

NOISE POLLUTION MONITORING

Problem Statement

Noise pollution to a huge extent is neglected despite its short and long-term ramifications on human health. The various permissible noise limits in residential, commercial, and industrial areas should be strictly followed in accordance with the governing law and WHO guidelines to reduce the effects of noise pollution on human health. Honking by commuters near traffic junctions is one of the main sources of noise and the effect of increased automobile use in recent years especially in various metropolitan cities has resulted in increased noise levels in urban areas.

Solution

The solution focuses on traffic junctions throughout the city and plans to deploy a noise monitoring node at every junction. As a part of the solution to this project, we have implemented a "**Punishing Signal**" to create awareness about noise pollution among commuters. The implementation includes a noise pollution monitoring node that reads the noise levels at a given traffic junction and also the prototype of a traffic light controller which controls the flow of traffic at a junction. The idea is to discourage commuters from honking unnecessarily at a traffic light.

Proposed System - How it works

I aim to propose a low-cost, reliable, compact, and real-time **OM2M** and **IoT** noise pollution monitoring system at high-density traffic junctions and develop a mechanism to **discourage people from honking**.

I use the **ESP8266** microcontroller interfaced with the **DF Robot Analog Sound Level Meter** sensor whose output is in Decibels with 'A' weighting (dB(A)) with the same weighting closest to how humans perceive sound. The Traffic Signal Controller code to implement the Punishing Signal runs on Arduino IDE.

The traffic signal prototype is programmed to stay RED for 20 seconds, YELLOW for 3 seconds, and GREEN for 30 seconds. The countdown timer for each state is shown on an OLED screen. When the noise level above a specified threshold is sensed more than 3 times when the traffic signal is in RED state then 10 seconds is added to the red signal countdown timer.

The noise data, signal status, and countdown timer are all sent to the **OM2M Eclipse server**, and using OM2M functionality the data is subscribed to a **Django** server. The data from the Django server is created into a database in **PostgreSQL**. The database is then used to create a **Grafana** dashboard for analytics.

Prerequisites

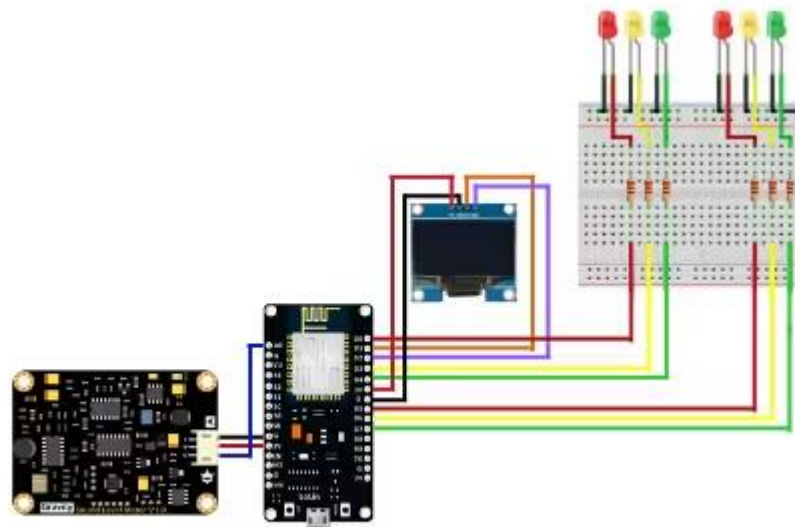
- Arduino IDE 1.8.5 or later(To compile and upload code on sensor nodes)
- JAVA 1.8 (To run the OneM2m platform)
- Eclipse OM2M v1.4.1 (implementation of the standard used in the project)
- NodeJS v14
- Postman
- Grafana v9.2 or later

Implementation of the Punishing Signal

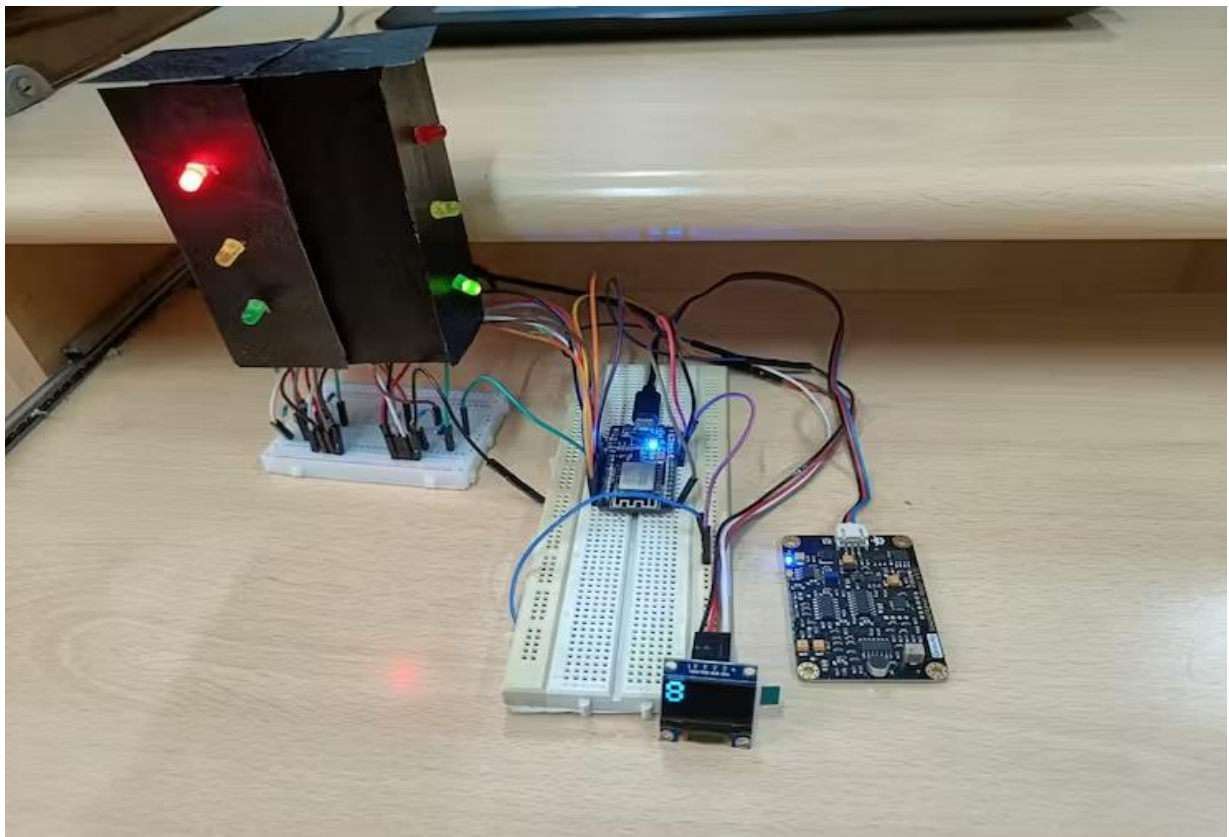
Schematics

Connect the DFRobot Analog Sound sensor, OLED, and the respective LEDs to the NodeMCU board using the following components:

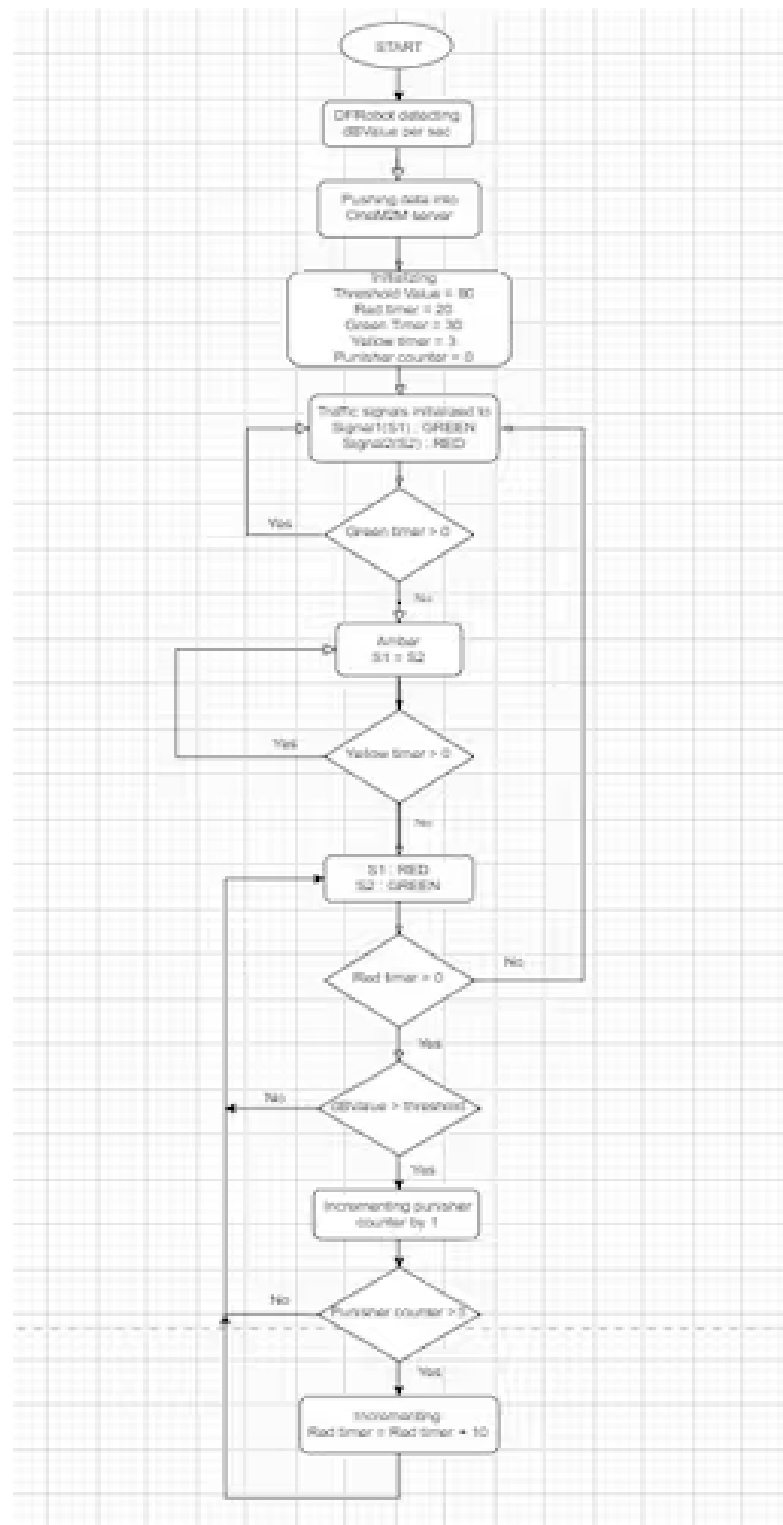
- Breadboard
- NodeMCU board
- DFRobot Analog Sound Level Meter
- OLED
- Red, Green, and Yellow LEDs.
- 220 ohmic resistors per LED.



Prototype



Flowchart



Working

Imposing the RED signal is done when the noise levels exceed the threshold more than 3 times. This is done by initiating a counter and is based on the frequency of noise levels in dB(A) above the threshold. If it exceeds the threshold count again during the imposition of the RED signal, an additional 10 seconds are added to the wait time.

The Traffic Signal is demonstrated by using 3 LEDs which glow corresponding to their interval. Considering an intersection with the main road and a side road crossing the main road the time interval taken for the signal is 20 seconds and the GREEN signal is 30 seconds for the main road and 30 seconds for RED and 20 seconds for GREEN for the side road. Yellow LED will glow on each transition between GREEN to RED for 3 seconds. The countdown timer for each state is shown on an OLED screen. When the noise level above a specified threshold is sensed more than 3 times when the traffic signal is in RED state then 10 seconds is added to the red signal countdown timer as a punishment for excessive and unnecessary honking. If the commuters still honk even after 10 second punishment has been added and the noise levels cross the threshold more than 3 times again another 10 seconds is added to count down the timer as a punishment the second time. We have limited the number of punishments to 3 times. After the maximum number of punishments, the traffic light will go back to green and it will work as the regular traffic light controller.

Actuators - Display Message

A message is generated to be flashed and displayed at the junction during the imposed AMBER signal time. The counter of the Signal states is also displayed clearly to the passengers. We are using an OLED connected to ESP8266 during the same interval. This puts into perspective the overall noise pollution and awareness to the general public of all the signals at the respective junction.

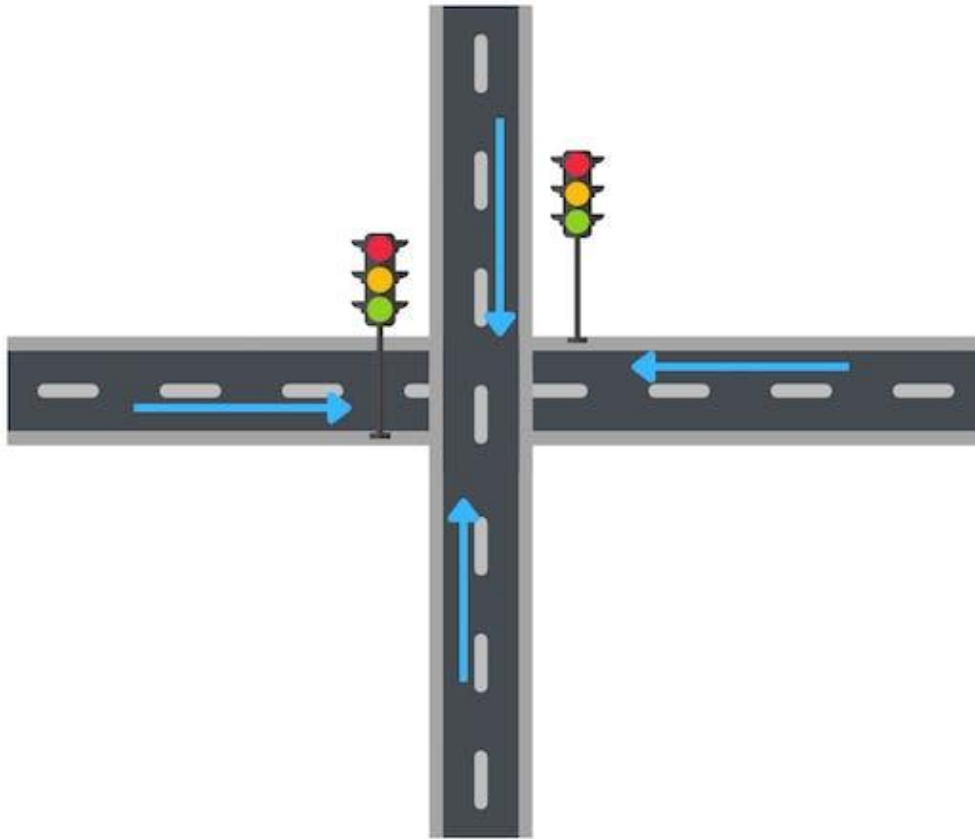


Scenarios and Scope of the Prototype

The threshold value can be changed according to the time of day and the recommended dB values following WHO guidelines. The Traffic Signal prototype can also be extended to 3- way T or Y junctions and even 4-way junctions respectively.

OneM2M can also be used to push data from various sensors data from across the city and store it in one place.

I intend to have different Data containers in the OM2M resource tree monitor various deployable noise pollution monitoring systems across the city.



Scope of the prototype

Ambient air quality standards in respect of noise :

Area Code	Category of Area / Zone	Limits in dB(A) Leq*	
		Day Time	Night Time
(A)	Industrial area	75	70
(B)	Commercial area	65	55
(C)	Residential area	55	45
(D)	Silence zone	50	40

Central Pollution Control Board, India.

OneM2M implementation

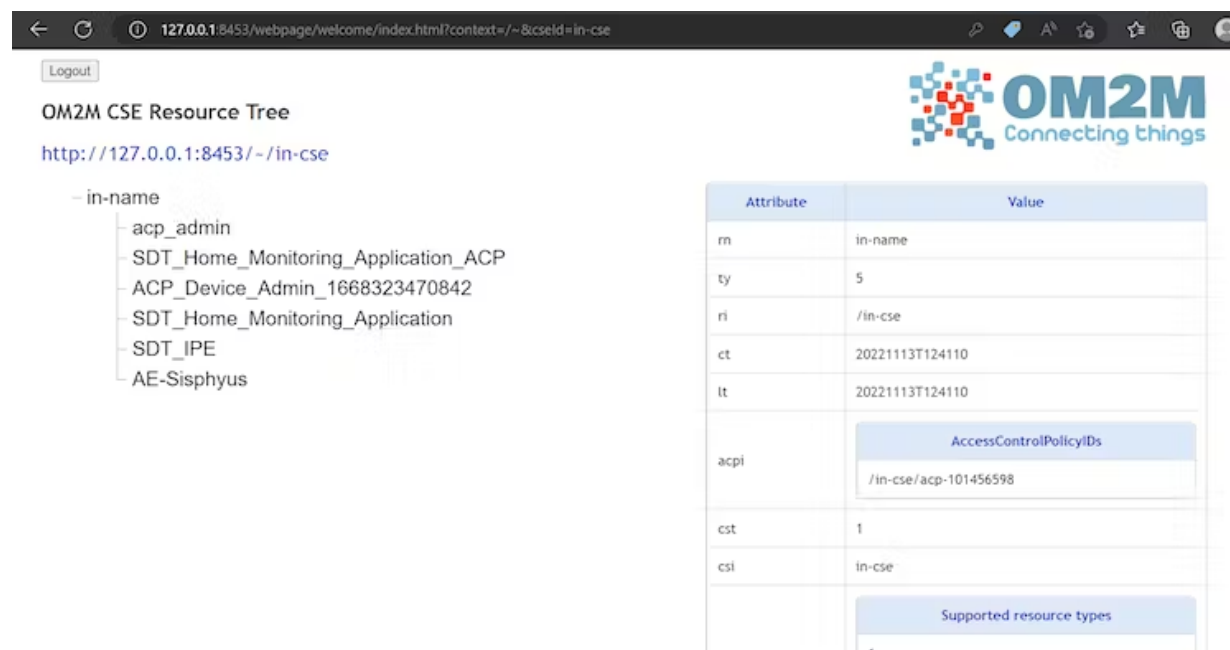
Configure the IoT platform

The IoT platform for the Eclipse server will listen on port 8080. In this case, we have changed the port to 8453 by changing the configuration by editing the file: *"config"* file.

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.



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>

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.

acp_admin	rn	data_1042
-SDT_Home_Monitoring_Application_ACP	ty	4
-ACP_Device_Admin_1668323470842	ri	/in-cse/cin-339189999
-SDT_Home_Monitoring_Application	pi	/in-cse/cnt-264476474
-SDT_IPE	ct	20221121T105151
-AE-Sisphyus	lt	20221121T105151
-Node-1	st	0
-Descriptor	cnf	text
-Data	cs	35
-data_1040	con	[data_1042,63.28,'GREEN',20000,0]
-data_1041		
-data_1042		
-data_1043		
-data_1044		
-data_1045		
-data_1046		
-data_1047		
-data_1048		
-data_1049		
-SUB-M		

OM2M server and content instances

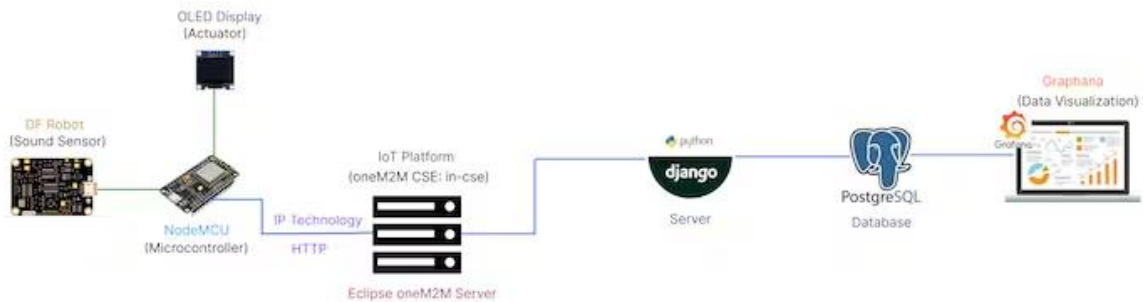
Subscription from OM2M server

The subscription resource connects to the Django server and keeps track of the status of active subscriptions to the parent resource. A request from the issuer is notified about modifications to the parent resource.

acp_admin	rn	SUB-M
-SDT_Home_Monitoring_Application_ACP	ty	23
-ACP_Device_Admin_1668323470842	ri	/in-cse/sub-678871514
-SDT_Home_Monitoring_Application	pi	/in-cse/cnt-264476474
-SDT_IPE	ct	20221119T143933
-AE-Sisphyus	lt	20221119T143933
-Node-1	acpi	AccessControlPolicyIDs /in-cse/acp-101456598
-Descriptor	nu	• http://127.0.0.1:8000/
-Data	nct	2
-cin_9		
-cin_10		
-cin_11		
-cin_12		
-cin_13		
-cin_14		
-cin_15		
-cin_16		
-cin_17		
-cin_18		
-SUB-M		

OM2M subscription resource attributes

High-Level Software/Hardware Architecture



The noise levels are detected from the sound sensor - DF Robot and is read by the Node MCU microcontroller, which implements the traffic light controlling system and uses the actuator OLED to display the warning message. The output from the microcontroller is then pushed to the OM2M server which includes - dB values, the current signal status, timeout of the RED signal and an indication of imposition of the red signal. Using the OM2M subscription functionality we have subscribed the Django server to the data in the "Data" container in the resource tree. The Django server receives the data through the OM2M server and send the data to postgresSQL, where the database is created. This database is then used as a data source for creating a Grafana dashboard to show the visualization and analytics of data.

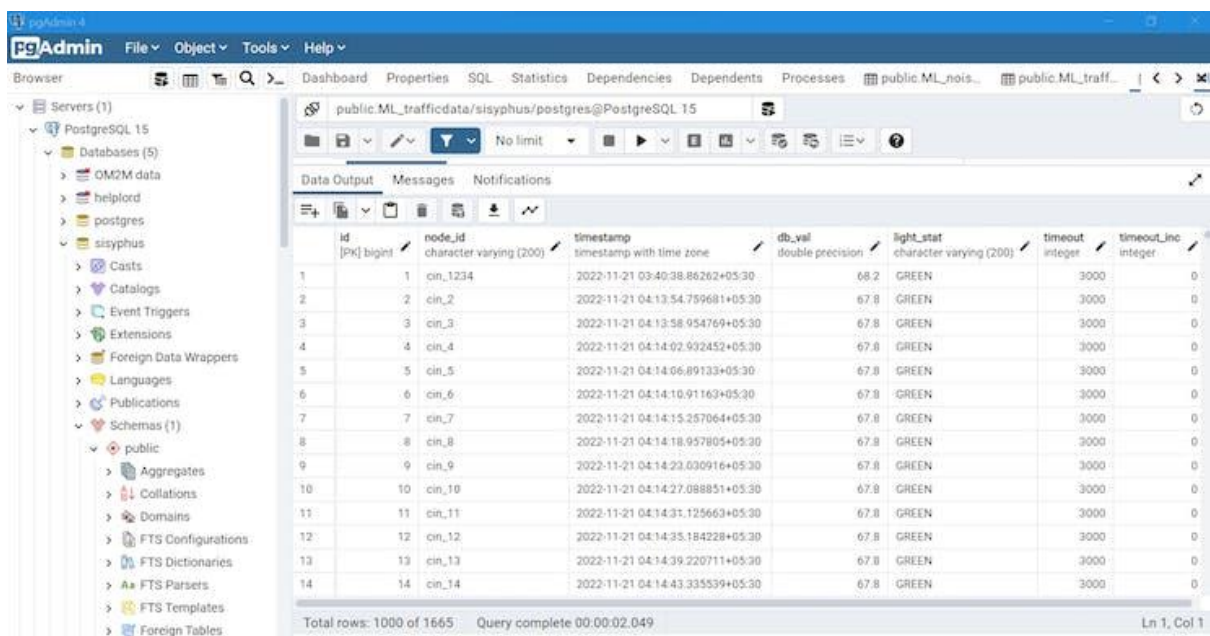
Collecting Data from OM2M Server and Posting To PostgreSQL

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 the 'public' schema. The main pane displays a table with the following columns: id (PK), node_id, timestamp, db_val, light_stat, timeout, and timeout_inc. The table contains 14 rows of data, all with a 'light_stat' of 'GREEN' and a 'timeout' of 3000.

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

postgresql database

Dashboard - Data Analysis

I am going to using an analytics and visualization application called **Grafana** to display the information that has been stored in the database.

The Noise Pollution levels and Traffic Signal Controller Dashboard is set up on Grafana. The OM2M server content instances will then be posted to a time-series database in PostgreSQL.

I built our dashboard by creating panels. Each panel is essentially a graphic that uses a query to pull information from the database that is provided as the data source for the dashboard.



Grafana Dashboard

I set up a **Bar gauge** and a **histogram graph** for dB value metrics from the database table. I also set each trend to display the graph of all the dB values and its ranges along with the time taken for the traffic signal to turn into its respective states.