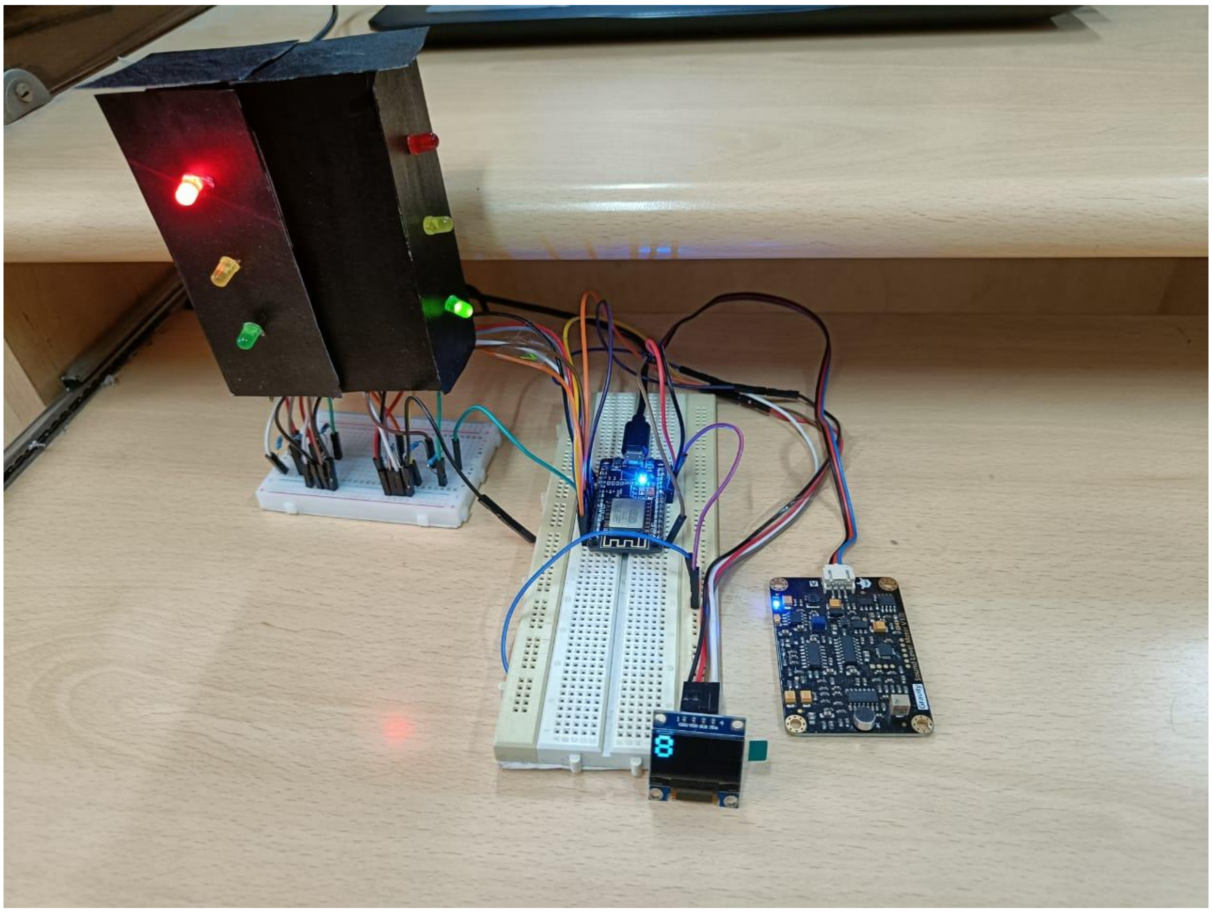
Phase 3: Innovation

To bring awareness on noise pollution levels in places with heavy traffic inflow and introduce measures to discourage people from honking.



Things used in this project

Hardware components

NodeMCU ESP8266 Breakout Board

DFRobot Gravity Analog Sound Level Meter

ElectroPeak 0.96" OLED 64x128 Display Module

5mm LED *10

USB-A to Micro-USB Cable

Resistor 220 ohm *6

Software apps and online services

Arduino IDE
Postman
Eclipse OM2M
Grafana
Postgresql
Django

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 that 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

We 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.**

We 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.

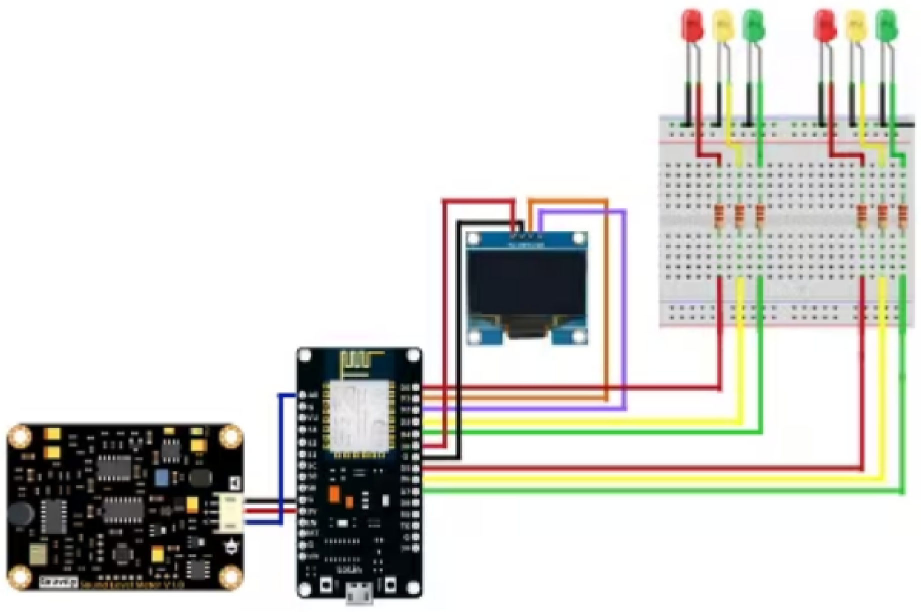
Implementation of the Punishing Signal

Schematics

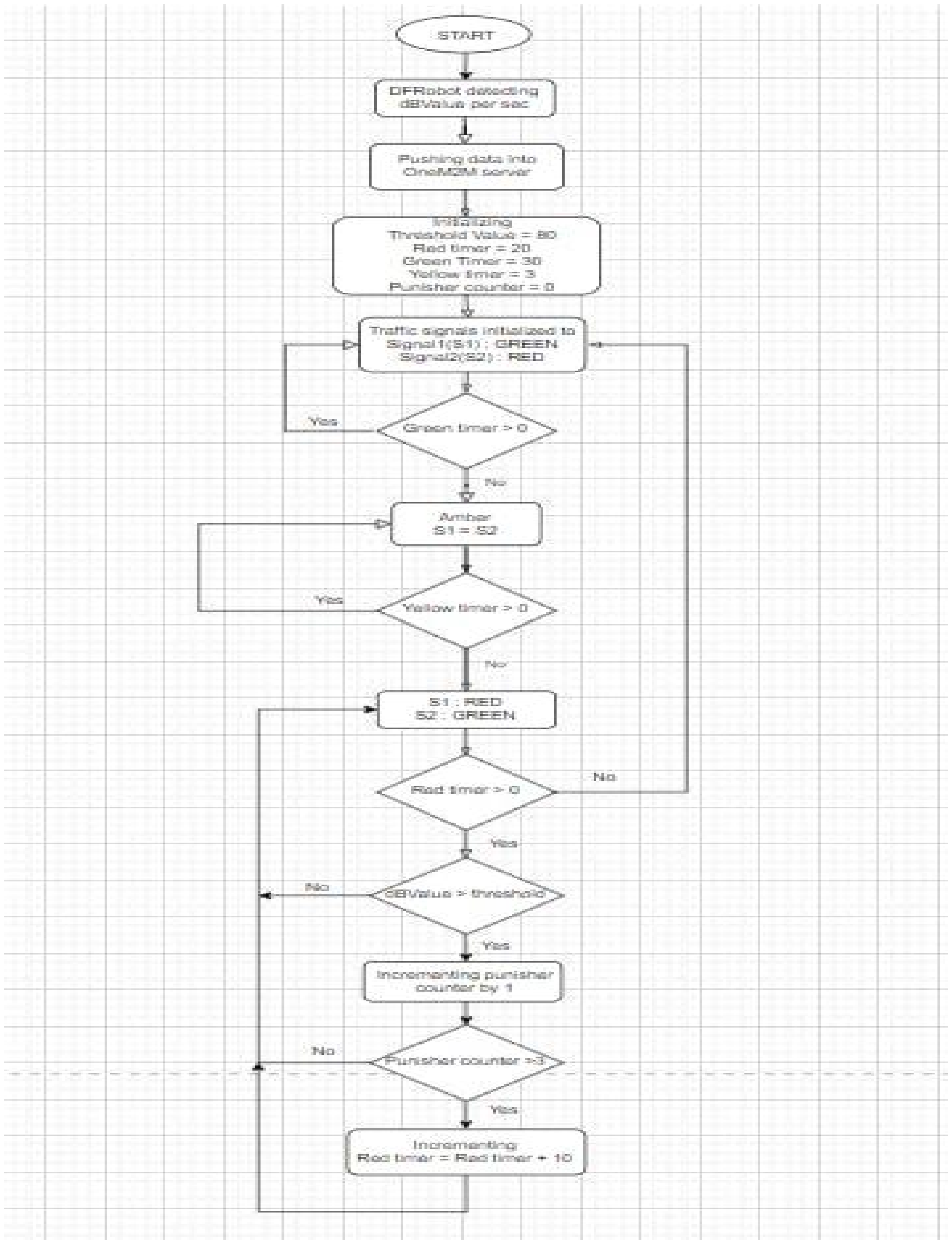
Connect the DFRobot Analog Sound sensor, OLED, and the respective LEDs to the NodeMCU board using the following components:

- Breadboard
- NodeMCU board

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



Flowchart


START
DFRobot detecting dBValue per sec
Pushing data into OneM2M server
Initializing
Threshold Value = 80
Red timer = 20
Green Timer = 30
Yellow timer = 3
Punisher counter = 0
Traffic signals initialized to
Signal1(S1) : GREEN
Signal2(S2) : RED
Yes
Green timer > 0
No
Amber
S1 = S2
Yes
Yellow timer > 0
No
S1 : RED
S2 : GREEN
Red timer > 0
No
Yes
No
dBValue > threshold
Yes
Incrementing punisher counter by 1
No
Punisher counter >3
Yes
Incrementing
Red timer = Red timer + 10

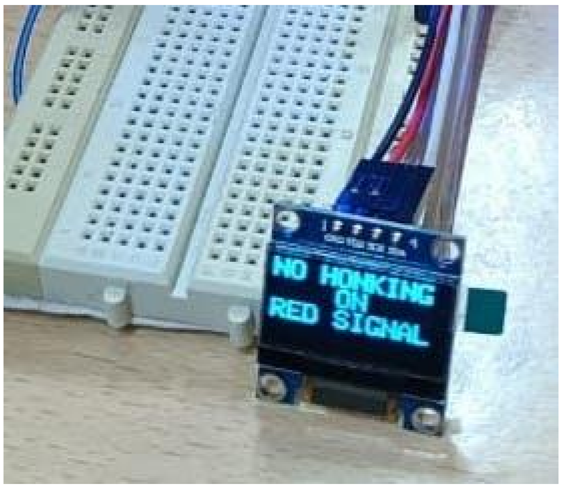
Working

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 30 seconds for RED and 20 seconds for GREEN for the side road. The 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 a 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.



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
    """
    node_id = models.CharField(max_length=200)
    timestamp =
models.DateTimeField(max_length=200,auto_no
w=True)
    db_val = models.FloatField(max_length=200)
    light_stat =
models.CharField(max_length=200,default='GRE
EN')
    timeout = models.IntegerField(default=3000)
    timeout_inc = models.IntegerField(default=0)



    def __str__(self):
        name = str(self.node_id) + "--" +
str(self.timestamp)
        name = name.upper()
        return name

    class Meta:
        unique_together = (("node_id",
"timestamp"),)
        ordering = ["-timestamp"]
```

OneM2M implementation

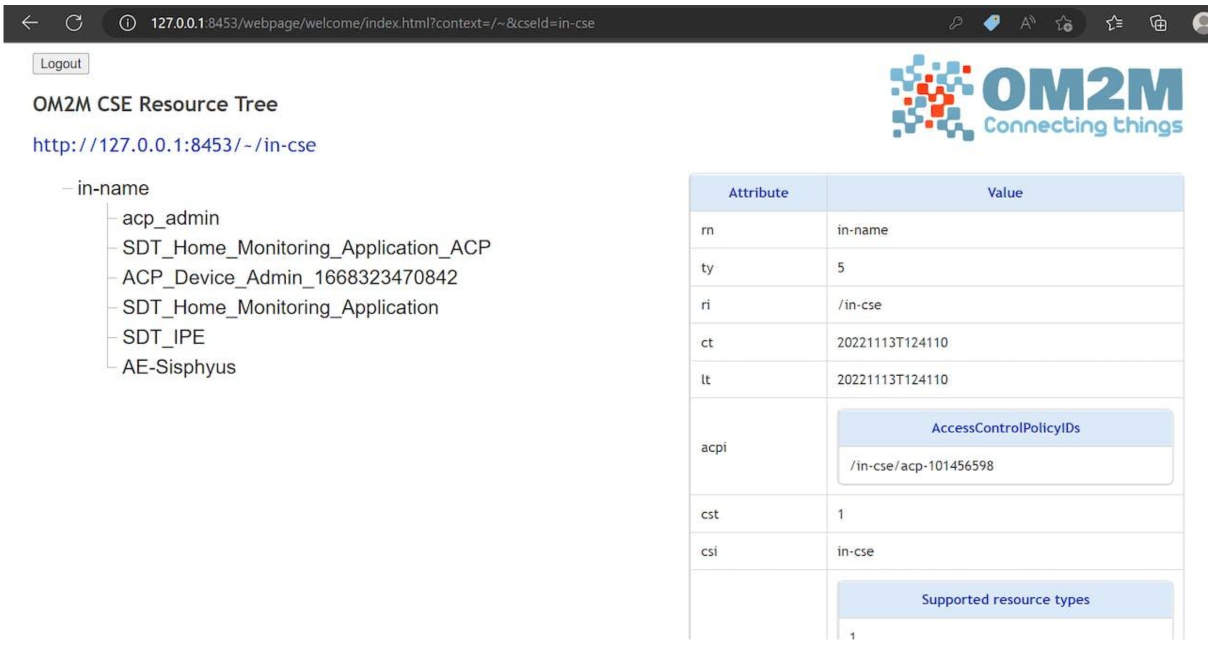
Configure the IoT platform

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.


127.0.0.1:8453/webpage/welcome/index.html?context=/~&cseId=in-cse
Logout
OM2M
Connecting things

OM2M CSE Resource Tree

http://127.0.0.1:8453/~/in-cse

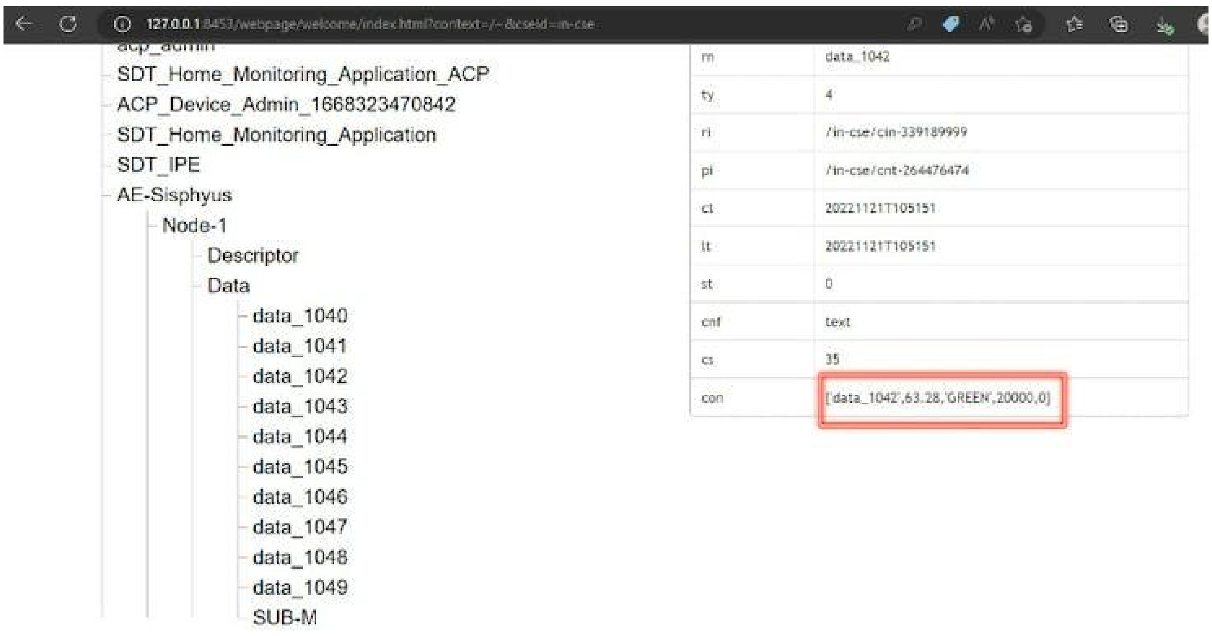
- in-name
  - acp_admin
  - SDT_Home_Monitoring_Application_ACP
  - ACP_Device_Admin_1668323470842
  - SDT_Home_Monitoring_Application
  - SDT_IPE
  - AE-Sisphyus

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

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.



127.0.0.1:8453/webpage/welcome/index.html?context=/~&cseId=in-cse

```
acp_admin
SDT_Home_Monitoring_Application_ACP
ACP_Device_Admin_1668323470842
SDT_Home_Monitoring_Application
SDT_IPE
AE-Sisphyus
    Node-1
        Descriptor
        Data
            data_1040
            data_1041
            data_1042
            data_1043
            data_1044
            data_1045
            data_1046
            data_1047
            data_1048
            data_1049
            SUB-M
```

rn	data_1042
ty	4
ri	/in-cse/cin-339189999
pi	/in-cse/cnt-264476474
ct	20221121T105151
lt	20221121T105151
st	0
cnf	text
cs	35
con	['data_1042',63.28,'GREEN',20000,0]

OM2M server and content instances

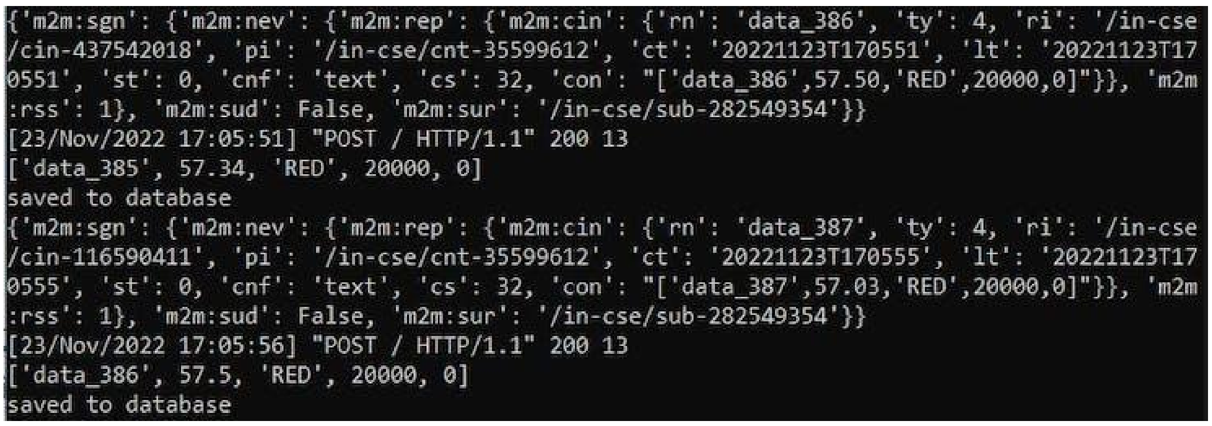
Subscription from OM2M server

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.

In this case, the Django server is http://127.0.0.1:8000/

Collecting Data from OM2M Server and Posting To PostgreSQL

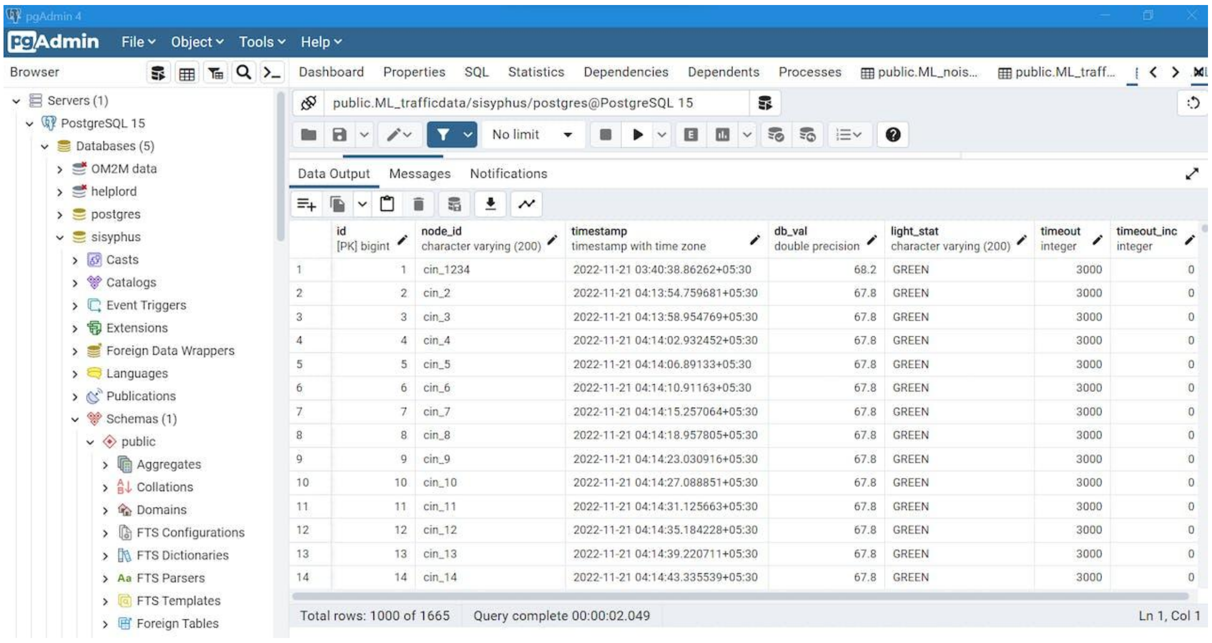
The data from the server is pushed to OM2M and using the subscription functionality, the Django server is now subscribed to the OM2M server. Django is an open-source Python framework used for highly functioning web applications, it uses the options method to send data when an HTTP request is sent and 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': '20221123T17
0551', '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': '20221123T17
0555', '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.



	id [PK] bigint	node_id character varying (200)	timestamp timestamp with time zone	db_val double precision	light_stat character varying (200)	timeout integer	timeout_inc integer
1	1	cin_1234	2022-11-21 03:40:38.86262+05:30	68.2	GREEN	3000	0
2	2	cin_2	2022-11-21 04:13:54.759681+05:30	67.8	GREEN	3000	0
3	3	cin_3	2022-11-21 04:13:58.954769+05:30	67.8	GREEN	3000	0
4	4	cin_4	2022-11-21 04:14:02.932452+05:30	67.8	GREEN	3000	0
5	5	cin_5	2022-11-21 04:14:06.89133+05:30	67.8	GREEN	3000	0
6	6	cin_6	2022-11-21 04:14:10.91163+05:30	67.8	GREEN	3000	0
7	7	cin_7	2022-11-21 04:14:15.257064+05:30	67.8	GREEN	3000	0
8	8	cin_8	2022-11-21 04:14:18.957805+05:30	67.8	GREEN	3000	0
9	9	cin_9	2022-11-21 04:14:23.030916+05:30	67.8	GREEN	3000	0
10	10	cin_10	2022-11-21 04:14:27.088851+05:30	67.8	GREEN	3000	0
11	11	cin_11	2022-11-21 04:14:31.125663+05:30	67.8	GREEN	3000	0
12	12	cin_12	2022-11-21 04:14:35.184228+05:30	67.8	GREEN	3000	0
13	13	cin_13	2022-11-21 04:14:39.220711+05:30	67.8	GREEN	3000	0
14	14	cin_14	2022-11-21 04:14:43.335539+05:30	67.8	GREEN	3000	0

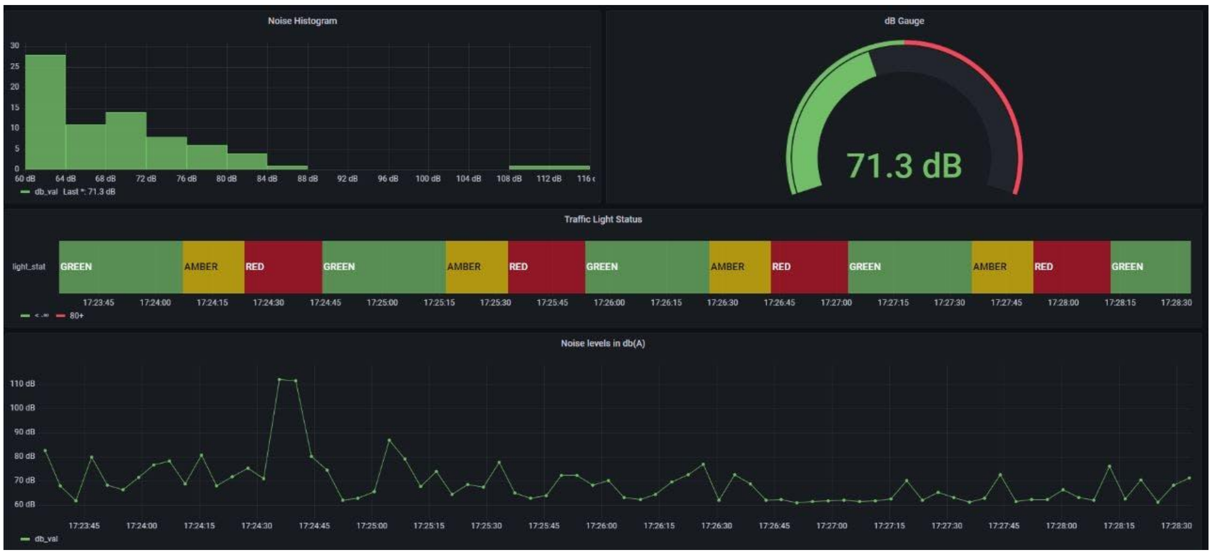
```
Total rows: 1000 of 1665    Query complete 00:00:02.049    Ln 1, Col 1
```

Dashboard - Data Analysis

We're then going to be 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.

We 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.



We set up a Bar gauge and a histogram graph for dB value metrics from the database table. We 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.