The Python script for an IoT noise sensor that sends real-time noise level data to a noise pollution information platform using the MQTT protocol:

# Description:

1. \*Importing Libraries\*: The script starts by importing the necessary libraries. It uses the paho.mqtt.client library for MQTT communication.

2. \*MQTT Broker Configuration\*: You need to configure the MQTT broker's address and port. Replace "mqtt.yourplatform.com" with the actual address of your MQTT broker, and adjust the port if necessary. The script will publish noise level data to a specific MQTT topic, defined as topic.

3. \*Simulating Noise Sensor Data\*: The read\_noise\_sensor function simulates reading noise sensor data. In this example, it generates random noise level data between 40 and 90 dB. You should replace this with actual logic to read data from your IoT noise sensor.

4. \*MQTT Client Setup\*: The script sets up an MQTT client, assigns an on\_connect callback function, and connects to the MQTT broker. The on\_connect function handles the connection status, printing a message when it successfully connects to the broker.

5. \*Data Transmission Loop\*: Within a while loop, the script reads noise sensor data using the read\_noise\_sensor function. It then publishes this data to the specified MQTT topic using the client.publish method. The data includes the noise level in decibels (dB).

6. \*Printing Data\*: The script also prints the data to the console to show that it's being published successfully.

7. \*Delay\*: After each data transmission, the script sleeps for 10 seconds (you can adjust this delay to match your desired data transmission frequency). This simulates sending data periodically. In a real-world scenario, you'd likely have your IoT sensor read data at specific intervals.

8. \*Running the Script\*: You can run this script on the IoT device with the noise sensor. It will continuously read simulated or real noise sensor data and send it to the MQTT broker. You should configure your noise pollution information platform to subscribe to the same MQTT topic, allowing it to receive and process the incoming data for noise pollution monitoring and analysis.

***Source Code***:

Main.py:

import machine

import time

import urequests

import ujson

import network

import math

# Define your Wi-Fi credentials

wifi\_ssid = 'Wokwi-GUEST'

wifi\_password = ''  # Replace with the actual Wi-Fi password

# Connect to Wi-Fi

wifi = network.WLAN(network.STA\_IF)

wifi.active(True)

wifi.connect(wifi\_ssid, wifi\_password)

# Wait for Wi-Fi connection

while not wifi.isconnected():

    pass

# Define ultrasonic sensor pins (Trig and Echo pins)

ultrasonic\_trig = machine.Pin(15, machine.Pin.OUT)

ultrasonic\_echo = machine.Pin(4, machine.Pin.IN)

# Define microphone pin

microphone = machine.ADC(2)

calibration\_constant = 2.0

noise\_threshold = 60  # Set your desired noise threshold in dB

# Firebase Realtime Database URL and secret

firebase\_url = 'https://noise-pollution-bd0ab-default-rtdb.asia-southeast1.firebasedatabase.app/'  # Replace with your Firebase URL

firebase\_secret = 'nBsgyQFTqHUe4qExlaZX6VL3mpf5gn6BlpnMiuR0'  # Replace with your Firebase secret

def measure\_distance():

    # Trigger the ultrasonic sensor

    ultrasonic\_trig.value(1)

    time.sleep\_us(10)

    ultrasonic\_trig.value(0)

    # Measure the pulse width of the echo signal

    pulse\_time = machine.time\_pulse\_us(ultrasonic\_echo, 1, 30000)

    # Calculate distance in centimeters

    distance\_cm = (pulse\_time / 2) / 29.1

    return distance\_cm

def measure\_noise\_level():

    # Read analog value from the microphone

    noise\_level = microphone.read()

    noise\_level\_db = 20 \* math.log10(noise\_level / calibration\_constant)

    return noise\_level, noise\_level\_db

# Function to send data to Firebase

def send\_data\_to\_firebase(distance, noise\_level\_db):

    data = {

        "Distance": distance,

        "NoiseLevelDB": noise\_level\_db

    }

    url = f'{firebase\_url}/sensor\_data.json?auth={firebase\_secret}'

    try:

        response = urequests.patch(url, json=data)  # Use 'patch' instead of 'put'

        if response.status\_code == 200:

            print("Data sent to Firebase")

        else:

            print(f"Failed to send data to Firebase. Status code: {response.status\_code}")

    except Exception as e:

        print(f"Error sending data to Firebase: {str(e)}")

try:

    while True:

        distance = measure\_distance()

        noise\_level, noise\_level\_db = measure\_noise\_level()

        print("Distance: {} cm, Noise Level: {:.2f} dB".format(distance, noise\_level\_db))

        if noise\_level\_db > noise\_threshold:

            print("Warning: Noise pollution exceeds threshold!")

        # Send data to Firebase

        send\_data\_to\_firebase(distance, noise\_level\_db)

        time.sleep(1)  # Adjust the sleep duration as needed

except KeyboardInterrupt:

    print("Monitoring stopped")

diagram.json:

{

  "version": 1,

  "author": "Gokul Raja",

  "editor": "wokwi",

  "parts": [

    {

      "type": "wokwi-esp32-devkit-v1",

      "id": "esp",

      "top": -72.1,

      "left": 52.6,

      "attrs": { "env": "micropython-20231005-v1.21.0" }

    },

    { "type": "wokwi-microphone", "id": "mic", "top": -16.98, "left": 263.79, "attrs": {} },

    {

      "type": "wokwi-hc-sr04",

      "id": "ultrasonic1",

      "top": -190.5,

      "left": 274.3,

      "attrs": { "distance": "88" }

    }

  ],

  "connections": [

    [ "esp:TX0", "$serialMonitor:RX", "", [] ],

    [ "esp:RX0", "$serialMonitor:TX", "", [] ],

    [ "mic:1", "esp:D2", "purple", [ "v0" ] ],

    [ "mic:2", "esp:GND.1", "black", [ "v0" ] ],

    [ "ultrasonic1:VCC", "esp:3V3", "red", [ "v0" ] ],

    [ "ultrasonic1:TRIG", "esp:D15", "yellow", [ "v0" ] ],

    [ "ultrasonic1:ECHO", "esp:D4", "green", [ "v0" ] ],

    [ "ultrasonic1:GND", "esp:GND.1", "black", [ "v0" ] ]

  ],

  "serialMonitor": { "display": "plotter" },

  "dependencies": {}

}

Output:

