

**KGiSL INSTITUTE OF TECHNOLOGY**

(Approved By AICTE, New Delhi, Affiliate to Anna University

Recognized by UGC, Accredited by NBA(IT)

265, KGISL Campus, Thudiyalur Road, Saravanampatti, Coimbatore-641035**.)**

**NAAN MUDHALVAN - INTERNET OF THINGS**

# NOISE POLLUTION MONITORING



**TEAM MEMBERS**

**Janatte Ruth J (711721106047)**

**Madhumitha p (711721106059)**

**Bhavana Grace (711721106018)**

**Deepti S D (711721106023)**

**Manisha K (711721106061)**

**Phase 3: Development Part 1**

**Building a Noise pollution Monitoring system using IoT sensors and Raspberry Pi integration**

**Hardware and Software Components Needed:**

**Hardware :**

1. Raspberry Pi (with internet connectivity)
2. Sound Sensor (such as the KY-038 Microphone sound sensor)
3. Jump wires
4. Noise Sensor:Use a noise sensor (such as a sound level sensor) to measure the noise levels.
5. Microcontroller: ESP8266 or ESP32 boards can be used for this purpose, which can communicate with the internet using Wi-Fi.
6. Cloud Service: Use a cloud service like Thing Speak or Firebase to store and analyse the data.
7. Display Unit: You can use an LCD screen or LEDs to display real-time noise levels locally.

**Software :**

* Python installed on Raspberry Pi
* Requests library for making HTTP requests (install it using pip install requests)

1. **Microcontroller Code (Python on MicroPython):** 
   * Connect the noise sensor to the microcontroller.
   * Read data from the sensor.
   * Send data to the cloud service at regular intervals.
2. **Data Analysis and Visualization:** 
   * Retrieve data from the database.
   * Analyse noise patterns over time.
   * Visualize the data using libraries like Matplotlib or Plotly.
3. **Cloud Service (Python using Flask for API):** 
   * Create a REST API using Flask to receive data from the microcontroller.
   * Store the received data in a database.
   * Implement endpoints for data retrieval and analysis.

**Procedure:**

## Step 1: Define Project Requirements and Components

* Define Project Scope: Determine the area you want to monitor for noise pollution and establish specific goals for the monitoring system.
* Select Components: Choose suitable noise sensors (such as sound level sensors or microphones), Raspberry Pi board, and necessary accessories (wires, resistors, breadboard, etc.).
* Choose IoT Platform: Select an IoT platform (e.g., ThingSpeak, AWS IoT, Google Cloud IoT) for storing and analyzing sensor data.

## Step 2: Set Up Raspberry Pi and Connect Sensors

* Set Up Raspberry Pi: Install the latest Raspbian OS on your Raspberry Pi. Configure Wi-Fi and update packages using terminal commands: sudo apt-get update and sudo apt-get upgrade.
* Connect Noise Sensor: Connect the noise sensor to the Raspberry Pi's GPIO pins. Refer to the sensor's datasheet for wiring instructions. **Step 3: Write Python Code for Data Collection**
* Install Necessary Libraries: Install Python libraries for GPIO control and sensor communication (e.g., RPi.GPIO, Adafruit\_GPIO, Adafruit\_ADS1x15).
* Write Python Code: Write Python code to read data from the noise sensor. Use appropriate libraries and communication protocols (e.g., I2C, SPI) based on your sensor's specifications.
* Test Sensor Reading: Verify that your Raspberry Pi can read data from the noise sensor accurately. Print sensor values to the console for testing.

## Step 4: Set Up IoT Platform

* Create Account: Sign up for an account on the chosen IoT platform (e.g., ThingSpeak).
* Create Channels: Create channels to store noise level data. Define fields for storing sensor values and metadata.
* Generate API Key: Obtain an API key from the IoT platform to authenticate your Raspberry Pi for data transmission**.**

## Step 5: Write Python Code for Data Transmission

* Install Requests Library: Install the requests library for making HTTP requests:

pip install requests.

* Write Transmission Code: Write Python code to send sensor data to the IoT platform using HTTP requests. Include the API key and appropriate endpoint URLs in your code.
* Implement Error Handling: Implement error handling in your code to handle network issues and server errors gracefully**.**

## Step 6: Schedule Data Transmission (Optional)

* Use Cron Jobs: Set up a cron job on your Raspberry Pi to run the Python script at regular intervals (e.g., every 5 minutes).
* Ensure Stability: Test the scheduled data transmission to ensure the system works reliably over extended periods. **Step 7: Data Visualization and Analysis**
* Retrieve Data: Implement code to retrieve data from the IoT platform's API for visualization and analysis.
* Visualize Data: Use libraries like Matplotlib or Plotly to create graphs and charts visualizing noise levels over time.
* Implement Analysis: Implement basic analysis algorithms to identify noise patterns, peaks, and trends.

## Step 8: Documentation and Reporting

* Document Your Project: Create detailed documentation covering hardware connections, software components, code explanations, and system architecture.
* Write Project Report: Prepare a comprehensive project report detailing the objectives, methodology, implementation, challenges faced, and results obtained. Include visualizations and analysis findings in your report.

## Step 9: Testing and Calibration

* Test System: Conduct thorough testing of the entire system to ensure sensor accuracy, data transmission reliability, and notification functionality (if implemented).
* Calibration: If required, calibrate the sensors to ensure accurate measurement based on real-world noise levels.

## Step 10: Deployment

* Deploy the System: Install the noise pollution monitoring system in the target location.
* Monitor and Maintain: Regularly monitor the system's performance and address any issues promptly. Maintain the system to ensure continuous and accurate operation**.**

**SOURCE CODE:** import time import requests from gpiozero import InputDevice

from flask import Flask, request, jsonify

# Define the GPIO pin connected to the noise sensor noise\_sensor = InputDevice(17) # Replace 17 with the actual GPIO pin

# Define the endpoint of your noise pollution information platform

platform\_url = ["https://noisepollution.com/api/noise-data"](https://noisepollution.com/api/noise-data)

app = Flask(\_\_name)

@app.route('/update', methods=['POST']) def update\_noise\_level(): try:

noise\_level = request.json.get('noise\_level') if noise\_level is not None:

# Simulate sending data to the platform print(f"Received noise level: {noise\_level}") return jsonify({"message": "Data received successfully"}) else:

return jsonify({"error": "Invalid data format"}), 400 except Exception as e: return jsonify({"error": str(e)}), 500

def send\_noise\_data(): while True: try:

noise\_level = noise\_sensor.value

# Create a JSON payload with the noise data data = {"noise\_level": noise\_level}

# Send the data to the platform

response = [requests.post(](http://requests.post/)platform\_url, json=data)

if response.status\_code == 200: print("Data sent successfully") else:

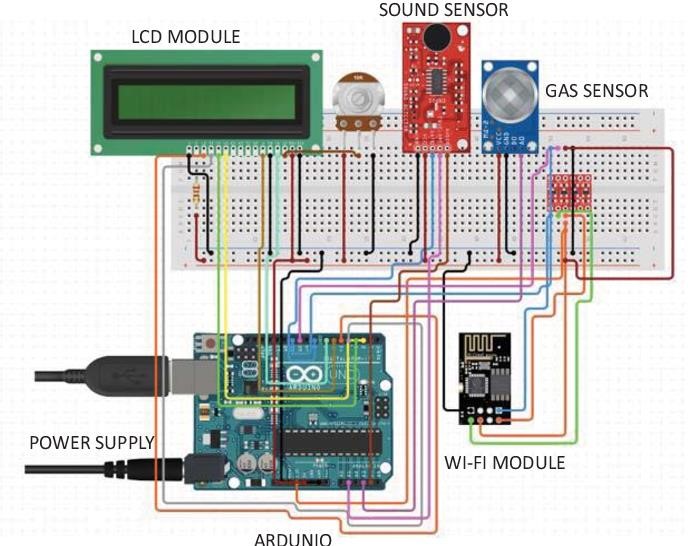
print(f"Failed to send data. Status code: {response.status\_code}")

# Adjust the frequency of data transmission as needed time.sleep(60) # Send data every 60 seconds except Exception as e: print(f"An error occurred: {str(e)}")

if \_\_name\_\_ == '\_\_main\_\_':

send\_noise\_data()

app.run(host='0.0.0.0', port=8080)



**CONCLUSION:**

In conclusion, building a Noise Pollution Monitoring using IoT sensors and

Raspberry Pi integration is a valuable. Project that offers solutions to Noise Pollution challenges. By following the step-by-step procedure outlined above, we can create a reliable and efficient Noise Monitoring system.