NOISE POLLUTION MONITORING USING IOT

PHASE 3: DEVELOPMENT PART -1:

ALGORITHM:

1. Start

2. Gather Data:
   - Collect noise data from sensors (e.g., microphones).
   - Convert analog data to digital if needed.

3. Preprocess Data:
   - Filter out irrelevant noise.
   - Normalize data if necessary.
4. Analyze Data:
   - Calculate noise levels..
   - Compare to predefined thresholds.

5. Decision Point:
   - Is the noise level above the threshold?
     - Yes: Proceed to the next step.
     - No: Continue monitoring.

6. Alert Generation:
   - Generate an alert or notification.
   - Specify the type and severity of the alert.

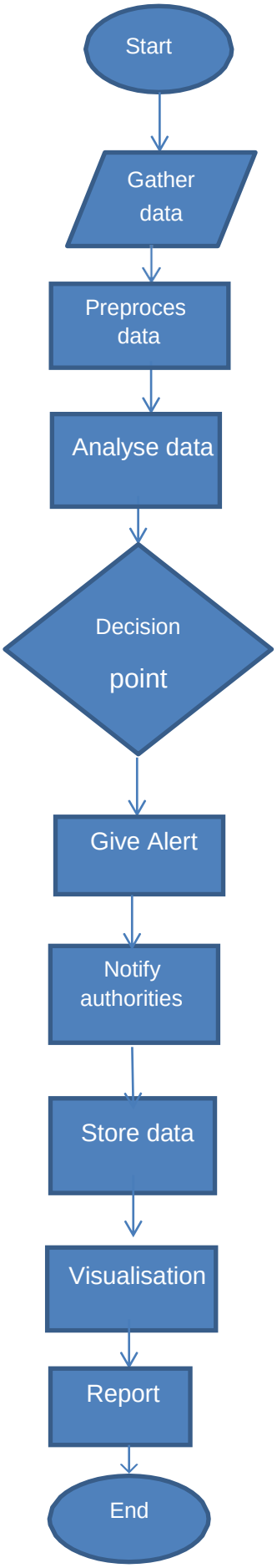
7. Notify Authorities:
   - Send alerts to relevant authorities Or individuals

8. 8. Store Data:
   - Save historical noise data for analysis and reporting.

9. Visualization:
   - Generate graphs or charts for noise trends.
   - Provide a user interface for real-time monitoring

10. Reporting:
    - Generate noise pollution reports periodically or on demand.
11. END

FLOW CHART:


Start
Gather data
Preproces data
Analyse data
Decision point
Give Alert
Notify authorities
Store data
Visualisation
Report
End

PYTHON SCRIPT

```
python
# Import necessary libraries for IoT components
import RPi.GPIO as GPIO  # For Raspberry Pi GPIO
import Adafruit_DHT  # For DHT temperature and humidity sensor
import Adafruit_BMP.BMP085 as BMP085  # For BMP085 barometric pressure sensor
import Adafruit_ADS1x15  # For ADC (Analog to Digital Converter)
import smbus  # For I2C sensors like GPS
import time  # For timing and sleep
from gpiozero import Button  # For remote sensor button
import os  # For system commands

# Pin definitions
MICROPHONE_PIN = 17  # Example pin for a microphone
SOUND_LEVEL_METER_PIN = 18  # Pin for a sound level meter
ACOUSTIC_SENSOR_PIN = 22  # Pin for an acoustic sensor
GPS_SDA_PIN = 2  # SDA pin for GPS sensor (I2C)
GPS_SCL_PIN = 3  # SCL pin for GPS sensor (I2C)
DATA_LOGGER_PIN = 23  # Pin for data logger (e.g., SD card reader)
REMOTE_SENSOR_PIN = 24  # Pin for a remote sensor (e.g., button)

# Initialize GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(MICROPHONE_PIN, GPIO.IN)
GPIO.setup(SOUND_LEVEL_METER_PIN, GPIO.IN)
GPIO.setup(ACOUSTIC_SENSOR_PIN, GPIO.IN)
# Initialize other sensors here...

# Create sensor objects
dht_sensor = Adafruit_DHT.DHT22
bmp_sensor = BMP085.BMP085()
adc = Adafruit_ADS1x15.ADS1115()
bus = smbus.SMBus(1)  # I2C bus
remote_sensor = Button(REMOTE_SENSOR_PIN)

# Main loop
try:
    while True:
        # Read from microphones, sound level meter, acoustic sensor
        microphone_data = GPIO.input(MICROPHONE_PIN)
        sound_level_meter_data = GPIO.input(SOUND_LEVEL_METER_PIN)
        acoustic_sensor_data = GPIO.input(ACOUSTIC_SENSOR_PIN)

        # Read from DHT temperature and humidity sensor
        humidity, temperature = Adafruit_DHT.read_retry(dht_sensor, DHT_PIN)

        # Read barometric pressure
        pressure = bmp_sensor.read_pressure()

        # Read data from ADC
        adc_value = adc.read_adc(0, gain=1)  # Read from ADC channel 0
```

```
        # Read GPS data (using I2C)
        gps_data = bus.read_byte_data(device_address, register)

        # Read data from the remote sensor
        if remote_sensor.is_pressed:
            # Handle remote sensor press

        # Log data to a data logger
        with open("data_log.txt", "a") as log_file:
            log_file.write(f"{time.time()}, {microphone_data}, {sound_level_meter_data},
{acoustic_sensor_data}, {humidity}, {temperature}, {pressure}, {adc_value},
{gps_data}\n")

        # Sleep for a defined interval
        time.sleep(1)  # Adjust the interval as needed

except KeyboardInterrupt:
GPIO.cleanup()  # Cleanup GPIO on program exit
```

PYTHON SCRIPT FOR MICROPHONE:

```
python
import
os
import subprocess

def record_audio(file_name, duration=10):
    # Use the arecord command to record audio
    cmd = f"arecord -d {duration} -f cd -t wav
    {file_name}" subprocess.call(cmd, shell=True)

def upload_to_server(file_name):
    # You can implement code here to upload the recorded audio to a server or cloud
    storage.

if __name__== "_main_":
    audio_file =
    "sample_audio.wav"
    record_duration = 10 #
    seconds
    record_audio(audio_file,
    record_duration)
    upload_to_server(audio_file)
```

PYTHON SCRIPT FOR SOUND LEVEL METER:

```
python
import os
import
time
import numpy as
np import pyaudio
import requests

# Configure your ThingSpeak channel details
THINGSPEAK_API_KEY =
'YOUR_THINGSPEAK_API_KEY'
THINGSPEAK_CHANNEL_ID =
'YOUR_THINGSPEAK_CHANNEL_ID'

# Initialize PyAudio
audio = pyaudio.PyAudio()

# Set up audio
stream CHUNK =
1024
FORMAT =
pyaudio.paInt16
CHANNELS = 1
RATE = 44100

stream = audio.open(format=FORMAT, channels=CHANNELS, rate=RATE,
input=True, frames_per_buffer=CHUNK)

while
  True:
  try:
    # Read audio data from the
    microphone data =
    stream.read(CHUNK)
    audio_data = np.frombuffer(data, dtype=np.int16)

    # Calculate sound level (change this calculation as
    needed) rms = np.sqrt(np.mean(audio_data**2))
    sound_level = 20 * np.log10(rms)

    # Print sound level to the console
    print(f"Sound Level (dB):
    {sound_level}")
```

```
# Send data to ThingSpeak
if THINGSPEAK_API_KEY and THINGSPEAK_CHANNEL_ID:
```

```
        data = {'api_key': THINGSPEAK_API_KEY, 'field1':
        sound_level} response =
        requests.post(f'https://api.thingspeak.com/update', data)
        print(f"ThingSpeak Response: {response.status_code}")

    except
      KeyboardInterrupt:
      break

# Cleanup
stream.stop_stream
() stream.close()
audio.terminate()
```

Make sure to replace `'YOUR_THINGSPEAK_API_KEY'` and `'YOUR_THINGSPEAK_CHANNEL_ID'` with your actual ThingSpeak channel details. Additionally, you may need to adjust the sound level calculation based on your microphone and requirements.

.

PYTHON SCRIPT FOR ACOUSTIC SENSOR:

```
python
import
time
import numpy as
np import pyaudio
import requests

# Configure your ThingSpeak channel details
THINGSPEAK_API_KEY =
'YOUR_THINGSPEAK_API_KEY'
THINGSPEAK_CHANNEL_ID =
'YOUR_THINGSPEAK_CHANNEL_ID'

# Initialize PyAudio
audio = pyaudio.PyAudio()

# Set up audio
stream CHUNK =
1024
FORMAT =
pyaudio.paInt16
CHANNELS = 1
RATE = 44100
```

```
stream = audio.open(format=FORMAT, channels=CHANNELS, rate=RATE,
input=True, frames_per_buffer=CHUNK)

while
   True:
   try:
      # Read audio data from the
      microphone data =
      stream.read(CHUNK)
      audio_data = np.frombuffer(data, dtype=np.int16)

      # Calculate some acoustic feature (e.g., peak
      amplitude) acoustic_feature =
      np.max(np.abs(audio_data))

      # Print the acoustic feature to the console
      print(f"Acoustic Feature:
      {acoustic_feature}")

      # Send data to ThingSpeak
      if THINGSPEAK_API_KEY and THINGSPEAK_CHANNEL_ID:
         data = {'api_key': THINGSPEAK_API_KEY, 'field1':
         acoustic_feature} response =
         requests.post(f'https://api.thingspeak.com/update', data)
         print(f"ThingSpeak Response: {response.status_code}")

   except
      KeyboardInterrupt:
      break

# Cleanup
stream.stop_stream
() stream.close()
audio.terminate()
```

PYTHON SCRIPT FOR WEATHER SENSOR:

```
python
import
time
import requests
import Adafruit_DHT # For DHT temperature and humidity sensor

# Configure your ThingSpeak channel details
THINGSPEAK_API_KEY =
'YOUR_THINGSPEAK_API_KEY'
```

```
THINGSPEAK_CHANNEL_ID =
'YOUR_THINGSPEAK_CHANNEL_ID'
```

```
# Specify your sensor type (DHT11, DHT22,
AM2302) SENSOR_TYPE =
Adafruit_DHT.DHT22
SENSOR_PIN = 4 # GPIO pin where the sensor is connected

while
   True:
   try:
      # Read data from the sensor
      humidity, temperature = Adafruit_DHT.read_retry(SENSOR_TYPE,
      SENSOR_PIN)

      if humidity is not None and temperature is not
         None: # Print sensor data to the console
         print(f"Temperature: {temperature:.2f}°C")
         print(f"Humidity: {humidity:.2f}%")

         # Send data to ThingSpeak
         if THINGSPEAK_API_KEY and THINGSPEAK_CHANNEL_ID:
            data = {'api_key': THINGSPEAK_API_KEY, 'field1': temperature, 'field2':
            humidity} response = requests.post(f'https://api.thingspeak.com/update',
            data) print(f"ThingSpeak Response: {response.status_code}")
         else:
            print("ThingSpeak API key or channel ID not provided. Data not
sent to ThingSpeak.")

      else:
         print("Failed to retrieve data from the sensor.")

      time.sleep(300) # Wait for 5 minutes before taking the next reading (adjust as

   needed) except KeyboardInterrupt:
      break
```

PYTHON SCRIPT FOR GPS(GLOBAL POSITIONING SYSTEM):

```
python
import time
import
serial
import
pynmea2
import
requests

# Configure your ThingSpeak channel details
THINGSPEAK_API_KEY =
```

'YOUR_THINGSPEAK_API_KEY'

```
THINGSPEAK_CHANNEL_ID = 'YOUR_THINGSPEAK_CHANNEL_ID'

# Serial port to which the GPS module is connected
GPS_SERIAL_PORT = '/dev/ttyS0' # Change to your GPS module's serial port

# Initialize the serial connection to the GPS module
ser = serial.Serial(GPS_SERIAL_PORT, baudrate=9600, timeout=1)

while
   True:
   try:
      # Read and parse NMEA sentences from the GPS
      module sentence = ser.readline().decode()
      if
         sentence.startswith('$GPGGA
         '): data =
         pynmea2.parse(sentence)

         # Extract latitude and
         longitude latitude =
         data.latitude longitude =
         data.longitude

         # Print GPS data to the console
         print(f"Latitude: {latitude:.6f}, Longitude: {longitude:.6f}")

         # Send data to ThingSpeak
         if THINGSPEAK_API_KEY and THINGSPEAK_CHANNEL_ID:
            data = {'api_key': THINGSPEAK_API_KEY, 'field1': latitude, 'field2':
            longitude} response =
            requests.post(f'https://api.thingspeak.com/update', data)
            print(f"ThingSpeak Response: {response.status_code}")
         else:
            print("ThingSpeak API key or channel ID not provided. Data not
sent to ThingSpeak.")

      time.sleep(5) # Read GPS data every 5 seconds (adjust as

   needed) except KeyboardInterrupt:
      break
```