NOISE POLLUTION MONITORING

DEVELOPMENT PHASE

Noise pollution is a growing concern in urban and industrial environments, impacting human health, wildlife, and overall quality of life. To address this issue effectively, monitoring and analyzing noise levels in real-time are essential.

The Internet of Things (IoT) offers a powerful solution by integrating sensor technologies, data analytics, and connectivity to create a comprehensive noise pollution monitoring system. This report explores the key aspects of noise pollution monitoring in IoT

Building an IoT noise pollution monitoring system involves both hardware and software components

Hardware Components:

1. **Sound Sensor/Microphone:**
    - Captures ambient noise levels.
2. **Microcontroller:**
    - Arduino (e.g., Arduino Uno, Arduino Nano) or other IoT-compatible microcontrollers (e.g., ESP32).
3. **Connectivity Module:**
    - Wi-Fi Module (e.g., ESP8266 for Arduino, built-in Wi-Fi in ESP32) or GSM Module for cellular connectivity.

4. **Power Supply:**
    - Depending on the deployment scenario, you might use batteries, solar panels, or a wired power source.
5. **Enclosure:**
    - Protects the components from environmental conditions (waterproof, if necessary).
6. **Cloud Platform:**
    - Choose a cloud platform for data storage, analysis, and remote access (e.g., AWS IoT, Azure IoT, Google Cloud IoT).
7. **User Interface:**
    - Development tools and platforms for creating a user interface (web or mobile app) for users to monitor noise levels.
8. **Alerting System:**
    - Components for notifications or alerts when noise levels exceed predefined thresholds (e.g., LEDs, buzzers).

SOFTWARE COMPONENTS:

1.Arduino IDE:

- Used to program the microcontroller. Write and upload firmware to read data from sensors and handle communication. While the code itself is written in C/C++, Python can be used for serial communication and data processing on the computer side.

2. PySerial:

- A Python library that allows communication with serial ports. Useful for reading data from the Arduino through the USB connection.

Iot platform:

- Choose an IoT platform for data storage and analysis. Common choices include AWS IoT, Azure IoT, Google Cloud IoT, or open-source platforms like MQTT with NodeRED.

Python Script :

- Develop a Python script to read sensor data, process it, and send it to the IoT platform. This script will run on the microcontroller.

Model Implementation:

1. Connect the Sound Sensor:

Connect the sound sensor to the analog input pin of the Arduino board. Ensure proper wiring and connections.

2. Set Up the Arduino Code:

Write a simple Arduino sketch to read data from the sound sensor. Sample code might look like this

Python code

```
import serial

import matplotlib.pyplot as plt

from drawnow import drawnow


# Initialize variables

noise_levels = []

max_data_points = 20  # Number of data points to display on the plot
```

```
# Create a function to update the plot
def plot_noise_levels():
    plt.title('Noise Pollution Monitoring')
    plt.ylabel('Noise Level')
    plt.xlabel('Time')
    plt.grid(True)
    plt.plot(noise_levels, label='Noise Level')
    plt.legend(loc='upper left')

# Connect to Arduino via serial port
ser = serial.Serial('COM3', 9600)  # Update 'COM3' with the correct serial port

# Continuous loop to read data from Arduino and update the plot
while True:
    # Read data from Arduino
    data = ser.readline().decode('utf-8').strip()

    # Append data to the list
    noise_levels.append(int(data))

    # Keep only the last 'max_data_points' data points
    noise_levels = noise_levels[-max_data_points:]

    # Call drawnow to update the plot
```

drawnow(plot_noise_levels)

3. Add Connectivity:

Integrate a Wi-Fi or GSM module to the Arduino for data transmission. Modify the code to include connectivity features.

4. Set Up Cloud Platform:

Choose a cloud platform (e.g., AWS, Azure, Google Cloud) and set up an IoT application to receive and store noise data. Implement security measures for data protection.

5. Develop User Interface:

Create a user interface (web or mobile app) to visualize real-time noise levels and historical data. The interface can interact with the cloud platform to fetch and display data.

6. Power Management:

Implement power management techniques to optimize energy consumption, especially in remote or battery-powered scenarios.

7. Deployment:

Install the noise monitoring system in the desired locations. Ensure proper protection from environmental conditions. **Data Transmission (Optional):**

If using a Wi-Fi module, you can transmit the noise data to a cloud platform or server for storage and analysis. This might involve setting up an MQTT or HTTP connection.

8.Visualization and Analysis:

Develop a user interface (web or mobile app) to visualize real-time noise levels and historical data. This could involve integrating with a cloud platform or building a local system for data analysis.

9.Enclosure and Deployment:

Protect the components in a suitable enclosure to withstand environmental conditions. Deploy the system in the desired monitoring locations.

Testing:

Conduct thorough testing to ensure the system's reliability, accuracy, and connectivity. Adjust the model as needed based on testing results.

Please note that this is a simplified example, and real-world implementations may require additional considerations such as calibration, data security, and adherence to local regulations. Depending on the complexity of your project, you might also explore more advanced sensors, microcontrollers, and cloud-based analytics.

SUBMITTED BY:

MENTOR: S Abikayil Aarthi AP/CSE

TEAM MEMBERS:

Aandal SA-au821121104002

Keerthana J-au821121104026

Anees Priyanka V-au821121104004

Shalini v-au821121104050

Gayathiri S-au821121104014