

IOT – based Noise pollution monitoring system

Phase3:Development

Part1



Abstract

●Noise pollution has become a growing concern in urban areas, leading to various health and environmental issues.

●To address this problem, an Internet of Things (IoT)-based noise pollution monitoring system is proposed. This system comprises several interconnected modules that collectively provide an efficient and real-time solution for monitoring and managing noise pollution.

●Noise pollution is a growing problem in many parts of the world, and it can have a significant impact on human health and well-being.

● IoT-based noise pollution monitoring systems can be used to track noise levels in real time and provide alerts when noise levels exceed safe limits. These systems can also be used to collect data on noise levels over time, which can be used to identify trends and patterns.

Modules

●Sensors: Noise sensors are used to measure the sound level in the environment. There are a variety of noise sensors available, each with its own advantages and disadvantages. Some common types of noise sensors include microphones, sound level meters, and acoustic cameras.

●Microcontroller: The microcontroller is the brain of the system. It is responsible for collecting data from the sensors, processing the data, and communicating with the cloud. Some popular microcontrollers for IoT projects include the Arduino and Raspberry Pi.

●Cloud platform: The cloud platform is used to store and analyze the data collected by the system. The cloud platform can also be used to generate alerts when noise levels exceed safe limits. Some popular cloud platforms for IoT projects include AWS IoT Core and Azure IoT Hub.

●Data Acquisition :Data collected from the sensors is acquired, processed, and analyzed. The system's central processing

unit is responsible for this task. The data includes sound intensity, frequency, and location information.

●**Data Analysis** : On the cloud platform, the data is further analyzed to identify noise pollution patterns, trends, and potential sources. Machine learning algorithms can be employed to detect unusual noise events.

●**Visualization and Alerts** : The system provides a user-friendly interface for visualizing the noise data in real-time. It also sends alerts to relevant authorities or stakeholders when noise levels exceed predefined thresholds.

●**Data Reporting** : Regular reports and insights are generated from the stored data, facilitating decision-making and policy formulation.

●**Historical Data Storage** : The system archives historical noise data, enabling the tracking of long-term trends and the assessment of the effectiveness of noise pollution mitigation measures.

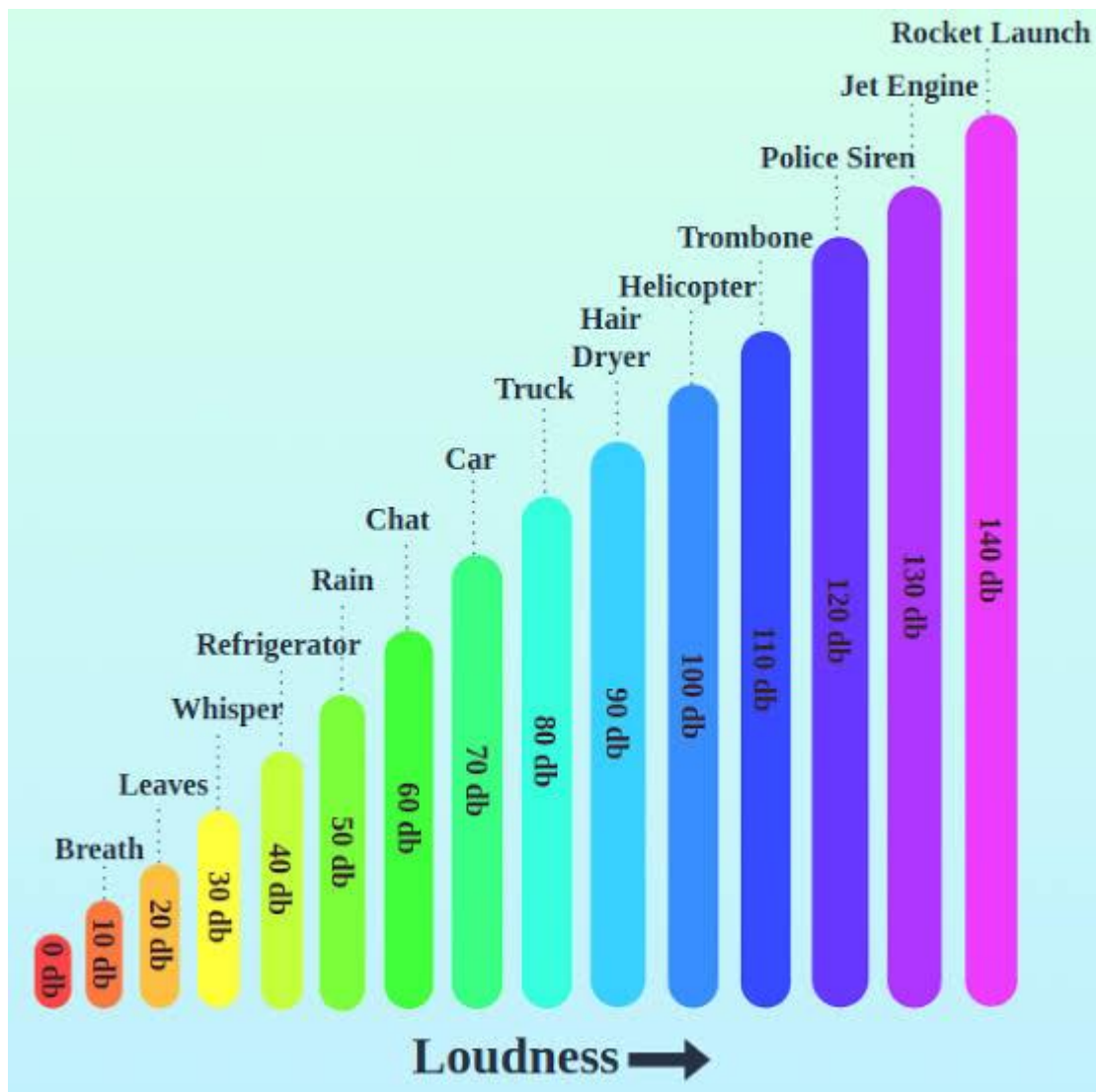


Fig Noise Pollution

Ideas for how to use the IoT-enabled Noise Pollution Monitoring system in real-time basis:

- Develop a mobile app or website that allows people to view the real-time noise levels in their area.

- Use the data to develop noise pollution maps that show the areas with the highest noise levels.

- Partner with local governments and businesses to develop noise mitigation strategies.



Steps and outline

#Step1. Import the necessary libraries.

#Step2. Connect to the noise sensors.

#Step3. Read the noise level data from the sensors

#Step4. Format the noise level data in a way that is compatible with the

noise pollution information platform.

#Step5. Send the noise level data to the noise pollution information

platform.

Sample code for the above process(python).:

Sample sensor data from various IoT devices

sensor_data = [

{‘ type ’: ‘ temperature ’, ‘ value ’: 20},

{‘ type ’: ‘ temperature ’, ‘ value ’: 22},

{‘ type ’: ‘ humidity ’, ‘ value ’: 55},

{‘ type ’: ‘ temperature ’, ‘ value ’: 24},

{‘ type ’: ‘ humidity ’, ‘ value ’: 60},

{‘ type ’: ‘ temperature ’, ‘ value ’: 25},

```
{'type': 'humidity', 'value': 50},  
]
```

```
# Group sensor data by type
```

```
grouped_data = { }
```

```
for data in sensor_data:
```

```
    sensor_type = data['type']
```

```
    if sensor_type not in grouped_data:
```

```
        grouped_data[sensor_type] = [ ]
```

```
    grouped_data[sensor_type].append(data['value'])
```

```
# Calculate average values for each sensor  
type and store them in a list
```

```
average_values = [ ]
```

```
for sensor_type, values in grouped_data.items( ):
```

```
    average_value = sum(values) / len(values)
```

```
    average_values.append({'type': sensor_type,  
                           'average_value': average_value})
```

```
# Sort the list of average values
```

```
average_values.sort(key=lambda x: x['average_value'])
```

```
# Print the sorted list of average
```

```
Sensor values print(average_values)
```

```
[{'type': 'temperature',  
  'average_value': 22.75},  
 {'type': 'humidity',  
  'average_value': 55.0}]
```

Python code: from django.db
import models

```
# Create your models here.
```

```
#class DataLake(models.Model):
```

```
class trafficdata(models.Model):
```

```
    """
```

Data Lake For Storing Data of all the data

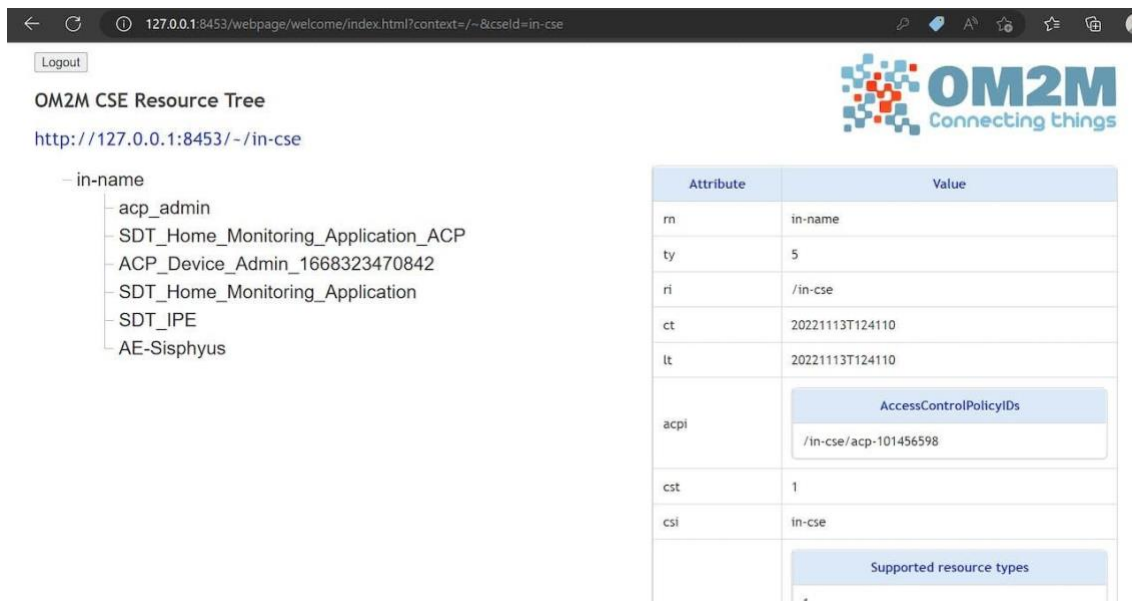
OneM2M implementation

Pushing Data to OM2M Eclipse server

We can see the content instances under the "Data" container updating the OM2M server. The attributes on the right also show the data sent from the Node

MCU microcontroller Arduino IDE code. We are sending the data that can be seen in the resource-specific attributes.

- **node_id:** which is the unique identifier of the data content instance
- **dB value** of the noise level detected
- **Signal status** as in GREEN, RED, AMBER
- **Time out** of the counter of the RED signal
- **State** indicating if the timer has been extended.



The screenshot shows the OM2M CSE Resource Tree web interface. On the left, there is a tree view under the path `/in-cse` with the following nodes: `acp_admin`, `SDT_Home_Monitoring_Application_ACP`, `ACP_Device_Admin_1668323470842`, `SDT_Home_Monitoring_Application`, `SDT_IPE`, and `AE-Sisphyus`. On the right, there is a table with two columns: **Attribute** and **Value**.

Attribute	Value
rn	in-name
ty	5
ri	/in-cse
ct	20221113T124110
lt	20221113T124110
acpi	<div>AccessControlPolicyIDs</div> <div>/in-cse/acp-101456598</div>
cst	1
csi	in-cse
	<div>Supported resource types</div> <div>1</div>

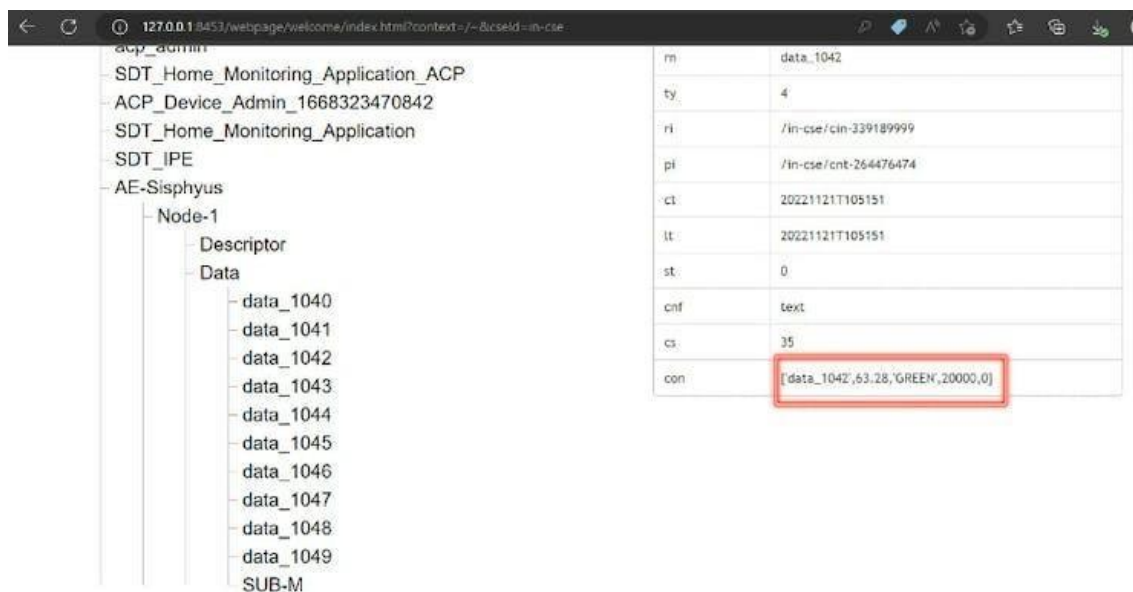
Activate the OM2M server. Opened the browser to access the OM2M

IoT platform web interface:

<http://127.0.0.1:8453/webpage>

OM2M Resource Tree

Using Postman, the resource tree is created by POST requests, and the Application Entity - AE Sisphyus is shown below.



PostgreSQL

The data from the server is pushed to OM2M and using the subscription functionality, the Django server is now subscribed to the OM2M server.

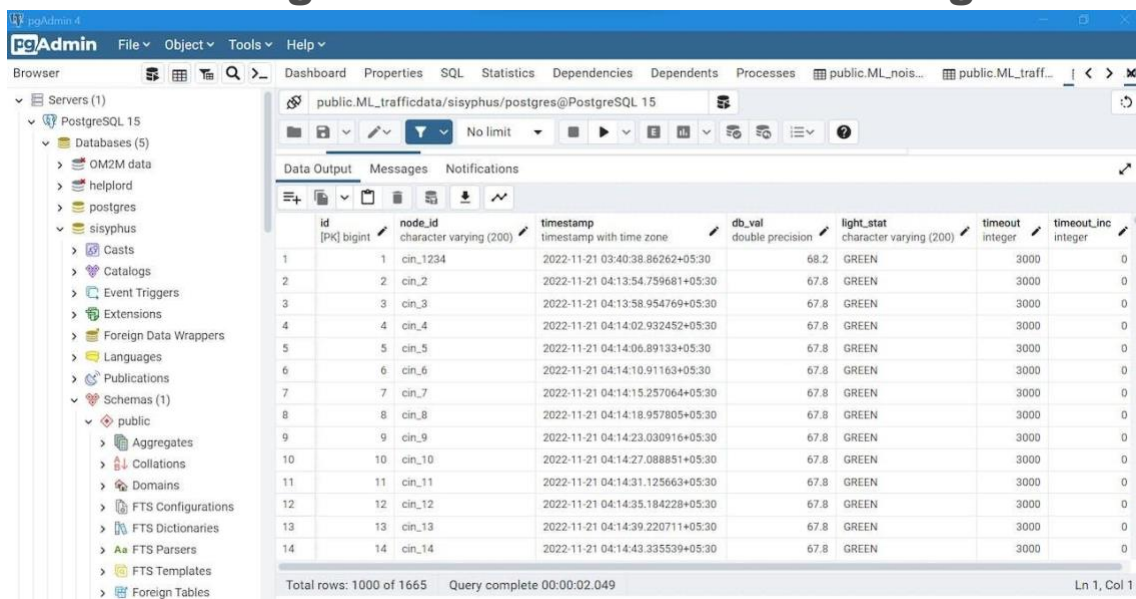
Django is an open-source Python framework used for highly functioning web applications, it uses the options method to send data when an HTTP request is sent and gives it in the form of a

JSON format. In this case, the server is used to create a database in PostgreSQL.

```
{'m2m:sgn': {'m2m:nev': {'m2m:rep': {'m2m:cin': {'rn': 'data_386', 'ty': 4, 'ri': '/in-cse/cin-437542018', 'pi': '/in-cse/cnt-35599612', 'ct': '20221123T170551', 'lt': '20221123T170551', 'st': 0, 'cnf': 'text', 'cs': 32, 'con': '['data_386',57.50,'RED',20000,0]'}}, 'm2m:rss': 1}, 'm2m:sud': False, 'm2m:sur': '/in-cse/sub-282549354'}}
[23/Nov/2022 17:05:51] "POST / HTTP/1.1" 200 13
['data_385', 57.34, 'RED', 20000, 0]
saved to database
{'m2m:sgn': {'m2m:nev': {'m2m:rep': {'m2m:cin': {'rn': 'data_387', 'ty': 4, 'ri': '/in-cse/cin-116590411', 'pi': '/in-cse/cnt-35599612', 'ct': '20221123T170555', 'lt': '20221123T170555', 'st': 0, 'cnf': 'text', 'cs': 32, 'con': '['data_387',57.03,'RED',20000,0]'}}, 'm2m:rss': 1}, 'm2m:sud': False, 'm2m:sur': '/in-cse/sub-282549354'}}
[23/Nov/2022 17:05:56] "POST / HTTP/1.1" 200 13
['data_386', 57.5, 'RED', 20000, 0]
saved to database
```

Django server output

The following database is created in PostgreSQL.



The screenshot shows the pgAdmin 4 interface. On the left, the 'Servers' tree is expanded to show a PostgreSQL 15 server with a 'public' schema. The main pane displays a table with 14 rows. The table has columns: id (PK, bigint), node_id (character varying (200)), timestamp (timestamp with time zone), db_val (double precision), light_stat (character varying (200)), timeout (integer), and timeout_inc (integer). The data shows a sequence of records with timestamps from 2022-11-21 03:40 to 2022-11-21 04:43. The status 'light_stat' is consistently 'GREEN'.

id	node_id	timestamp	db_val	light_stat	timeout	timeout_inc
1	cin_1234	2022-11-21 03:40:38.86262+05:30	68.2	GREEN	3000	0
2	cin_2	2022-11-21 04:13:54.759681+05:30	67.8	GREEN	3000	0
3	cin_3	2022-11-21 04:13:58.954769+05:30	67.8	GREEN	3000	0
4	cin_4	2022-11-21 04:14:02.932452+05:30	67.8	GREEN	3000	0
5	cin_5	2022-11-21 04:14:06.89133+05:30	67.8	GREEN	3000	0
6	cin_6	2022-11-21 04:14:10.91163+05:30	67.8	GREEN	3000	0
7	cin_7	2022-11-21 04:14:15.257064+05:30	67.8	GREEN	3000	0
8	cin_8	2022-11-21 04:14:18.957805+05:30	67.8	GREEN	3000	0
9	cin_9	2022-11-21 04:14:23.030916+05:30	67.8	GREEN	3000	0
10	cin_10	2022-11-21 04:14:27.088851+05:30	67.8	GREEN	3000	0
11	cin_11	2022-11-21 04:14:31.125663+05:30	67.8	GREEN	3000	0
12	cin_12	2022-11-21 04:14:35.184228+05:30	67.8	GREEN	3000	0
13	cin_13	2022-11-21 04:14:39.220711+05:30	67.8	GREEN	3000	0
14	cin_14	2022-11-21 04:14:43.335539+05:30	67.8	GREEN	3000	0

Total rows: 1000 of 1665 Query complete 00:00:02.049 Ln 1, Col 1

Conclusion

This is a basic overview of how to build an IoT-enabled Noise Pollution Monitoring system. This system can be used to monitor noise levels in real time and collect data on noise levels over time. The data collected by the system can be used to identify and address noise pollution problems.