IoT-Based Air Pollution Monitoring System

Prepared by: Jana Ayman - Habiba Amr
Maya Hossam - Mohamed ElDairoty

Introduction

Problem Statement

Air pollution has significant negative impacts on health, environment, and quality of life. Traditional monitoring methods are limited by lack of real-time data and remote accessibility. Our IoT-based air pollution monitoring system addresses these limitations.

Objectives

- Measure harmful gases.
- Trigger alerts via a buzzer and automatically activate ventilation based on predefined thresholds.
- Offer remote monitoring and control via a secure web interface accessible worldwide.

Background and Theory

IoT Technology Overview

Internet of Things (IoT) is an interconnected network of devices capable of collecting and exchanging data over the internet. This technology facilitates real-time monitoring and remote management capabilities, transforming everyday objects into smart systems.

Importance of Air Quality Monitoring

Continuous monitoring and timely action on air pollution can drastically reduce health risks and environmental damage by enabling proactive measures and public awareness.

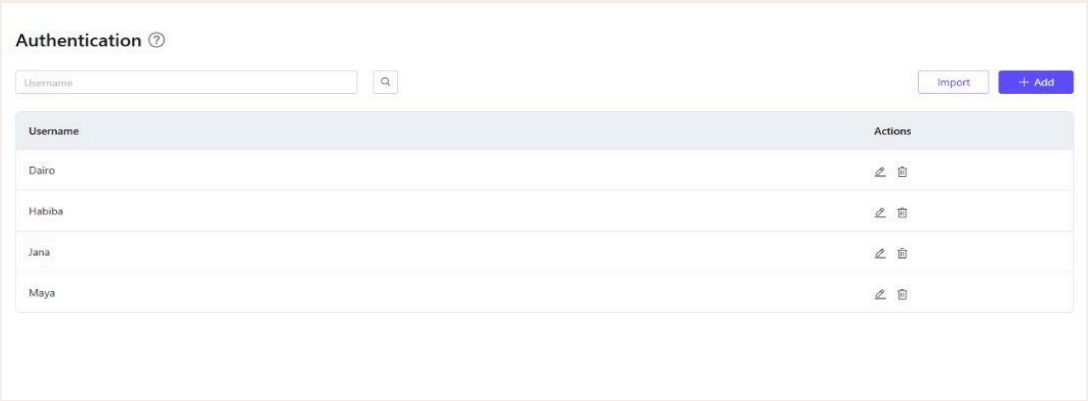
Project Description

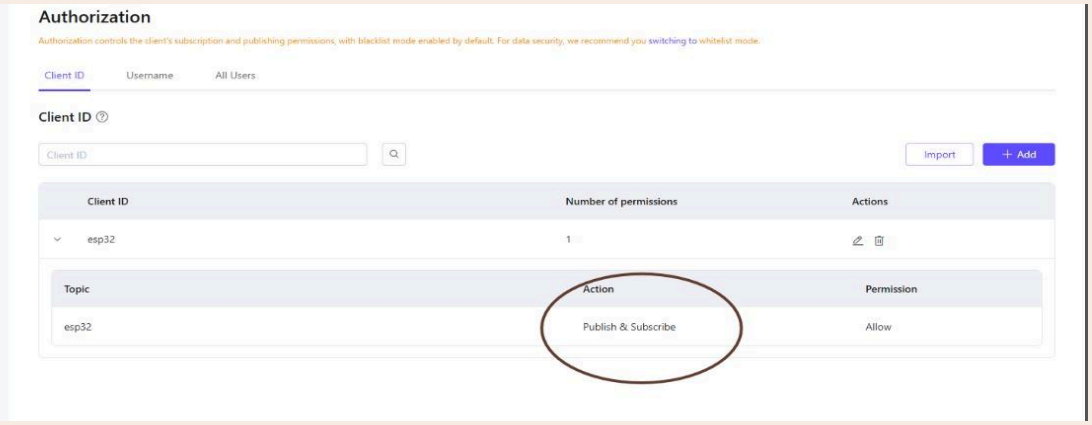
Components

- **ESP32 Node- MCU Microcontroller**: The device used for sensor integration and network communication.
- **BME680 Sensor**: It measures temperature, humidity, pressure, and gas concentrations.
- **Buzzer and Fan**: Used as alert mechanisms activated upon threshold exceedance.
- **OLED Display**: For data visualisation.
- **Relay Module:** 2-channel relay to act as a switch between the fan and buzzer.
- **Battery:** To power the relay module.

IoT Protocol (EMQX MQTT)

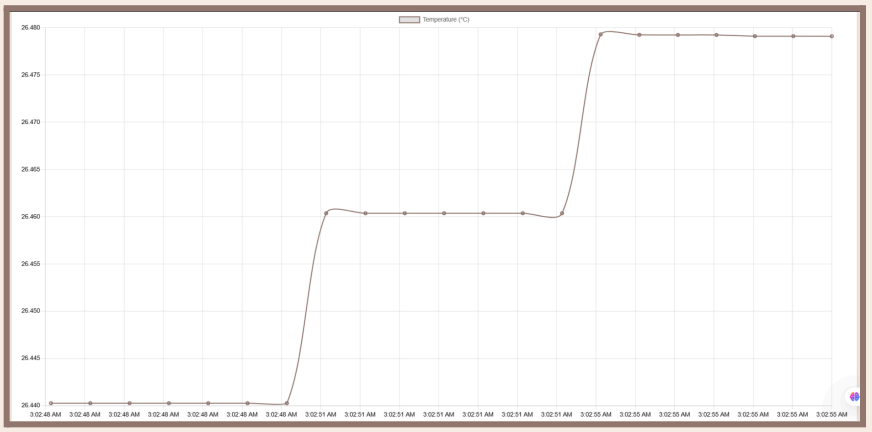
To manage communication between our ESP32 sensor node and the user interface, we used the **EMQX MQTT broker**, hosted securely on a cloud server. This broker plays a central role in handling all messaging between the publisher (the ESP32 board) and the subscribers, which include our custom-built web dashboard and any connected monitoring tools. EMQX was chosen for its scalability, reliability, and ease of integration with TLS.

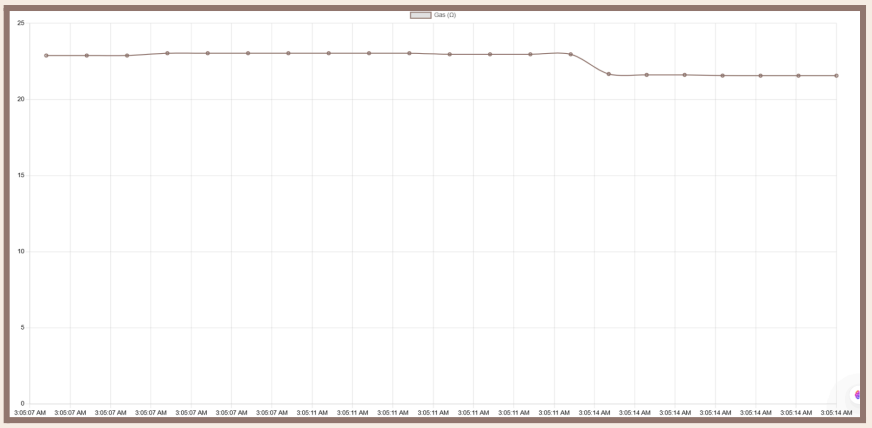




For secure data transmission, we implemented **MQTT over TLS**. We created our own Certificate Authority (CA) and used it to issue client certificates to the ESP32. The broker was configured to verify these certificates before allowing any connection, ensuring only authenticated devices could publish or subscribe. TLS was enabled on port **8883**, which is the standard secure MQTT port.

Networking and Web Page Implementation





A custom-designed webpage serves as the user interface, accessible from anywhere through ngrok tunneling. The page features:

- Real-time data visualization with charts and indicators.
- Live readings from the BME680 sensor.
- Remote control of the fan and buzzer based on sensor thresholds or manual commands.

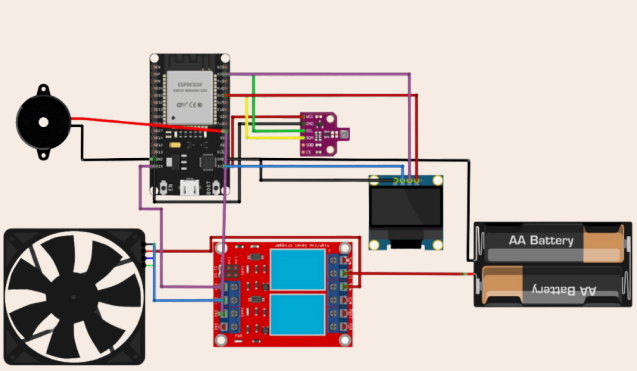
System Design

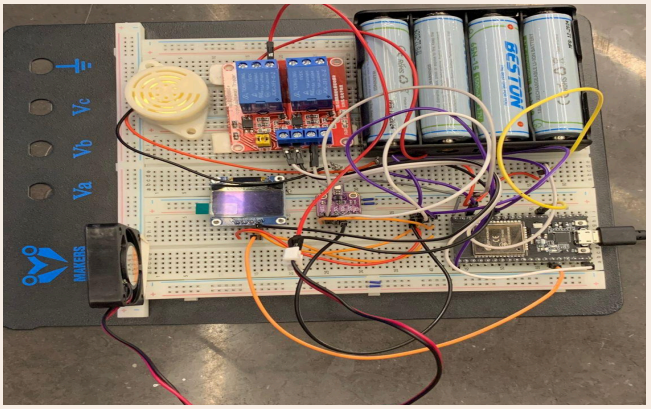
Microcontroller (ESP32) ↔ Backend Server (EMQX MQTT Broker) ↔ Frontend Interface (Webpage)

Communication Workflow

- ESP32 publishes sensor data to the MQTT server.
- Web client subscribes to these MQTT topics and displays live data.
- Commands from the web client are sent back through MQTT topics to control the buzzer and fan.

Implementation





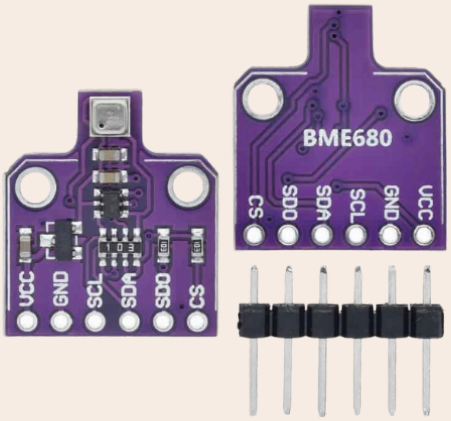
Breadboard and Circuit Implementation

The ESP32 microcontroller is what manages all sensor readings, controls, and network communication.

- The **BME680 sensor** is connected to the ESP32 using the I2C protocol. The SCL and SDA pins from the sensor are linked to GIOP 22 and GIOP 21 on the ESP32. VCC and GND are connected to 3V3 and GND.
- An **OLED display** is also connected by I2C method using the same GIOP pins (22 and 21), with its power lines connected to 3.3V and GND. Allowing for simultaneous real-time display of sensor data.
- A **buzzer** is connected to GIOP 4 of the ESP32. It activates when the gas reading exceeds the threshold value.
- A **relay** is connected to control our **cooling fan**, powered externally by a battery pack. The relay's signal pin is connected to GIOP 13 of the ESP32. When it is triggered, it will close the circuit and power the fan.

- The **fan** itself is connected to the relay's normally open (NO) and common (COM) terminals, drawing power from an external battery holder. This setup separates high-current fan operation from the microcontroller's power domain, preventing voltage drops.

Challenges Faced and How We Solved Them



During development, we encountered several technical issues. The most significant was with the **BME680 sensor**, which wasn't responding even though all connections were correct. After troubleshooting the wiring and I2C settings, we discovered that the **sensor's header pins weren't soldered**, causing poor electrical contact. Once we **soldered the pins properly**, the ESP32 was able to detect the sensor and read data successfully.

We also faced occasional **MQTT connection drops**, which we resolved by adjusting the keep-alive interval and double-checking our certificate setup. Additionally, our **OLED screen initially showed no output** due to incorrect GPIO pins, which we corrected after verifying the I2C wiring.

These challenges helped us better understand both hardware and software integration and emphasised the importance of careful inspection and testing at each step.

Conclusion

This project successfully delivered a secure and scalable IoT-based air pollution monitoring system that provides real-time tracking of environmental conditions. By integrating the ESP32 microcontroller with the BME680 sensor, we were able to monitor critical air quality parameters and trigger automatic responses when thresholds were exceeded. The system leverages secure MQTT communication through the EMQX broker and offers a user-friendly, remotely accessible webpage for live data visualization and device control.