**GSM MQTT WITH BUZZER**

**ABSTRACT:**

This project aims to develop a remote buzzer control system using GSM (Global System for Mobile Communications) and MQTT (Message Queuing Telemetry Transport) protocols, integrated with the Shiftr.io MQTT broker. The system employs the SIM800 GSM module for cellular connectivity and an Arduino microcontroller to interface with the GSM module and handle MQTT communication. The main objective is to enable the control of a buzzer through remote commands sent over the internet, achieving real-time status updates of the buzzer state using MQTT while utilizing readily available and cost-effective hardware components for implementation.

The system components include the SIM800 GSM module for cellular network connectivity, an Arduino microcontroller as the central control unit, a buzzer as the output device, and the Shiftr.io MQTT broker to facilitate MQTT messaging between the client and the server. The methodology involves setting up the hardware by connecting the SIM800 module to the Arduino microcontroller and the buzzer to a designated GPIO pin. The software implementation includes initializing the GSM module, configuring the Arduino to connect to the Shiftr.io MQTT broker, handling received MQTT messages to control the buzzer, and publishing the buzzer's state to inform remote users.

The system successfully demonstrates the ability to remotely control the buzzer through MQTT messages. The buzzer can be turned on or off based on commands received via the MQTT broker, and its status is accurately reported back to the users. The implementation proves to be reliable with a stable connection and real-time responsiveness.

The project showcases an effective way to control and monitor devices remotely using GSM and MQTT technologies. This solution is scalable and can be extended to other IoT applications requiring remote control and monitoring. The use of the Shiftr.io MQTT broker simplifies the messaging infrastructure and enhances overall system efficiency. Future work could focus on extending the system to control multiple devices, implementing advanced security measures for communication, and exploring low-power modes to make the system more energy-efficient

**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.



**ULTRASONIC SENSOR:** The ultrasonic sensor emits sound waves and measures the time it takes for them to bounce back, enabling precise distance calculation. Integrated with the ESP8266, it provides real-time distance data wirelessly to the Blynk app, offering remote distance monitoring on mobile devices.



**CODE:**

#define TINY\_GSM\_MODEM\_SIM800

#define SerialMon Serial

#define SerialAT Serial

#define TINY\_GSM\_DEBUG SerialMon

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\* topic = "/ultra";

const char simPIN[] = "";

#include <Wire.h>

#include <TinyGsmClient.h>

#include <PubSubClient.h>

#include <NewPing.h>

#define TRIGGER\_PIN 5

#define ECHO\_PIN 4

#define BUZZER\_PIN 6

#define MAX\_DISTANCE 200 // Maximum distance we want to measure (in centimeters)

TinyGsm modem(SerialAT);

TinyGsmClient client(modem);

PubSubClient mqtt(client);

NewPing sonar(TRIGGER\_PIN, ECHO\_PIN, MAX\_DISTANCE);

long lastReconnectAttempt = 0;

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(60); // Set keep-alive interval to 60 seconds

}

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();

}

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(topic);

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

SerialMon.println(topic);

return mqtt.connected();

}

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();

// Wait a few seconds between measurements

delay(2000);

// Read distance from ultrasonic sensor

unsigned int distance = sonar.ping\_cm();

// Check if the reading is valid

if (distance == 0) {

Serial.println("Failed to read from ultrasonic sensor or out of range!");

return;

}

// Print and publish distance

Serial.print("Distance: ");

Serial.print(distance);

Serial.println(" cm");

mqtt.publish(topic, String(distance).c\_str(), true);

}

**CODE EXPALNATION**

### Preprocessor Directives

* #define TINY\_GSM\_MODEM\_SIM800: Specifies that the code should 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.
* #define TINY\_GSM\_DEBUG SerialMon: Enables debug messages printed to SerialMon.

### 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\* topic = "/ultra";: Topic to publish ultrasonic sensor readings and subscribe for incoming messages.

### SIM PIN

* 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.
* #include <NewPing.h>: Header file for the NewPing library used to handle the ultrasonic sensor.

### 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.
* NewPing sonar(TRIGGER\_PIN, ECHO\_PIN, MAX\_DISTANCE);: Creates a sonar object of type NewPing to handle ultrasonic sensor measurements.

### Pin Definitions

* #define TRIGGER\_PIN 5: Defines the pin connected to the ultrasonic sensor's trigger pin.
* #define ECHO\_PIN 4: Defines the pin connected to the ultrasonic sensor's echo pin.
* #define BUZZER\_PIN 6: Defines the pin connected to the buzzer.
* #define MAX\_DISTANCE 200: Sets the maximum distance to measure with the ultrasonic sensor in centimeters.

### 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.
* pinMode(BUZZER\_PIN, OUTPUT);: Sets the buzzer pin as an output pin.
* digitalWrite(BUZZER\_PIN, LOW);: Ensures the buzzer is off initially.
* 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(60);: Sets the keep-alive interval for the MQTT connection.

### MQTT Callback (mqttCallback() function)

* Handles incoming MQTT messages. Prints the topic and message to the serial monitor.

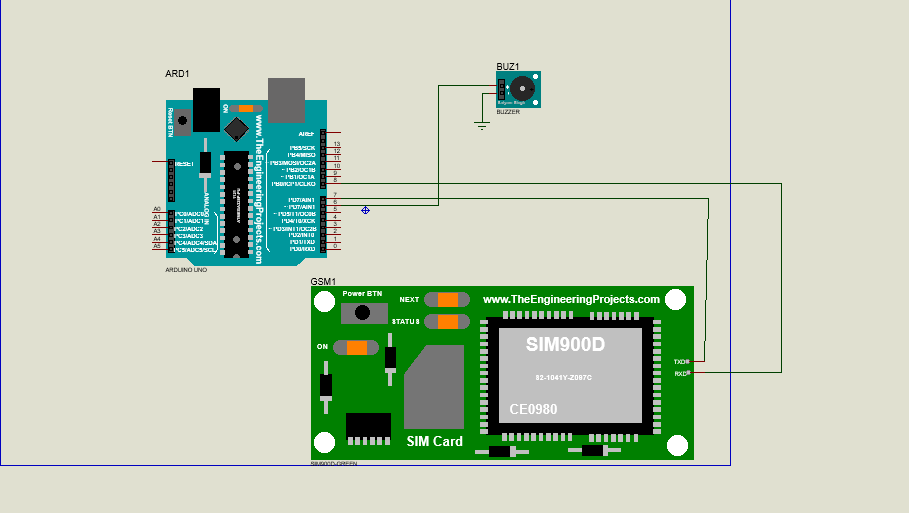
### MQTT Connection Handling (mqttConnect() function)

* SerialMon.print("Connecting to "); SerialMon.print(broker);: Prints the broker address.
* boolean status = mqtt.connect("GsmClientN", mqttUsername, mqttPassword);: Attempts to connect to the MQTT broker using the provided credentials.
* if (status == false) { SerialMon.println(" fail"); return false; } else { SerialMon.println(" success"); }: Checks the connection status and prints a success or failure message.
* mqtt.subscribe(topic);: Subscribes to the specified topic for incoming messages.
* SerialMon.print("Subscribed to topic: "); SerialMon.println(topic);: Prints the subscribed topic.
* Returns the MQTT connection status.

### Main Loop (loop() function)

* Handles MQTT connection status and ultrasonic sensor readings.
* if (!mqtt.connected()) { SerialMon.println("=== MQTT NOT CONNECTED ==="); }: Checks if the mqtt client is connected and prints a message if not.
* uint32\_t t = millis(); if (t - lastReconnectAttempt > 10000L) { lastReconnectAttempt = t; if (mqttConnect()) { lastReconnectAttempt = 0; } }: Attempts to reconnect to the MQTT broker if the last attempt was more than 10 seconds ago. Updates the last reconnect attempt time.
* mqtt.loop();: Keeps the MQTT connection alive.
* delay(2000);: Waits a few seconds between measurements.
* unsigned int distance = sonar.ping\_cm();: Reads distance from the ultrasonic sensor.
* if (distance == 0) { Serial.println("Failed to read from ultrasonic sensor or out of range!"); return; }: Checks if the reading is valid and prints a message if not.
* Serial.print("Distance: "); Serial.print(distance); Serial.println(" cm");: Prints the distance to the serial monitor.
* mqtt.publish(topic, String(distance).c\_str(), true);: Publishes the distance to the specified MQTT topic

**CIRCUIT DIAGRAM:**

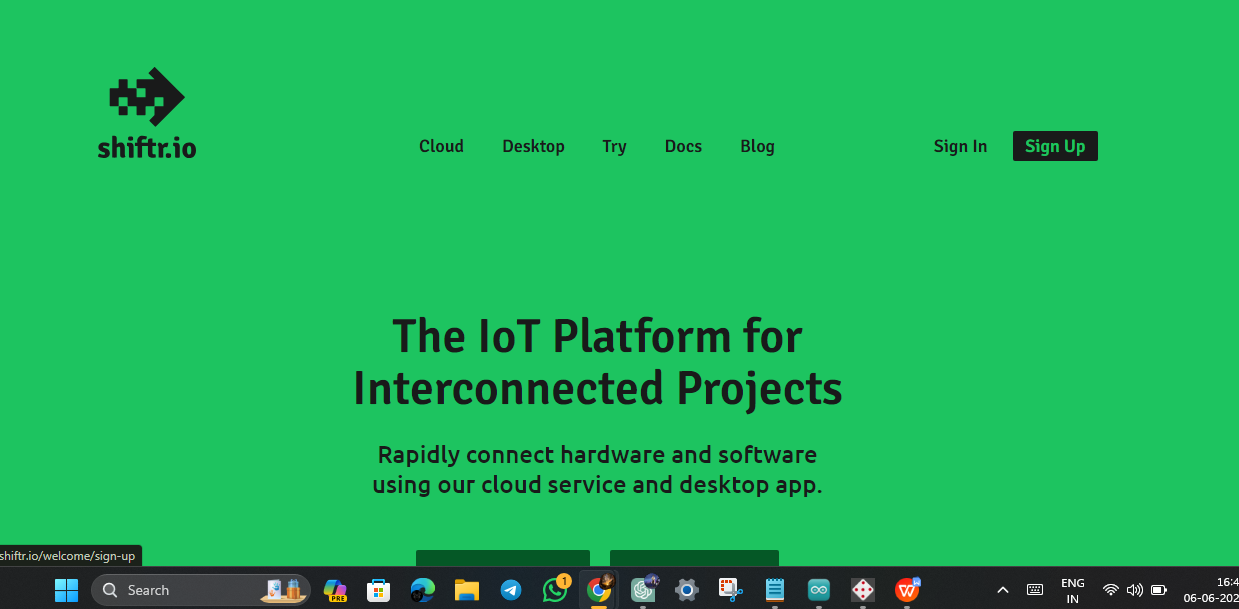


**STEPS TO CREATE A PROJECT**

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

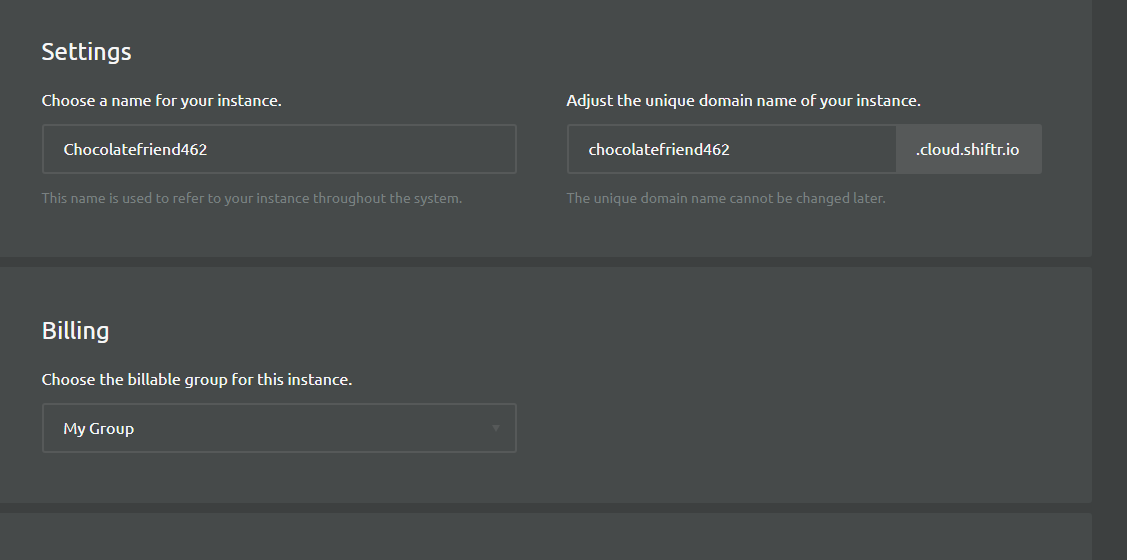
**Create an Account:**

* 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.

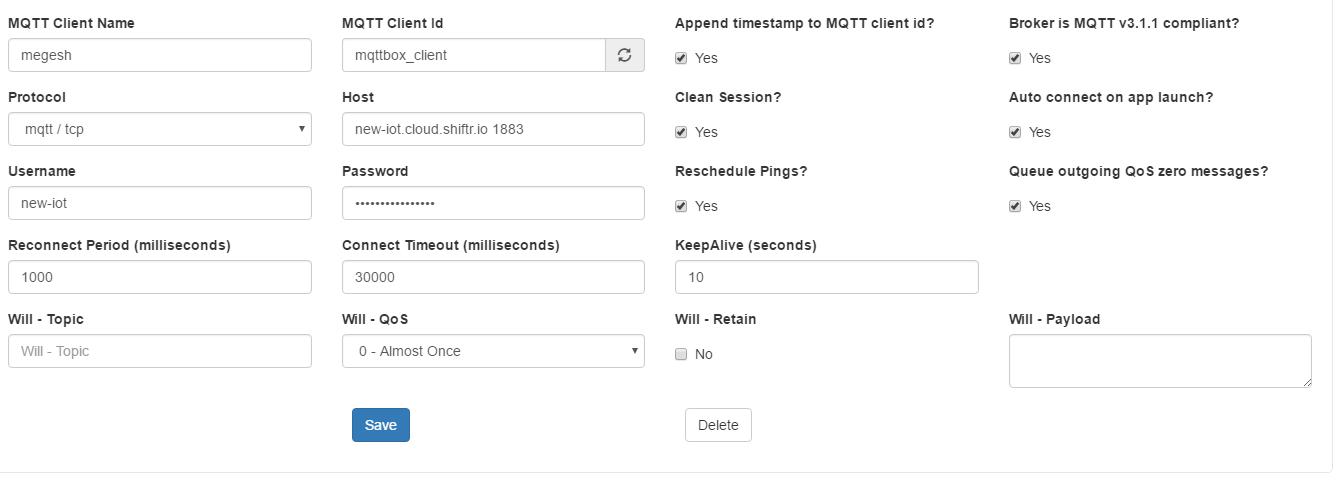
**Step 2: Set Up MQTT Box**

**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:
* /ultra for controlling the buzzer

**Step 3: Connect Hardware Components**

**Buzzer:**

* Connect the positive terminal to a digital pin (e.g., D26).
* Connect the negative terminal to the ground.
* Step 4: Write the Arduino Code

**Libraries:**

* Ensure you have the necessary libraries installed:
* WiFi
* MQTT

**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.

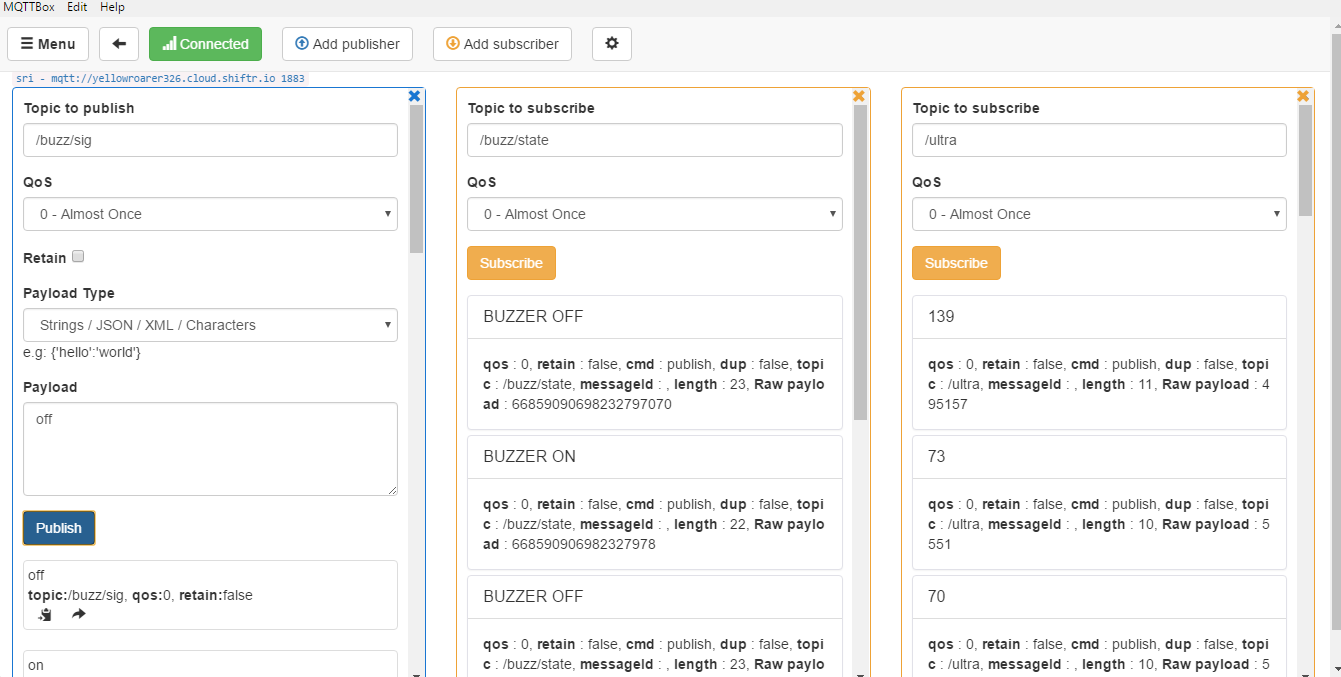
**Step 6: Monitor and Control via MQTT Box**

**Control the Buzzer:**

* Publish "on" to the /buzzer topic to turn the buzzer on.
* Publish "off" to the /buzzer topic to turn the buzzer off.

**Monitor Buzzer Status:**

* Subscribe to the /buzzerStatus topic to receive updates on the buzzer's status (ON or OFF).



**CONCLUSION:**

The project successfully demonstrates the capability of remote buzzer control using GSM communication and the MQTT protocol. By integrating the SIM800 GSM module with an Arduino microcontroller and utilizing the Shiftr.io MQTT broker, the system achieves reliable remote control and state reporting of the buzzer. This integration highlights the effective use of the SIM800 module for providing robust cellular network connectivity, allowing for internet communication without the need for local Wi-Fi networks.

Using the MQTT protocol ensures lightweight and efficient message exchanges, making it highly suitable for IoT applications that require real-time updates and commands. The ability to remotely toggle the buzzer and monitor its state through specific MQTT topics showcases the practicality and functionality of the system for various remote control applications. This capability is crucial for scenarios where devices need to be controlled and monitored from a distance, providing flexibility and convenience.

Moreover, the modular and well-structured code of this project ensures that it can be easily adapted and scaled for other IoT devices and use cases, making it a versatile solution for remote monitoring and control. The success of this project underlines the potential of combining GSM and MQTT technologies to create reliable and scalable IoT solutions. The project not only serves as a practical implementation of these technologies but also opens avenues for further enhancements and applications in the field of IoT.