantages of using MQTT for real-time control and monitoring in IoT systems.

In practical terms, this project demonstrates how GSM and MQTT technologies can be combined to create a versatile and scalable remote monitoring and control solution. The ability to control a buzzer remotely and monitor distances through an ultrasonic sensor opens up numerous applications, from security systems to environmental monitoring. The modular and well-structured code allows for easy adaptation and scaling, making it a valuable framework for various IoT projects. Overall, this project not only showcases the technical feasibility of such integrations but also emphasizes their practical applications in enhancing connectivity and control in remote or challenging environments

**COMPONENTS:**

**ARDUINO:**Arduino is an open-source electronics platform based on easy-to-use hardware and software, designed for creating interactive projects. It allows users to write code and upload it to a microcontroller to control various sensors, motors, lights, and other electronic components.

.

**GSM SIM800C:**The GSM SIM800C is a quad-band GSM/GPRS module that provides a cost-effective and reliable solution for embedding cellular connectivity in projects. It supports voice, SMS, and data transmission, making it ideal for IoT applications requiring remote communication.



**Jumper Wires**: Jumper wires are insulated wires with connectors (typically male or female pins) at each end, used to create temporary or semi-permanent connections between different components on a breadboard or between a breadboard and other devices.



**BUZZER:**The buzzer system integrated with the Arduino IoT Cloud enables remote toggling of its state, providing a seamless user experience for activating or deactivating auditory feedback. This functionality is achieved through the synchronization of the NodeMCU microcontroller with the cloud platform, facilitating real-time control and monitoring of the buzzer's operations from anywhere with an internet connection.

**CODE:**

#define TINY\_GSM\_MODEM\_SIM800

#define SerialMon Serial

#define SerialAT Serial

// #define TINY\_GSM\_DEBUG SerialMon // Comment this line to disable debug output

const char apn[] = "airtelgprs.com";

const char gprsUser[] = "";

const char gprsPass[] = "";

const char\* broker = "yellowroarer326.cloud.shiftr.io";

const char\* mqttUsername = "yellowroarer326";

const char\* mqttPassword = "jrSOKfZYGNgdnS7a";

const char\* topicOutput1 = "/buzz/sig";

const char\* topic = "/buzz/state";

// SIM card PIN (leave empty if not defined)

const char simPIN[] = "";

int p = -1;

bool buzzerStateChanged = false; // Flag to indicate if the buzzer state has changed

#include <Wire.h>

#include <TinyGsmClient.h>

#include <PubSubClient.h>

TinyGsm modem(SerialAT);

TinyGsmClient client(modem);

PubSubClient mqtt(client);

#define BUZZER\_PIN 6

long lastReconnectAttempt = 0;

void mqttCallback(char\* topic, byte\* message, unsigned int len) {

Serial.print("Message arrived on topic: ");

Serial.print(topic);

Serial.print(". Message: ");

String messageTemp;

for (int i = 0; i < len; i++) {

Serial.print((char)message[i]);

messageTemp += (char)message[i];

}

Serial.println();

if (String(topic) == "/buzz/sig") {

Serial.print("Changing buzzer state to ");

if (messageTemp == "on") {

Serial.println("ON");

digitalWrite(BUZZER\_PIN, HIGH);

if (p != 1) {

p = 1;

buzzerStateChanged = true; // Set the flag when the buzzer state changes

}

} else if (messageTemp == "off") {

Serial.println("OFF");

digitalWrite(BUZZER\_PIN, LOW);

if (p != 0) {

p = 0;

buzzerStateChanged = true; // Set the flag when the buzzer state changes

}

}

}

}

boolean mqttConnect() {

SerialMon.print("Connecting to ");

SerialMon.print(broker);

boolean status = mqtt.connect("GsmClientN", mqttUsername, mqttPassword);

if (status == false) {

SerialMon.println(" fail");

return false;

}

SerialMon.println(" success");

mqtt.subscribe(topicOutput1);

SerialMon.print("Subscribed to topic: ");

SerialMon.println(topicOutput1);

return mqtt.connected();

}

void setup() {

SerialMon.begin(9600);

delay(10);

pinMode(BUZZER\_PIN, OUTPUT);

digitalWrite(BUZZER\_PIN, LOW); // Ensure the buzzer is off initially

SerialMon.println("Wait...");

SerialAT.begin(9600);

delay(6000);

SerialMon.println("Initializing modem...");

modem.restart();

String modemInfo = modem.getModemInfo();

SerialMon.print("Modem Info: ");

SerialMon.println(modemInfo);

if (simPIN[0] != '\0' && modem.getSimStatus() != 3) {

modem.simUnlock(simPIN);

}

SerialMon.print("Connecting to APN: ");

SerialMon.print(apn);

if (!modem.gprsConnect(apn, gprsUser, gprsPass)) {

SerialMon.println(" fail");

} else {

SerialMon.println(" OK");

}

if (modem.isGprsConnected()) {

SerialMon.println("GPRS connected");

}

mqtt.setServer(broker, 1883);

mqtt.setCallback(mqttCallback);

mqtt.setKeepAlive(120); // Set keep-alive interval to 120 seconds

}

void loop() {

if (!mqtt.connected()) {

SerialMon.println("=== MQTT NOT CONNECTED ===");

uint32\_t t = millis();

if (t - lastReconnectAttempt > 10000L) {

lastReconnectAttempt = t;

if (mqttConnect()) {

lastReconnectAttempt = 0;

}

}

delay(100);

return;

}

mqtt.loop();

if (buzzerStateChanged) { // Check if the buzzer state has changed

const char\* status = (p == 1) ? "BUZZER ON" : "BUZZER OFF";

SerialMon.print("Publishing: ");

SerialMon.println(status);

delay(1000);

bool publishStatus = mqtt.publish(topic, status, true);

if (publishStatus) {

SerialMon.println("Publish successful");

} else {

SerialMon.println("Publish failed");

}

buzzerStateChanged = false; // Reset the flag after publishing the message

}

}



**CODE EXPALNATION**

### Preprocessor Directives

* #define TINY\_GSM\_MODEM\_SIM800: Tells the code to use the SIM800 modem library for cellular communication.
* #define SerialMon Serial: Defines SerialMon as an alias for the hardware serial port used for communication with the monitor.
* #define SerialAT Serial: Defines SerialAT as an alias for the hardware serial port used for communication with the cellular modem.

### Connection Details

* const char apn[] = "airtelgprs.com";: Specifies the Access Point Name (APN) for connecting to the cellular network.
* const char gprsUser[] = "";: Username for GPRS authentication (if required).
* const char gprsPass[] = "";: Password for GPRS authentication (if required).

### MQTT Details

* const char\* broker = "yellowroarer326.cloud.shiftr.io";: Specifies the MQTT broker address.
* const char\* mqttUsername = "yellowroarer326";: Username for MQTT authentication.
* const char\* mqttPassword = "jrSOKfZYGNgdnS7a";: Password for MQTT authentication.
* const char\* topicOutput1 = "/buzz/sig";: Topic to subscribe to for receiving buzzer control messages.
* const char\* topic = "/buzz/state";: Topic to publish the buzzer state.

### Buzzer and SIM PIN

* #define BUZZER\_PIN 6: Defines the pin connected to the buzzer.
* const char simPIN[] = "";: Specifies the SIM card PIN (if defined).

### Includes

* #include <Wire.h>: Included for potential I2C communication (not used in this code).
* #include <TinyGsmClient.h>: Header file for the TinyGSM library used for cellular communication.
* #include <PubSubClient.h>: Header file for the PubSubClient library used for MQTT communication.

### Object Instantiation

* TinyGsm modem(SerialAT);: Creates a modem object of type TinyGsm to interact with the cellular modem using the SerialAT port.
* TinyGsmClient client(modem);: Creates a client object of type TinyGsmClient to handle network communication using the modem object.
* PubSubClient mqtt(client);: Creates an mqtt object of type PubSubClient to manage communication with the MQTT server using the client object.

### Buzzer Setup

* pinMode(BUZZER\_PIN, OUTPUT);: Sets the buzzer pin as an output pin.
* digitalWrite(BUZZER\_PIN, LOW);: Initializes the buzzer to be off initially.

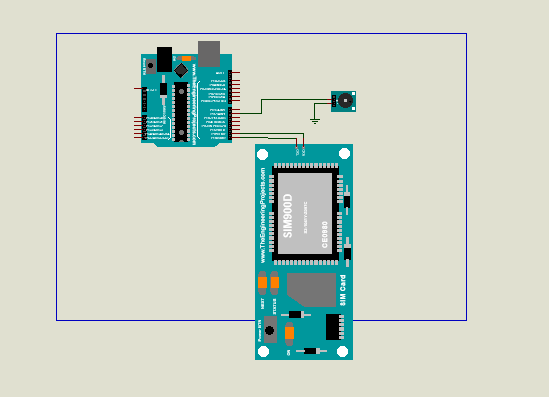
### Initialization (setup() function)

* SerialMon.begin(9600);: Initializes communication with the monitor at 9600 baud rate.
* delay(10);: Adds a small delay to allow for serial initialization.
* SerialAT.begin(9600);: Initializes communication with the cellular modem at 9600 baud rate.
* delay(6000);: Adds a delay to allow the modem to boot up.
* modem.restart();: Restarts the modem.
* String modemInfo = modem.getModemInfo();: Retrieves and prints modem information.
* if (simPIN[0] != '\0' && modem.getSimStatus() != 3) { modem.simUnlock(simPIN); }: Checks if a SIM PIN is defined and the SIM status is not ready (PIN locked). If so, unlocks the SIM.
* if (!modem.gprsConnect(apn, gprsUser, gprsPass)) { SerialMon.println(" fail"); } else { SerialMon.println(" OK"); }: Attempts to connect to the cellular network using the defined APN, username, and password. Prints success or failure message.
* if (modem.isGprsConnected()) { SerialMon.println("GPRS connected"); }: Checks if the GPRS connection is established and prints a message accordingly.
* mqtt.setServer(broker, 1883);: Sets the MQTT broker address (including port).
* mqtt.setCallback(mqttCallback);: Sets the callback function for handling incoming MQTT messages.
* mqtt.setKeepAlive(120);: Sets the keep-alive interval for the MQTT connection.

### Main Loop (loop() function)

* **MQTT Connection Handling**:
  + Checks if the mqtt client is connected.
  + If not connected:
    - Prints a message indicating MQTT disconnection.
    - Attempts to reconnect after a 10-second delay if the last reconnect attempt was more than 10 seconds ago.
    - If reconnection is successful, resets the last reconnect attempt time.
  + Calls mqtt.loop() to keep the MQTT connection alive.
* **Buzzer State Publishing**:
  + Checks the buzzerStateChanged flag.
  + If the flag is true (meaning the buzzer state has changed):
    - Creates a message string indicating the new state ("BUZZER ON" or "BUZZER OFF").
    - Prints a message indicating publishing the state.
    - Attempts to publish the message to the specified topic using mqtt.publish(). Prints success or failure message.
    - Resets the buzzerStateChanged flag after publishing.

**CIRCUIT DIAGRAM:**

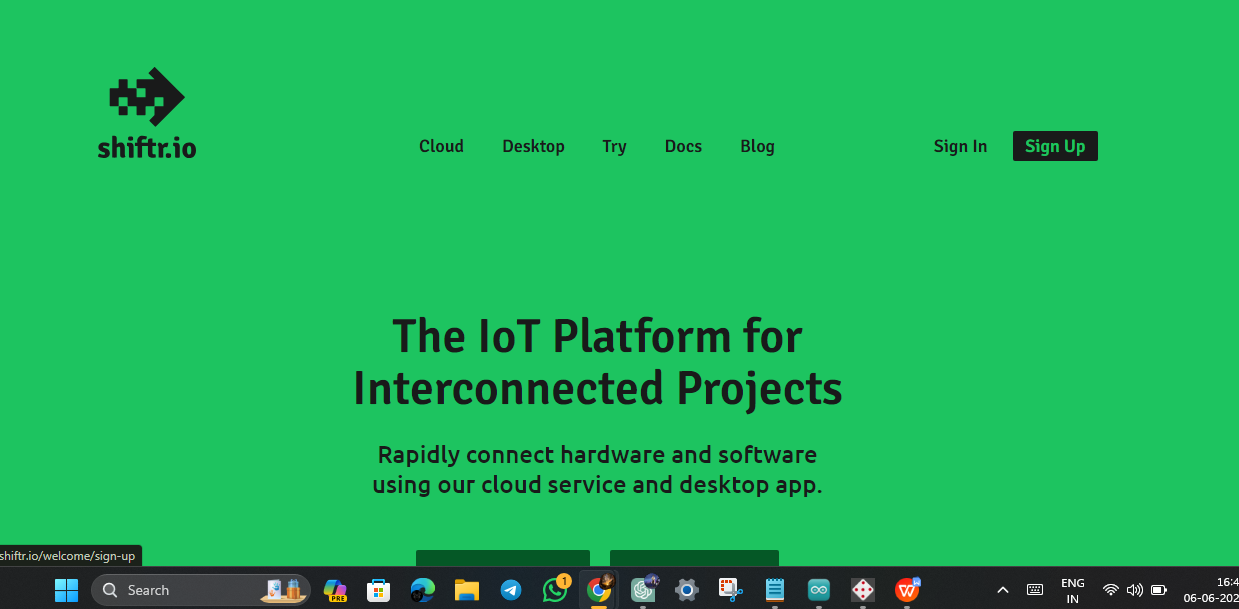


**STEPS TO CREATE A PROJECT**

**Step 1: Set Up Shiftr.io**

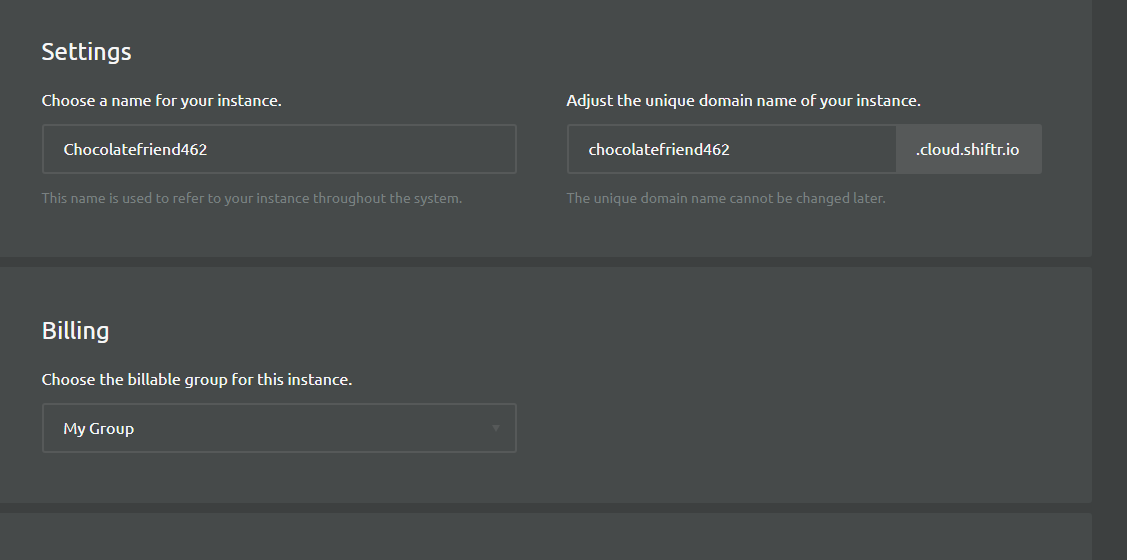
**Create an Account:**

* Visit Shiftr.io and create an account.



**Create a New Space:**

* After logging in, create a new space. This space will be your MQTT broker.



**Get Connection Details:**

* Note the broker URL, username, and password provided by Shiftr.io. You will need these for your ESP32 and MQTT Box setup.

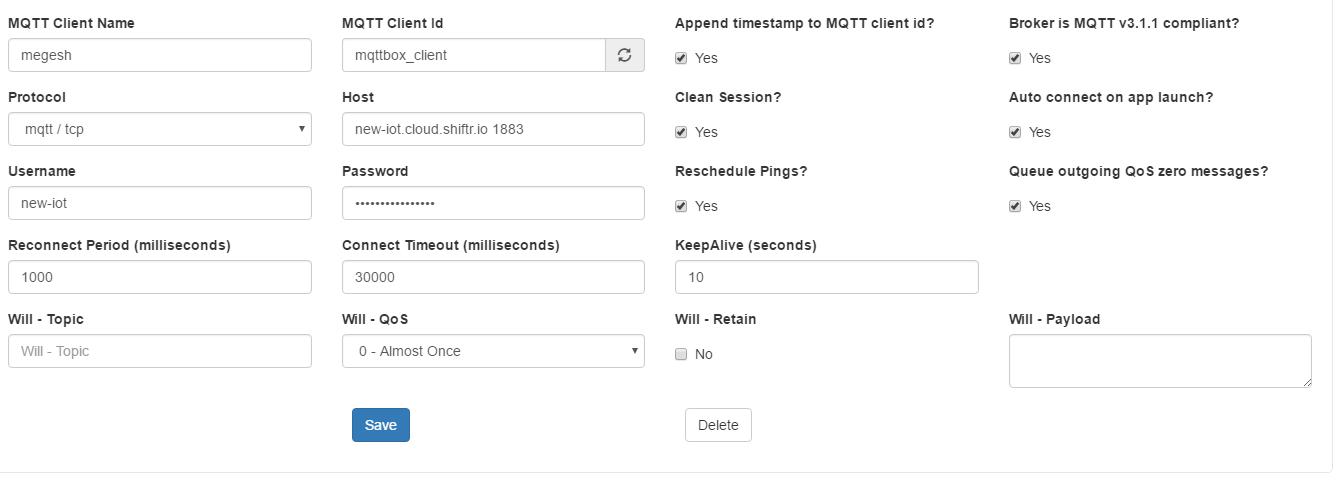
**Step 2: Set Up MQTT Box**

**Install MQTT Box:**

* Download and install MQTT Box from MQTT Box.

**Create a New Connection:**

* Open MQTT Box and create a new connection.
* Enter the broker URL, username, and password you got from Shiftr.io.



**Subscribe to Topics:**

* Subscribe to the following topics to monitor and control the buzzer:
* /ulta for distance measuring

**Step 3: Connect Hardware Components**

**Ultrasonic Sensor:**

* Connect the VCC pin to 5V on the ESP32.
* Connect the GND pin to the ground.
* Connect the Trig pin to a digital pin (e.g., D4).
* Connect the Echo pin to another digital pin (e.g., D5).

**Libraries:**

* Ensure you have the necessary libraries installed:
* MQTT
* TINY\_GSM\_MODEM\_SIM800

**Step 5: Upload Code to arduino**

**Arduino IDE Setup:**

* Ensure you have thearduini uno board support installed in the Arduino IDE.
* Select the correct board and port from the Tools menu.

**Upload the Code:**

* Connect your arduino uno to the computer via USB.
* Upload the code.

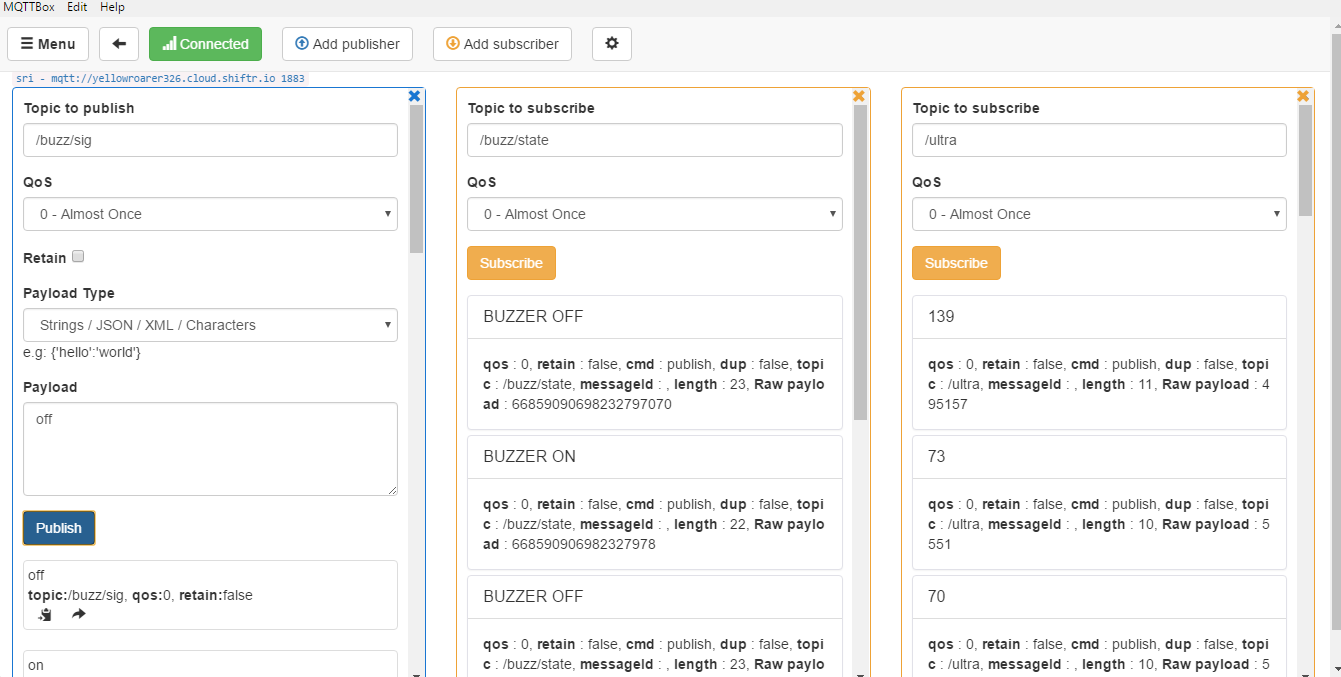
**Step 6: Monitor and Control via MQTT Box**

**Monitor Distance**

* In MQTT Box, check the messages published to the /ultrasonic topic to see the distance measurements from the ultrasonic sensor.

**Monitor Buzzer Status:**

* Subscribe to the /ultra topic to receive updates distance



**CONCLUSION:**

The project successfully demonstrates the capability and effectiveness of remote monitoring and control using GSM and MQTT technologies. By integrating the SIM800 GSM module with an Arduino microcontroller, and utilizing the Shiftr.io MQTT broker, the system achieves reliable remote control and real-time data transmission. This integration highlights the robustness of the SIM800 module for cellular connectivity, which is essential for applications in areas lacking Wi-Fi access. The use of the MQTT protocol ensures efficient and lightweight communication, ideal for IoT applications requiring real-time updates and control.

The practical implementation of this project showcases how remote control of a buzzer and monitoring of distance measurements via an ultrasonic sensor can be achieved seamlessly. The modular design and clear structure of the code make it adaptable and scalable for other IoT devices and applications. This versatility underscores the potential for creating a wide range of remote monitoring solutions, from security systems to environmental monitoring, using similar technological frameworks.

Furthermore, the project emphasizes the importance of reliable internet connectivity and real-time communication in remote applications. The combination of GSM and MQTT technologies provides a robust solution for maintaining connectivity and control in challenging environments. This project serves as a practical demonstration of how these technologies can be leveraged to enhance the functionality and reach of IoT systems.

In conclusion, this project not only achieves its objective of remote monitoring and control but also highlights the broader applicability and potential of integrating GSM and MQTT technologies in IoT solutions. The successful implementation and operation of the system pave the way for further innovations and developments in the field of remote IoT applications, offering valuable insights and a solid foundation for future projects.