KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY KUMASI, GHANA

GROUP 5

Title: IoT-Based Noise Pollution Monitor – Tracks noise levels and alerts when limits are exceeded.

GROUP MEMBERS

APAU, Kwadwo 1845322
AHENE, Emmanuel Adjei 1842822
KWAO NORTEY, Andrews 7134521
APEKE, Ebenezer Elorm 1845422
AIKINS, Olivia 1843022
ANUMEL, Henry Kwesi 1845122
APPIAH, Andrew Benedict 1845522
AYAMGA, Peter Atanga 1848722
DANSO, Carl Kwarteng 1850922
HAYIBOR, Irene Eyram Ama 1853522

Catalog

LIST OF TABLES AND FIGURES

PROBLEM STATEMENT

Noise pollution is a growing environmental concern, especially in urban areas, industrial zones, and public spaces. Excessive noise levels can lead to serious health issues, including stress, hearing loss, sleep disturbances, and reduced productivity. Traditional noise monitoring methods are either manual, expensive, or lack real-time tracking and alert mechanisms

There is a need for an automated, real-time noise pollution monitoring system that can continuously track noise levels and alert authorities or individuals when predefined limits are exceeded. By leveraging IoT technology, this system can provide accurate, remote, and real-time monitoring, helping to enforce noise regulations and promote a healthier environment.

METHODOLOGY

1. Simulation-Based Implementation

The project was not physically built but simulated using Wokwi, a powerful online simulation tool.
Key components from Wokwi used in the simulation:
Potentiometer (acts as the sound sensor)
Buzzer (alerts when noise exceeds a threshold)
ESP32 (microcontroller with IoT capabilities)
OLED Display (shows noise levels in dB)

2. Potentiometer as a Noise Sensor

Since real sound sensors were not used, the potentiometer simulated noise levels by producing a varying analog voltage.
The principle of a voltage divider allowed the potentiometer to generate different voltage levels, simulating real-time noise variations.

3. ESP32 as the Core Controller

The ESP32 microcontroller processed the analog voltage from the potentiometer.
It performed analog-to-digital conversion (ADC), mapping voltage values (0V to 3.3V → 0 to 4095 digital values).
These values were then mapped to corresponding decibel (dB) levels using software calculations.
WiFi-enabled IoT connectivity allowed data transmission and remote monitoring.

4. Displaying Noise Levels & Alert System

The OLED display showed real-time noise levels in dB.

A buzzer was programmed to sound whenever the noise level exceeded a predefined threshold in the code.

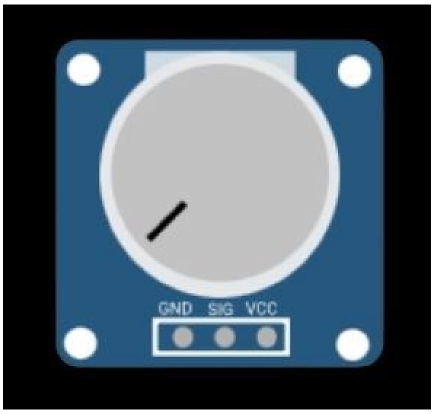

Figure 1: Figure of Potentiometer

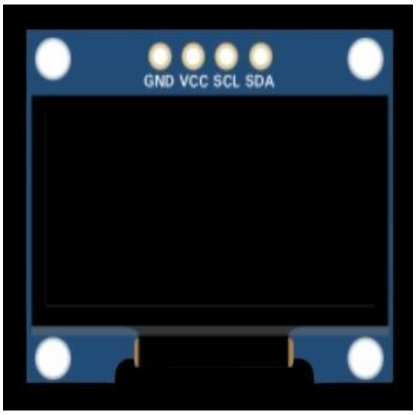

Figure 2: Figure of OLED

Figure 3: Figure of the ESP32

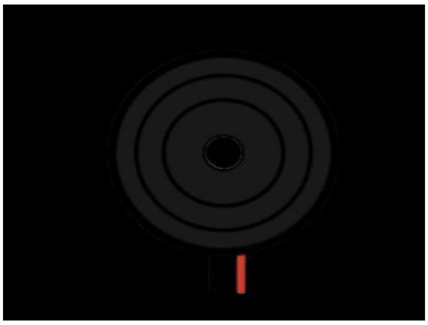
Figure 4: Figure of buzzer

Key Functionalities of the Code

1.Reading Analog Input from Potentiometer
The potentiometer acts as a simulated noise sensor, generating varying voltage levels.
The ESP32 reads this analog voltage via pin 34 (SOUND_SENSOR_PIN).

```
int sensorValue = analogRead(SOUND_SENSOR_PIN);
```

2. Mapping the Analog Value to Decibel (dB) Levels
The ESP32 has 12-bit ADC (0-4095 range).
The potentiometer's voltage output is mapped to a noise level range of 30dB to 100dB using:

```
float noiseLevel = map(sensorValue, 0, 4095, 30, 100);
```

3. Displaying Noise Level on OLED Screen
The Adafruit SSD1306 library is used to interface with the OLED display.
Noise levels are displayed in real-time.

```
display.print("Noise: ");
display.print(noiseLevel);
display.print(" dB");
```

4. Triggering the Buzzer Alert When Noise Exceeds a Threshold
If the noise level exceeds 60dB, the buzzer turns ON (1000 Hz tone).
Otherwise, it remains OFF.

```
if (noiseLevel > NOISE_THRESHOLD) {
   tone(BUZZER_PIN, 1000);
} else {
   noTone(BUZZER_PIN);
}
```

5. Sending Data to IoT Cloud (ThingSpeak)
The ESP32 connects to WiFi and sends data to ThingSpeak via HTTP requests.
The noise level is uploaded for real-time cloud monitoring and data logging.

```
String url = String(server) + "&field1=" + String(noiseLevel);
http.begin(url);
http.GET();
```

Schematics Overview

Components Used

- ESP32 – The main microcontroller that reads data, processes it, and sends it to the cloud.
- Potentiometer – Simulates noise levels by varying resistance (voltage output).
- Buzzer – Produces an alert sound when noise exceeds the threshold.
- OLED Display (SSD1306) – Displays real-time noise levels.

Table 1: Component connections

Component	ESP32 Pin	Function
Potentiometer (Vout)	GPIO 34 (Analog In)	Reads analog noise levels
Buzzer (+)	GPIO 26	Sounds alert when threshold exceeded
OLED Display (SDA)	GPIO 21	I2C Data Line
OLED Display (SCL)	GPIO 22	I2C Clock Line
WiFi	Built-in ESP32	Connects to IoT cloud

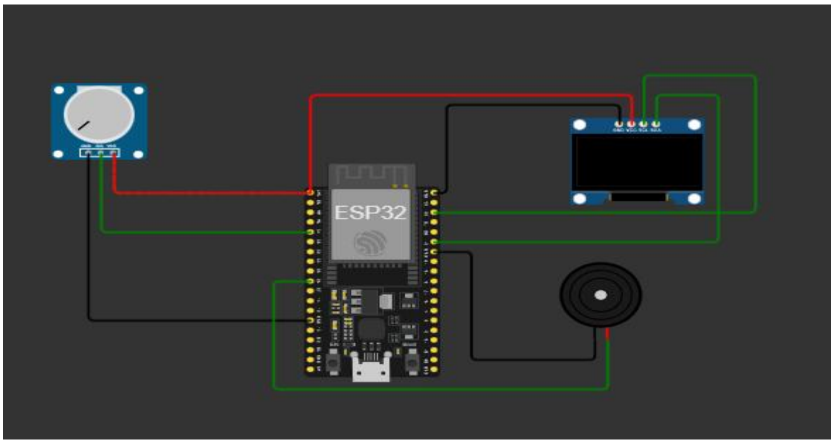

Figure 5: Diagramatic representation of project

CODE SNIPPET

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <WiFiClient.h>

// Define the analog pin for the sound sensor (simulated with a potentiometer)
#define SOUND_SENSOR_PIN 34

// Define the digital pin for the buzzer
#define BUZZER_PIN 26

// Set noise threshold (in decibels) for triggering the buzzer alert
#define NOISE_THRESHOLD 60

// OLED Display settings
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

// Create an OLED display object with the specified dimensions
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// WiFi Credentials (SSID and password)
const char* ssid = "Wokwi-GUEST";  // WiFi network name
const char* password = "";         // No password required for this network

// ThingSpeak API endpoint for sending noise level data
const char* server = "http://api.thingspeak.com/update?api_key=LC3VAKERDWOTKGV8";

void setup() {
    Serial.begin(115200);  // Initialize serial communication for debugging

    // Configure sound sensor pin as input and buzzer pin as output
    pinMode(SOUND_SENSOR_PIN, INPUT);
    pinMode(BUZZER_PIN, OUTPUT);
```

```
    // Initialize OLED Display
    if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
        Serial.println("OLED initialization failed");
        while (1);  // Halt execution if display fails to initialize
    }
```

```
    display.clearDisplay();  // Clear the screen buffer
    display.setTextSize(1);  // Set font size
    display.setTextColor(WHITE);  // Set text color
```

```
    // Connect to WiFi
    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi");

    // Wait until WiFi connection is established
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println(" Connected!");
}
```

```
void loop() {
    // Read the analog value from the sound sensor (simulated with a potentiometer)
    int sensorValue = analogRead(SOUND_SENSOR_PIN);
```

```
    // Map the sensor value (0-4095) to a noise level range (30-100 dB)
    float noiseLevel = map(sensorValue, 0, 4095, 30, 100);
```

```
    // Print the noise level to the Serial Monitor
    Serial.print("Noise Level: ");
    Serial.print(noiseLevel);
    Serial.println(" dB");
```

```
    // Display noise level on the OLED screen
    display.clearDisplay();  // Clear the display buffer
    display.setCursor(0, 10);  // Set cursor position
    display.print("Noise: ");
    display.print(noiseLevel);
    display.print(" dB");
    display.display();  // Update the display
```

```
    // Check if the noise level exceeds the threshold
    if (noiseLevel > NOISE_THRESHOLD) {
        Serial.println("Buzzer ON: Noise Too High!");
        tone(BUZZER_PIN, 1000);  // Activate the buzzer with a 1kHz tone
    }
    else {
        Serial.println("Buzzer OFF: Noise Normal");
        noTone(BUZZER_PIN);  // Deactivate the buzzer
    }
```

```
    // Send noise level data to ThingSpeak if WiFi is connected
    if (WiFi.status() == WL_CONNECTED) {
        HTTPClient http;
        String url = String(server) + "&field1=" + String(noiseLevel);  // Construct the API
request
        http.begin(url);  // Initialize HTTP request
        int httpResponseCode = http.GET();  // Send GET request to ThingSpeak
        http.end();  // Close HTTP connection
```

```
    }

    delay(2000);  // Wait for 2 seconds before the next reading
}
```

SIMULATION

Outputs from the Demonstration

1. Real-Time Noise Level Monitoring on OLED Display
The OLED screen successfully displayed real-time noise levels (in dB) based on the potentiometer's position.
Adjusting the potentiometer simulated different noise levels (e.g., increasing it represented louder environments).

From the simulation, the potentiometer was adjusted and the noise levels retrieved was 30dB
Noise: 30 dB
Noise: 40 dB
Noise: 52 dB
Noise: 100 dB
It was realised that, when the potentiomer was adjusted to a level where the noise level rose to 100dB, the buzzer sounded showing the threashold noise level of 60dB had been exceeded

2. WiFi Connectivity and Data Transmission to ThingSpeak
The ESP32 successfully connected to WiFi and sent noise level data to ThingSpeak.
This allowed real-time cloud monitoring of environmental noise pollution.
For every noise level detected, a unique HTTP request is sent to thinkspeak. The HTTP requests in the form, String url = String(server) + "&field1=" + String(noiseLevel); are;

http://api.thingspeak.com/update?api_key=LC3VAKERDWOTKGV8&field1=30
http://api.thingspeak.com/update?api_key=LC3VAKERDWOTKGV8&field1=40
http://api.thingspeak.com/update?api_key=LC3VAKERDWOTKGV8&field1=52
http://api.thingspeak.com/update?api_key=LC3VAKERDWOTKGV8&field1=100

RESULTS FROM THINKSPEAK

The following the graph is a graph from thinkspeak where you can see the noise levels have been tracked and logged since 24th of March 2025 by interfacfing with the ESP32 through IoT

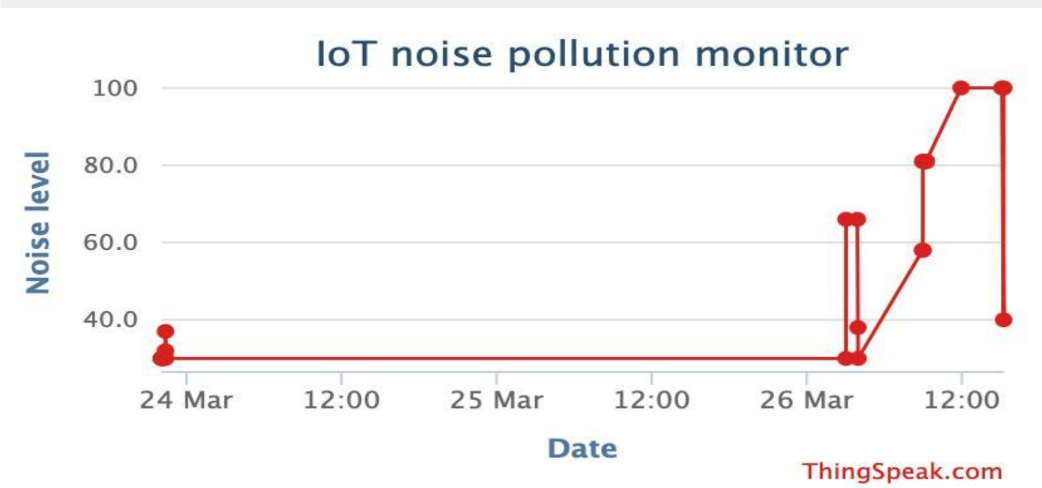

Figure 6: Screenshot of data accumulated over time in thinkspeak

Key Takeaways from the Simulation

- The System Effectively Simulated Noise Level Detection
- The OLED Display Provides Clear and Readable Outputs
- The Buzzer Alert Works as Expected
- The IoT Feature Successfully Sent Data to the Cloud
- The IoT Feature Successfully Sent Data to the Cloud
- Possible Improvements for a Real Deployment
- As the Voltage level from the potentiometer increases, the noise level increases

CONCLUSION

Summary of Findings

The simulation successfully demonstrated an IoT-based noise monitoring system using an ESP32 microcontroller. Key findings include:
✔ Real-time noise monitoring was accurately simulated using a potentiometer, which provided variable analog input.
✔ The OLED display effectively displayed the noise levels in decibels (dB).
✔ The buzzer alert system worked correctly, triggering an alarm when the noise level exceeded the set threshold (60 dB).
✔ WiFi connectivity enabled real-time data transmission to the ThingSpeak cloud platform, allowing remote monitoring of noise levels.
✔ The ESP32 proved to be an efficient microcontroller choice, offering WiFi, low power consumption, and multiple analog inputs for sensor integration.

Challenges Encountered

1. Lack of a Physical Sound Sensor
2. WiFi Connectivity Issues
3. Mapping Noise Levels Accurately
4. Limited Alert Mechanism

Possible Improvements

✅ 1. Use an Actual Sound Sensor
✅ 2. Enhance Alert System
✅ 3. Improve Cloud Integration
✅ 4. Optimize Power Consumption
✅ 5. Expand Functionality