PHASE 5

NOISE POLLUTION MONITORING

Definition:

Noise pollution is a growing problem in many urban areas, with negative impacts on human health, well-being, and quality of life. Real-time noise pollution monitoring and data sharing can help to raise awareness about the problem, enable informed decision-making, and support noise reduction efforts.

IoT Sensor Deployment:

The IoT sensor for this project is an ESP32 microcontroller with an I2S microphone. The ESP32 is a low-cost and powerful microcontroller that is well-suited for IoT applications. The I2S microphone is a high-quality microphone that is capable of measuring noise levels with a high degree of accuracy.

The ESP32 is programmed with a Python script that continuously measures the noise level and sends it to the server. The script also controls a buzzer and LED, which are used to indicate the noise level. The buzzer turns on and the LED lights up when the noise level exceeds a certain threshold.

The ESP32 is deployed in a location where the noise level is to be monitored. For example, it could be deployed in a city street, a construction site, or a factory.

Platform and Mobile App Development:

The noise pollution information platform is a web application that is developed using Python and the Django framework. The platform provides a user interface for viewing the noise level data collected from the ESP32. The platform also allows users to set noise level thresholds and receive notifications when the thresholds are exceeded.

The mobile app is developed using a cross-platform mobile development framework such as Flutter or React Native. The mobile app provides a user interface for viewing the noise level data collected from the ESP32 and receiving notifications when the noise level exceeds a certain threshold.

Code Implementation:

The code for the ESP32 is written in Python and uses the following libraries:

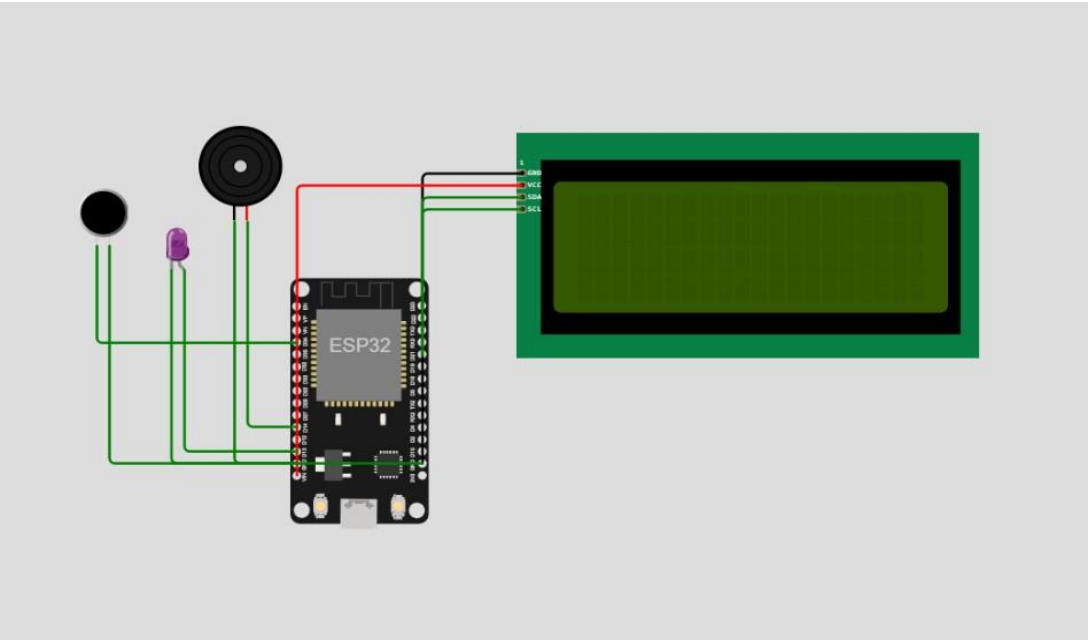
- machine: This library provides access to the ESP32's hardware resources, such as the ADC, PWM, and I2C interfaces.
- requests: This library provides a simple interface for making HTTP requests.
- Json: This library provides support for JSON data.

The code for the noise pollution information platform is written in Python and uses the Django framework. The code for the mobile app is written in Dart or JavaScript, depending on the mobile development framework that is used.

Components Required:

- ESP32 Board
- LCD Display
- Microphone Sensor
- Buzzer
- LED

Circuit Diagram:



Components:

Display:

Liquid Crystal Display (LCD) and the Internet of Things (IoT) are related in that LCDs are often used as display devices for IoT devices and systems. For example, LCD screens can be used to display real-time data from IoT sensors or to provide a user interface for controlling IoT devices.

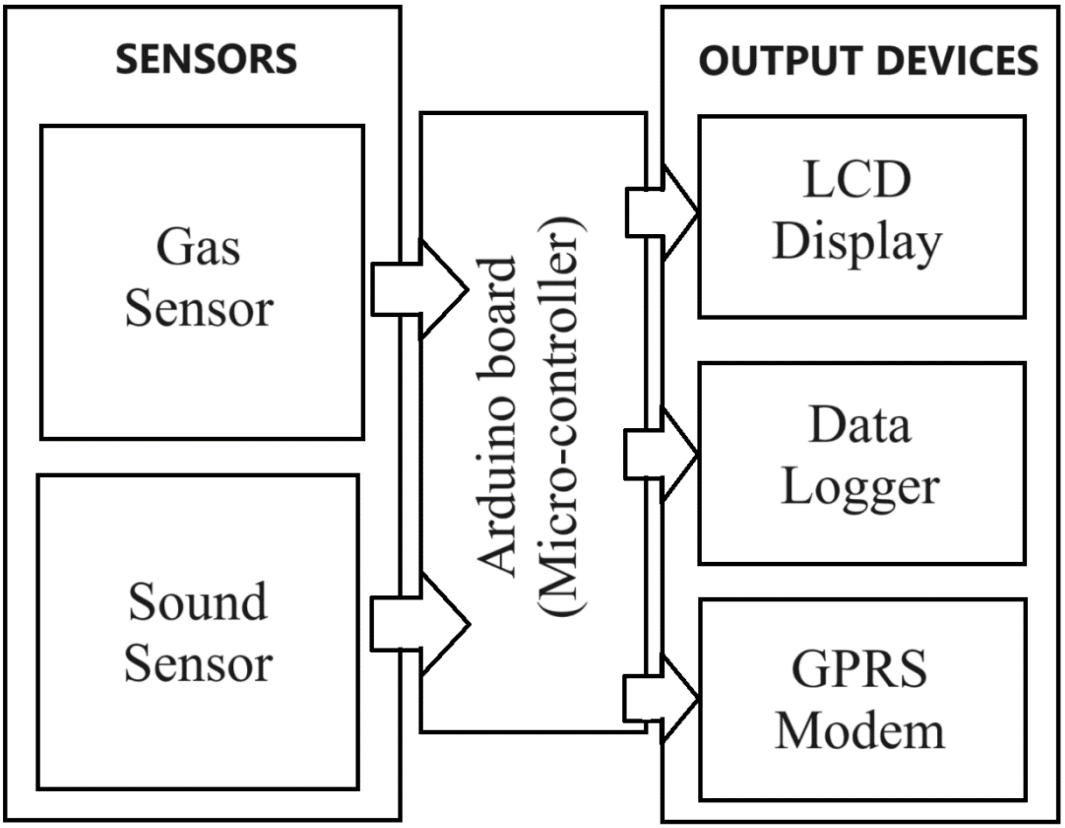
Sound sensor:

A sound sensor is an electronic device that detects sound waves, converting them into electrical signals, often used for applications like voice recognition, security systems, and noise monitoring.

ESP32:

ESP32 is a series of low-cost, low-power system on a chip microcontroller with integrated Wi-Fi and dual-mode Bluetooth.

Block Diagram:



Working:

1. Noise sensors are deployed at strategic locations in the area to be monitored. These sensors can be either standalone devices or integrated into IoT networks.
2. The sensors use microphones to detect sound waves and convert them into electrical signals.
3. The electrical signals are then processed by the sensors to determine the sound pressure level (SPL) in decibels (dB).
4. The SPL data is transmitted to a central data collection server via wired or wireless communication.
5. The data collection server stores and analyses the SPL data to identify trends and patterns.

6. The data collection server can also generate reports and alerts for stakeholders, such as government agencies, environmental groups, and businesses.

Some noise pollution monitoring systems also use sound dosimeters to measure the amount of noise exposure that people receive. Sound dosimeters are typically worn by individuals to track their noise exposure over time.

Python code:

```
import time
import math
from machine import ADC, Pin, I2C, PWM
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd
import network
import urequests as requests


# Define ADC pin for the microphone
mic_pin = ADC(Pin(34))


# Initialize I2C
i2c = I2C(0, sda=Pin(21), scl=Pin(22), freq=400000)


# Initialize LCD display with your specific settings
I2C_ADDR = 39
I2C_ROWS = 4
I2C_COLS = 20
lcd = I2cLcd(i2c, I2C_ADDR, I2C_ROWS, I2C_COLS)


# Define the buzzer and LED pins
```

```
buzzer_pin = Pin(14)
led_pin = Pin(13, Pin.OUT)

# Define the noise threshold in dB
noise_threshold = 60  # Adjusted to 60 dB

# Microphone sensitivity (in dB per Volt) - replace with your microphone's sensitivity
MIC_SENSITIVITY = 3.0

# Define your Wi-Fi credentials
WIFI_SSID = "Wokwi-GUEST"
WIFI_PASS = ""

# Initialize Wi-Fi
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(WIFI_SSID, WIFI_PASS)

# Wait until Wi-Fi connection is established
while not wifi.isconnected():
    pass

# Create a PWM object for the buzzer
buzzer_pwm = PWM(buzzer_pin)
buzzer_pwm.deinit()  # Turn off the buzzer at the beginning
```

```
# Define a function to calculate sound pressure level (SPL)
def calculate_noise_level(adc_value, reference_voltage, sensitivity):
    # Calculate noise level in decibels (dB) based on sensitivity and voltage
    noise_db = 20 * math.log10(adc_value / (reference_voltage * sensitivity))
    return noise_db

# Define a function to update noise level and control the buzzer and LED
def update_noise_level():
    global noise_level

    # Read ADC values from the microphone
    mic_value = mic_pin.read()

    # Set a constant reference voltage based on your system's maximum voltage
    reference_voltage = 5.0  # Assuming 5V maximum reference voltage

    # Calculate noise level
        noise_level = calculate_noise_level(mic_value, reference_voltage, MIC_SENSITIVITY)  # Update noise_level directly

    # Control the buzzer and LED based on the noise level
    if noise_level > noise_threshold:
        # Turn on the buzzer and LED
        buzzer_pwm.freq(1000)
        led_pin.on()
    else:
        # Turn off the buzzer and LED
```

```
        buzzer_pwm.deinit()
        led_pin.off()

# Define a function to display noise level status
def display_noise_status():
    if noise_level < noise_threshold - 10:
        status = "Quiet"
    elif noise_level >= noise_threshold - 10 and noise_level < noise_threshold + 10:
        status = "Normal"
    else:
        status = "High"

    # Display noise level status and the actual noise level on the LCD and serial monitor
    lcd.clear()
    lcd.move_to(0,0)
    lcd.putstr("Loudness: {:.2f}dB".format(noise_level))
    lcd.move_to(0,2)
    lcd.putstr("Level: {}".format(status))

    print("Loudness: {:.2f} dB".format(noise_level))
    print("Level: {}".format(status))

# Main loop with continuous data transmission
first_data_sent = False  # Track if the first data is sent

while True:
```

```
    # Update noise level
    update_noise_level()

    # Display noise level status
    display_noise_status()

    # Send data to your server continuously after the first data
    if first_data_sent:
        data = {
            "noise_level": noise_level
        }
        response = requests.post(SERVER_URL, json=data)

    if not first_data_sent:
        first_data_sent = True
# Mark the first data as sent
```

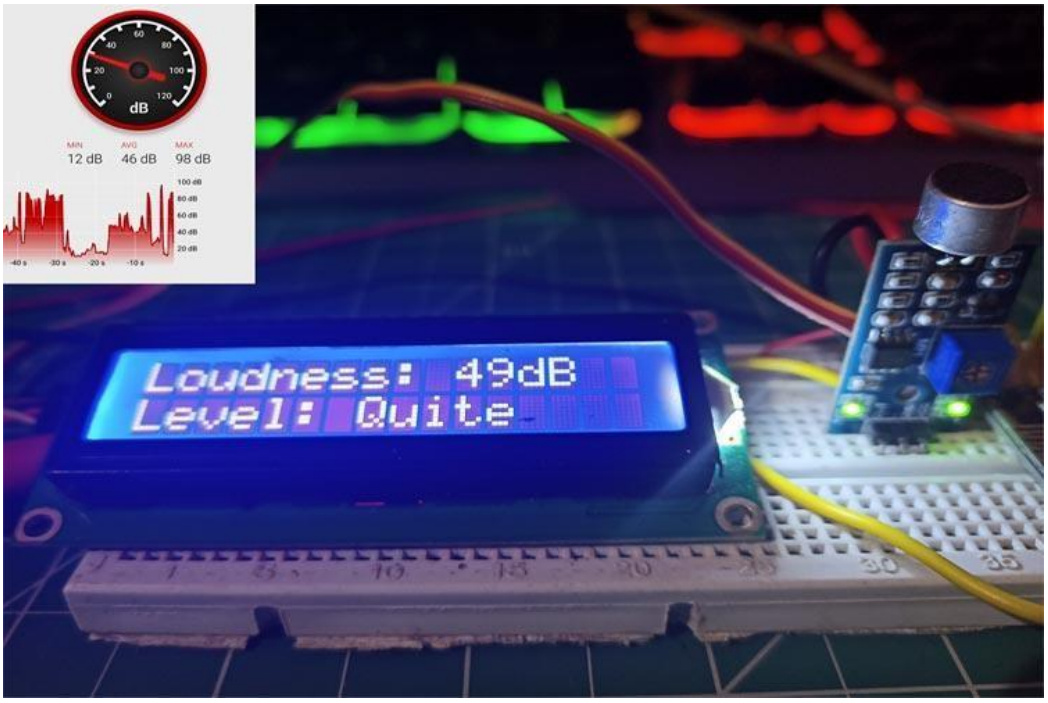
Output of Program:

The output of a noise pollution monitoring project using an ESP32 can be in a variety of formats, depending on the specific needs of the project. Some common output formats include:

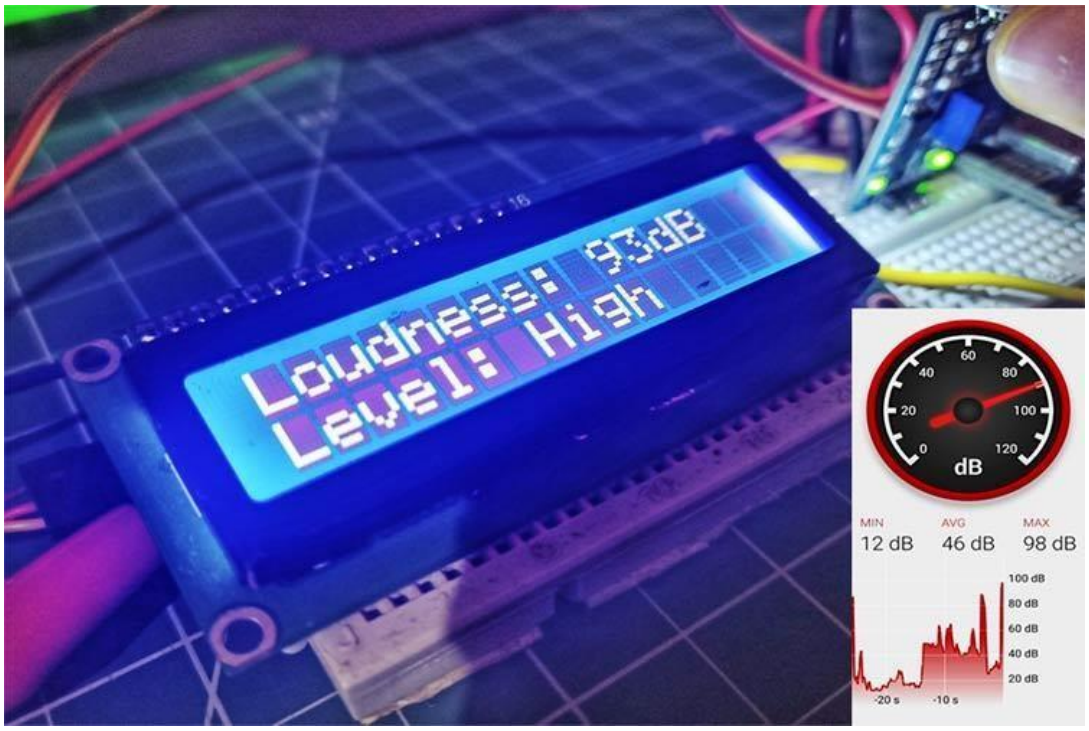
- Real-time noise level data: This is the most common type of output, and it consists of a continuous stream of noise level data sent to a server or other device. The data can be in any format, but it is typically sent in JSON or CSV format.
- Noise level alerts: This type of output is generated when the noise level exceeds a certain threshold. The alert can be sent to a variety of devices, such as a smartphone, email, or SMS.

- Noise level reports: This type of output is generated over a period of time, such as a day, week, or month. The report can include statistics such as the average noise level, the maximum noise level, and the number of noise level alerts that were generated.

Quite:



High:



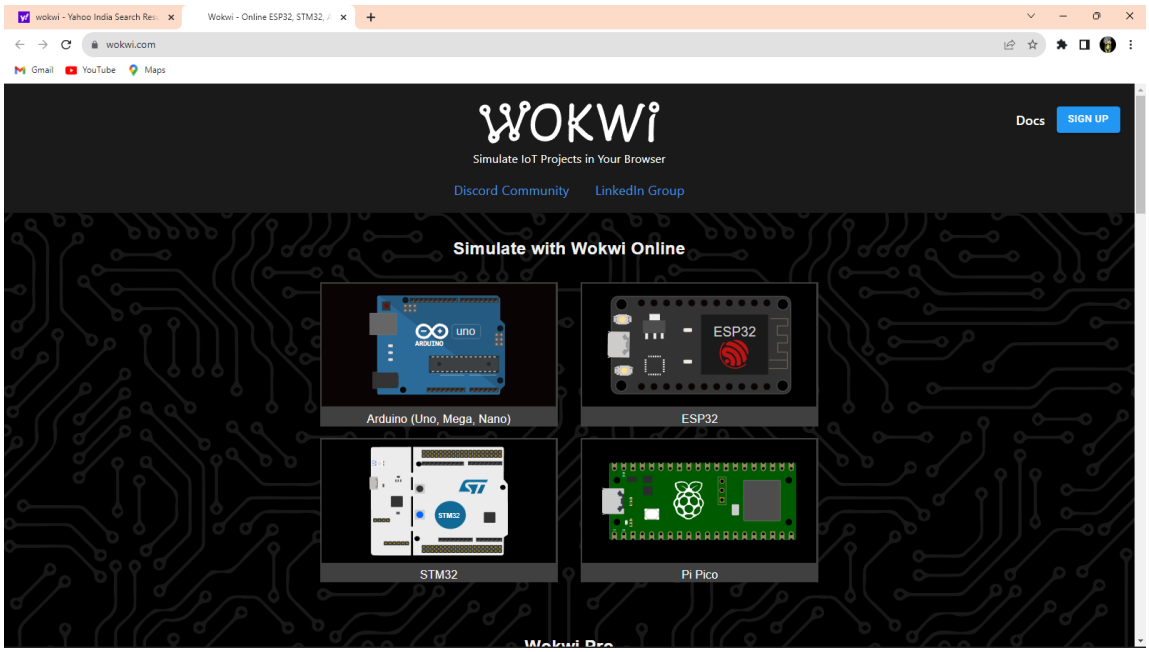
The specific output format that you choose will depend on the needs of your project and the devices that you are using. For example, if you are building a noise pollution monitoring system for a city, you might want to send the real-time noise level data to a server that can be used to generate a noise level map. If you are building a noise pollution monitoring system for a home, you might want to receive noise level alerts on your smartphone.

Wokwi Simulation Platform:

An IoT simulation platform is a software tool that allows you to create and test IoT devices and systems without having to use physical hardware. This can be useful for a variety of reasons, such as:

- Testing new IoT devices and systems: You can use an IoT simulation platform to test new IoT devices and systems before you deploy them in the real world. This can help you to identify and fix any problems before they cause any disruptions.
- Prototyping IoT solutions: An IoT simulation platform can be used to prototype IoT solutions before you develop the actual hardware and software. This can help you to validate your ideas and to get feedback from users early on.
- Training and education: An IoT simulation platform can be used to train and educate people about IoT devices and systems. This can be useful for students, engineers, and other professionals who need to learn about IoT.

IoT simulation platforms typically work by creating a virtual representation of the IoT device or system that you want to test. This virtual representation can then be used to simulate the behaviour of the device or system in the real world.



WEBSITE LINK:

Web: https://wokwi.com/

Promoting Public Awareness and Contributing to Noise Pollution Mitigation

The real-time noise pollution monitoring system promotes public awareness of noise pollution by providing users with real-time information about the noise level in their environment. This information can help users to make informed decisions about their exposure to noise pollution. For example, a user may choose to avoid a noisy area or take steps to reduce their noise exposure.

The system also contributes to noise pollution mitigation by providing users with the ability to set noise level thresholds and receive notifications when the thresholds are exceeded. This information can help users to take action to reduce noise pollution in their environment. For example, a user may contact the authorities to complain about a noisy establishment.

Output web:

Link: **https://wokwi.com/projects/379487417096095745**

Conclusion:

Noise pollution monitoring is an important issue, and the ESP32 microcontroller is a powerful tool that can be used to monitor noise levels in real time. By using an ESP32 and a sound sensor, you can create a noise pollution monitoring system that can be used to improve the quality of life in your community.

THANK YOU