NOISE POLLUTION MONITORING
IOT_PHASE_3

REG NO:610821106008
NAME:ARVIND Y

1 ` **Identify Sensor Locations:**

- Identify public areas where noise levels need monitoring, such as parks, busy streets, or event venues.
- Consider local regulations and guidelines for noise monitoring.

2. **Define Objectives:**
   - Clearly define the objectives of noise monitoring. This could be to assess the impact on public health, enforce noise regulations, or analyze trends over time.

Technology Selection:

3. **Choose Noise Sensors:**
   - Select appropriate noise sensors that meet your requirements. Consider factors like accuracy, frequency range, and connectivity options (Wi-Fi, cellular, LPWAN).
4. **Communication Infrastructure:**
   - Decide on the communication infrastructure for the sensors. Options include Wi-Fi, cellular networks, or Low-Power Wide-Area Network (LPWAN) technologies like LoRa or NB-IoT.

Implementation:

5. **Power Supply:**
   - Determine the power supply for the sensors. Options include battery power, solar panels, or a combination of both. Ensure that the chosen power source is reliable for continuous operation.
6. **Sensor Deployment:**
   - Install sensors in identified locations. Consider weatherproofing and security measures to protect the sensors from environmental conditions and vandalism.
7. **Connectivity Setup:**
   - Configure the connectivity for the sensors. Ensure that they can transmit data reliably to a central data storage or processing system.
8. **Data Storage and Processing:**
   - This could be on a cloud platform or a local server, depending on the scale of your deployment.

Data Analysis and Visualization:

9. **Data Analysis Tools:**
    - Implement tools and algorithms for analyzing noise data. This could involve identifying noise trends, peak hours, and compliance with noise regulations.
10. **Visualization Platform:**
    - Develop a user-friendly platform for visualizing the data. Consider creating dashboards that provide real-time information and historical trends.

Maintenance and Monitoring:

11. **Maintenance Plan:**
    - Establish a regular maintenance plan to ensure the sensors are functioning correctly. This includes checking the power supply, updating firmware, and replacing faulty sensors.
12. **Security Measures:**
    - Implement security measures to protect the sensors and data from unauthorized access. This is crucial to maintain the integrity and privacy of the collected information.

Compliance and Communication:

13. **Regulatory Compliance:**
    - Ensure that your noise monitoring system complies with local regulations and privacy laws.
14. **Public Communication:**
    - Communicate the purpose of the noise monitoring system to the public. Transparency helps build trust and may encourage compliance with noise regulations.
    -

Please note that you need to replace the placeholders (e.g., `YOUR_SENSOR_ID`, `YOUR_API_KEY`, `API_ENDPOINT`) with your actual credentials and endpoints.

Python code:

```
import time
import random
import paho.mqtt.client as mqtt
import requests

# Replace these values with your actual credentials and endpoints
SENSOR_ID = "YOUR_SENSOR_ID"
API_KEY = "YOUR_API_KEY"
```

```
API_ENDPOINT = "https://your-api-endpoint.com/data"

# MQTT Configuration
MQTT_BROKER = "mqtt.eclipse.org"  # Replace with your MQTT broker address
MQTT_PORT = 1883
MQTT_TOPIC = "noise_sensor_data"

# Simulated Noise Sensor Data
def generate_noise_data():
    return round(random.uniform(50, 80), 2)  # Replace with your actual noise data source

# MQTT Callbacks
def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))
    client.subscribe(MQTT_TOPIC)

def on_message(client, userdata, msg):
    noise_level = float(msg.payload.decode())
    print(f"Received noise level: {noise_level} dB")

    # Send data to the API endpoint
    send_data_to_api(noise_level)

def send_data_to_api(noise_level):
    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Bearer {API_KEY}",
    }

    data = {
        "sensor_id": SENSOR_ID,
        "noise_level": noise_level,
        "timestamp": int(time.time()),
    }

    try:
        response = requests.post(API_ENDPOINT, json=data, headers=headers)
        if response.status_code == 200:
            print("Data sent successfully")
        else:
            print(f"Failed to send data. Status code: {response.status_code}")
    except Exception as e:
        print(f"Error sending data: {str(e)}")

# MQTT Client Setup
mqtt_client = mqtt.Client()
mqtt_client.on_connect = on_connect
mqtt_client.on_message = on_message

# Connect to MQTT broker
mqtt_client.connect(MQTT_BROKER, MQTT_PORT, 60)

# Start the MQTT loop
mqtt_client.loop_start()

try:
```

```
    while True:
        # Simulate noise data and publish to MQTT
        noise_data = generate_noise_data()
        mqtt_client.publish(MQTT_TOPIC, str(noise_data))
        print(f"Published noise level: {noise_data} dB")
        time.sleep(5)  # Adjust the interval based on your requirements

except KeyboardInterrupt:
    print("Script terminated by user.")
    mqtt_client.disconnect()
```

Explanation:

- The script uses the **paho.mqtt.client** library for MQTT communication. Install it using **pip install paho-mqtt**.
- Replace the placeholders (**YOUR_SENSOR_ID**, **YOUR_API_KEY**, **API_ENDPOINT**, etc.) with your actual values.
- The **generate_noise_data** function simulates the noise level data. Replace it with the actual function or sensor data source.
- The script subscribes to an MQTT topic (**noise_sensor_data**) to receive real-time noise data.
- Upon receiving the data, the script sends it to the specified API endpoint using an HTTP POST request.