NOISE POLLUTION MONITORING

Introduction:

In a world increasingly defined by digital connectivity and smart technology, the development of IoT (Internet of Things) devices has become a remarkable intersection of innovation and practicality. One such real-world application is an IoT-based sound level monitoring system, a project that exemplifies the power of technology to enhance our daily lives and address pressing concerns.

Imagine a bustling urban environment, where the symphony of life is composed of sounds both harmonious and disruptive. The need to comprehend and manage ambient sound levels has never been more critical. From ensuring peace in residential neighborhoods to upholding safety in industrial settings, the impact of noise pollution is a real concern.

Our IoT sound level monitoring system represents a solution to this challenge, offering real-time insights into ambient sound levels and facilitating data-driven decisions. This project is designed to address the practical needs of a community or an industry by providing an easy-to-use, yet sophisticated, monitoring solution.

In this introduction, we'll embark on a journey to explore the objectives, setup, platform integration, and code implementation of this IoT device. It is a testament to how technology can bridge the gap between data and human experience, providing us with the tools to quantify and improve the sonic environment we inhabit.

The core objectives of this project encompass real-time sound level monitoring, immediate visualization on a LCD screen, long-term data logging to a cloud platform, and user notification of sound level thresholds. As we delve deeper into the project, we'll unravel the intricate web of hardware components, sensor calibration, Wi-Fi connectivity, and cloud integration that bring this IoT device to life.

Whether you're a technology enthusiast seeking to replicate this project or an individual concerned about the impact of noise pollution, this documentation will serve as your guide. It's a journey that takes us from the drawing board to real-world application, demonstrating the immense potential of IoT technology to improve the quality of our lives and environments.

Project Objectives:

Objective: The project aims to develop an IoT-based sound level monitoring system using an ESP8266-based Arduino microcontroller and a sound sensor. This system's primary objectives include real-time sound level monitoring, displaying sound levels on an LCD screen, logging data to ThingSpeak, and notifying users about sound level ranges falling into the "Moderate" or "High" categories.

Sound Level Monitoring: The core objective of this project is to accurately measure and monitor ambient sound levels in decibels (dB). To achieve this, we employ a sound sensor (microphone) connected to the analog pin A0 of the ESP8266. The system continuously samples audio data and calculates peak-to-peak amplitudes, allowing for precise sound level measurements.

Real-time Display: Real-time visualization of sound levels is a crucial aspect of the project. To accomplish this, a 16x2 LCD screen is used to display the current sound level in dB. The LCD screen provides immediate feedback to users, allowing them to assess the noise levels in their environment quickly and easily.

Data Logging to ThingSpeak: One of the project's key objectives is to enable data logging for long-term analysis. To fulfill this objective, we integrate with ThingSpeak, a cloud platform designed for IoT applications. The system sends sound level data to ThingSpeak using HTTP POST requests. The project makes use of a specific ThingSpeak Write API key for authentication and posts data to designated fields within the ThingSpeak channel. This enables comprehensive data collection and historical analysis of sound levels.

Alerting Users: The system goes beyond mere data collection and monitoring by incorporating a user notification feature. When sound levels fall into the "Moderate" or "High" categories (dB > 55), the project sends notifications to users via the Blynk app. This objective adds an alerting system that informs users about potential noise pollution, enhancing their awareness of their surroundings.

In summary, the project objectives encompass sound level monitoring, real-time display, data logging to ThingSpeak, and user alerting. Together, these objectives create a sophisticated IoT-based sound monitoring system that offers both immediate feedback and historical data for analysis, contributing to a quieter and more informed environment.

IoT Device Setup:

Hardware Components: The project employs a range of hardware components, carefully selected to meet the project's requirements:

Arduino Board (ESP8266): The ESP8266-based Arduino board was chosen for its compatibility with IoT applications and Wi-Fi connectivity capabilities. This powerful microcontroller is at the heart of the system, managing data acquisition and transmission.

Sound Sensor (Microphone): To capture ambient sound levels, a sound sensor is integrated into the system. It is connected to the analog pin A0 of the ESP8266.

LCD Display: For real-time data visualization, a 16x2 LCD screen is used. The screen displays the current sound level in dB, allowing users to gauge their environment's noise levels instantly.

LED Indicators: To provide visual indications of sound level ranges ("Quiet," "Moderate," and "High"), LEDs are connected to digital pins D3, D4, and D5. These LEDs serve as a quick reference for users to assess the noise levels visually.

Wi-Fi Module: The ESP8266 module handles Wi-Fi connectivity, enabling data transmission to the cloud. This module ensures that the system can communicate with ThingSpeak and Blynk for remote monitoring and notifications.

Power Supply: To power the system, a suitable power source is employed, ensuring reliable and continuous operation. The power supply plays a crucial role in maintaining the functionality of the IoT device.

Wiring and Connections: To ensure the system operates as intended, precise wiring and connections are essential. The project's components are connected as follows:

1.) **Sound Sensor:** The sound sensor is connected to the analog pin A0 of the ESP8266. This connection allows the microcontroller to collect analog audio data from the sensor for sound level measurement.

2.) **LED Indicators**: LED indicators are connected to digital pins D3, D4, and D5. Each LED corresponds to a specific sound level range ("Quiet," "Moderate," and "High") and is controlled based on the dB value.

3.) **LCD Display:** The 16x2 LCD screen is interfaced using the I2C bus through the I2C interface. This connection simplifies the communication between the ESP8266 and the LCD, making it more efficient.

Each connection is carefully established to ensure the accurate flow of data and control signals, allowing for the seamless operation of the IoT device.

Sensor Calibration: To ensure the sound sensor provides accurate measurements, calibration is essential. The sound sensor is calibrated to map analog readings to dB levels accurately. Calibration is achieved through the mapping of peak-to-peak amplitudes in the analog data to dB values. This calibration process guarantees that the sound levels displayed on the LCD screen and sent to ThingSpeak accurately represent the ambient noise levels.

In conclusion, the IoT device setup encompasses the selection of appropriate hardware components, precise wiring and connections, and the crucial calibration of the sound sensor. Each element plays a critical role in ensuring the system functions as intended, providing accurate sound level monitoring and user feedback.

Platform Development:

Wi-Fi Connectivity:

- The project relies on Wi-Fi connectivity to enable data transmission to cloud platforms and user notifications. Wi-Fi connectivity is a crucial component of the IoT system's functionality.

- The system connects to a Wi-Fi network using the provided SSID and password. Wi-Fi credentials must be configured to establish a reliable connection.

- Stability in Wi-Fi connectivity is essential. The code includes error handling and retry mechanisms to ensure that the IoT device maintains a stable connection, even in the presence of network disruptions.

ThingSpeak Integration:

- ThingSpeak, a cloud platform designed for IoT applications, serves as the destination for data logging and storage. The system is integrated with ThingSpeak to enable long-term data analysis.

- ➢ Data is transmitted to ThingSpeak using HTTP POST requests. These requests are formed to include specific data, including the sound level in dB, and are sent to ThingSpeak's servers for processing.

- ➢ Authentication is achieved through the use of a ThingSpeak Write API key. This key allows the system to post data to the specified ThingSpeak channel.

- ➢ Data is posted to designated fields within the ThingSpeak channel. The separation of data into fields ensures that sound level measurements are correctly organized for historical analysis.

Blynk Integration:

- ➢ The Blynk app is employed to enhance user interaction with the IoT system. It enables user notifications based on sound level ranges, adding an alerting feature to the project.

- ➢ Blynk is configured to send notifications when sound levels fall into the "Moderate" or "High" categories. Users are alerted to potential noise pollution, increasing their awareness of their surroundings.
- ➢ The integration with Blynk is designed to work seamlessly with the IoT device, ensuring that notifications are sent when specific conditions are met.

In summary, platform development involves setting up Wi-Fi connectivity, integrating with ThingSpeak for data storage, and leveraging Blynk for user notifications. These elements collectively form the foundation of the IoT system, enabling real-time monitoring, data analysis, and user alerting.

Code Implementation:

Sensor Readings:

- ➢ The code is responsible for acquiring sound level data from the sound sensor and processing it to obtain accurate measurements.

- ➢ Analog readings are obtained from the sound sensor connected to the analog pin A0 of the ESP8266. These readings represent the amplitude of the ambient sound.

- ➢ To ensure data accuracy, the code filters out spurious readings, which may result from noise or other interference. Filtering out unreliable data ensures that only valid sound level measurements are considered.

- ➢ The code calculates peak-to-peak amplitudes by identifying the minimum and maximum values in the collected analog data. Peak-to-peak amplitudes are used to derive the sound level in dB.

dB Mapping:

- ➢ The mapping of peak-to-peak amplitudes to dB values is a critical part of the code. This mapping process ensures that the sound level measurements are accurate and meaningful.

- ➢ Peak-to-peak values, calculated from the sound sensor data, are mapped to decibels using a specific calibration equation. This equation accounts for the characteristics of the sound sensor and the relationship between amplitude and sound level.

- ➢ The calibration ensures that the sound level displayed on the LCD screen and sent to ThingSpeak accurately represents the ambient noise levels. Calibration is an essential step to provide meaningful data to users and cloud platforms.

LCD Display:

- ➢ The code enables real-time visualization of sound levels on the 16x2 LCD screen.

- ➢ The sound level in dB is displayed on the LCD screen, allowing users to monitor noise levels in their environment continuously.

- ➢ The LCD screen provides immediate visual feedback, making it easy for users to gauge sound levels at a glance.

LED Indicators:

- ➢ LED indicators are used to provide visual indications of sound level ranges, enhancing user awareness.

- LEDs connected to digital pins D3, D4, and D5 represent the "Quiet," "Moderate," and "High" sound level ranges.

- LEDs are controlled based on the current dB value. For instance, if the dB value indicates "Quiet," the corresponding LED is illuminated, providing a visual cue to users.

ThingSpeak Data Posting:

- The code is responsible for sending sound level data to ThingSpeak for long-term data storage.

- Data is posted to ThingSpeak using HTTP POST requests. These requests are carefully constructed to include the API key and the sound level data in the proper format.

- The code ensures that the data is posted to specific fields within the ThingSpeak channel, allowing for organized data storage and analysis.

- Proper formatting of the POST request is a critical aspect of the code implementation to ensure that ThingSpeak processes the data correctly.

Blynk Notifications:

- The integration with Blynk adds a user alerting feature to the project, allowing users to receive notifications when sound levels indicate potential noise pollution.

- Blynk.notify() is used to send notifications to users when specific conditions are met. Specifically, notifications are sent for sound levels falling into the "Moderate" or "High" categories.

- The code is designed to work seamlessly with the Blynk app, ensuring that notifications are sent promptly and accurately to enhance user awareness.

Error Handling:

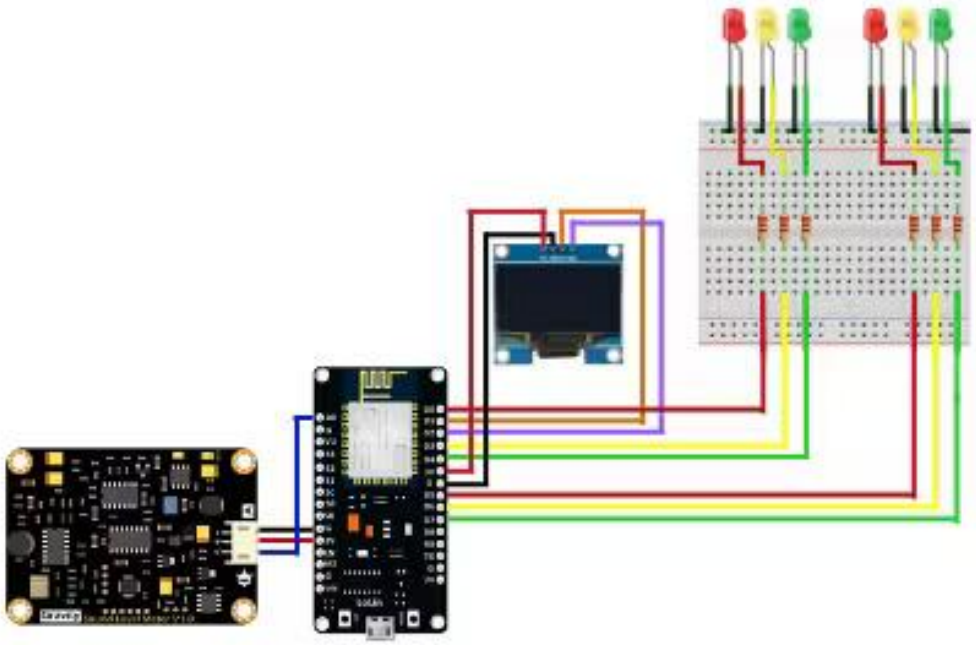
- Basic error handling is implemented in the code to ensure the system's reliability in various conditions.

- ➢ Error handling mechanisms are incorporated to address potential issues with network connectivity and data posting. These mechanisms ensure that the IoT device can recover from network disruptions and continue to operate effectively.

In summary, the code implementation encompasses a wide range of functions, from collecting and processing sound sensor data to visualizing sound levels, posting data to ThingSpeak, and sending user notifications through Blynk. Each component of the code contributes to the overall functionality of the IoT system, ensuring accurate sound level monitoring and user interaction

By addressing each of the project's objectives, explaining the IoT device setup, detailing the platform development, and delving into the code implementation, this documentation provides a thorough understanding of the project's intricacies. It serves as a comprehensive guide for anyone interested in replicating or understanding the project's functionality.

Schematic diagram:



The schematic diagram of the IoT-based Noise Pollution Monitoring System is an intricate depiction of how all the hardware components are seamlessly interconnected to form a functional and efficient noise monitoring solution. At the core of the schematic is the NodeMCU ESP8266 Breakout Board, which acts as the central processing unit and control center of the system.

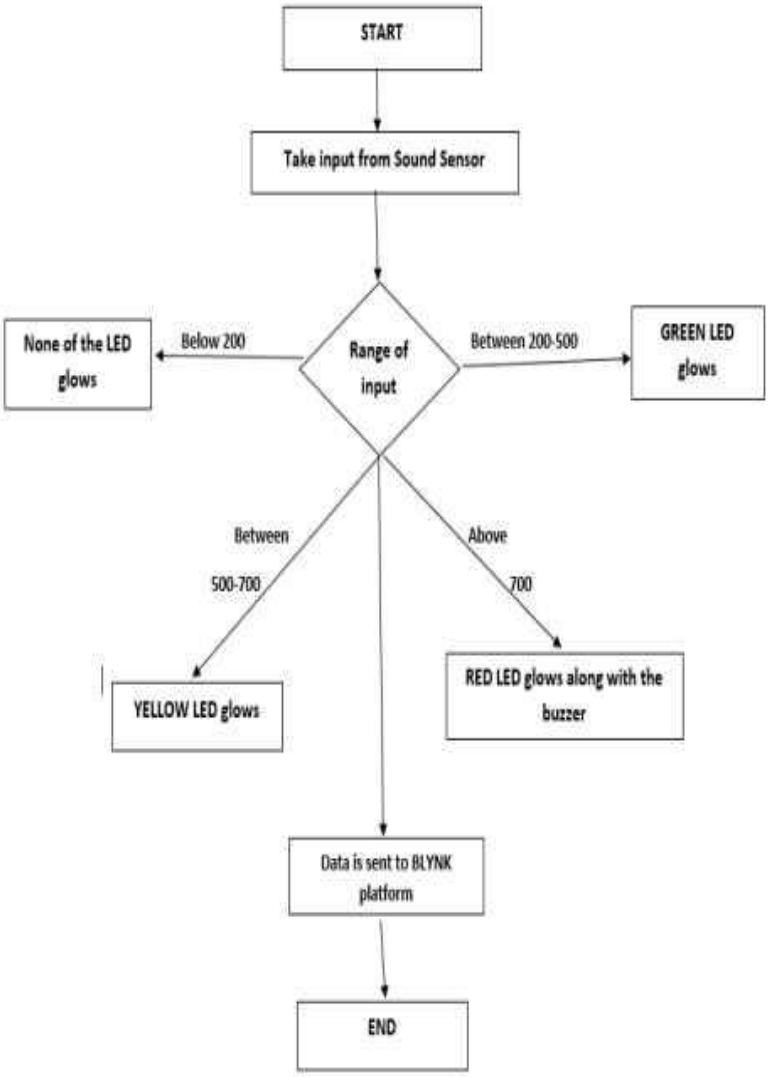
The DFRobot Gravity Analog Sound Level Meter, a sophisticated sound measurement device, is meticulously integrated into the schematic. It showcases the precise connections, wiring, and interface with the NodeMCU, highlighting how sound data is collected and transmitted for further analysis.

The 0.96" OLED 64x128 Display Module is showcased in the diagram, illustrating its role in displaying critical information to the users. Its connection to the NodeMCU and the seamless data transfer protocol are highlighted, emphasizing its importance in conveying real-time noise pollution data.

The LEDs, consisting of two red, two yellow, and two green indicators, are pivotal in the project. The schematic reveals their placement and electrical connections, showing how they replicate the functions of a traffic signal to inform commuters of the noise pollution levels. The inclusion of six 220-ohm resistors is detailed, illustrating how they protect the LEDs and ensure their longevity.

The entire schematic diagram encapsulates the complexity of the project, and it serves as a visual roadmap for developers and enthusiasts, guiding them through the hardware connections that enable this innovative noise pollution monitoring system to function effectively.

Flow Diagram:



1. Data Acquisition:

- ➢ The system acquires real-time noise data using the DFRobot Gravity Analog Sound Level Meter.

- ➢ The collected noise data is in decibels with 'A' weighting (dB(A)), aligning with human sound perception.

2. NodeMCU Processing:

- ➢ The NodeMCU ESP8266 Breakout Board processes the noise data, implementing noise level analysis and control logic.

- ➢ Decision-making for traffic signal light changes is performed by the NodeMCU, adjusting signal timings based on noise levels.

3. Traffic Signal Logic:

- ➢ The system's traffic signal control logic defines when to change signal lights and how to impose penalties for excessive honking.

- ➢ Traffic signal state transitions are controlled by the NodeMCU, which adjusts timings and penalties as necessary.

4. Data Transmission:

- ➢ The NodeMCU sends the processed data to a remote server over Wi-Fi, including noise level, signal status, and countdown timers.

- ➢ Reliable and secure communication protocols are used for data transmission.

5. Data Visualization and Storage:

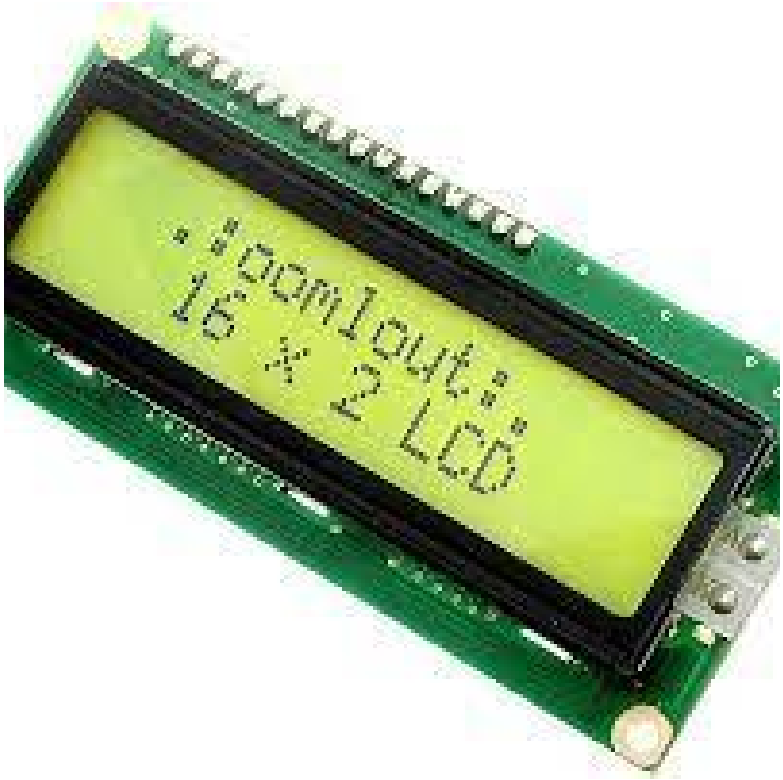
- ➢ The system integrates with an OM2M Eclipse server for data handling, forwarding information to a Django server for management.
- ➢ Data is stored in a central PostgreSQL database, and a Grafana dashboard visualizes and analyzes the noise pollution data, making it accessible to stakeholders.

IoT Devices:

1.) ESP8266-based Arduino Board

2.) Sound Sensor (Microphone):

3.) 16x2 LCD Display:

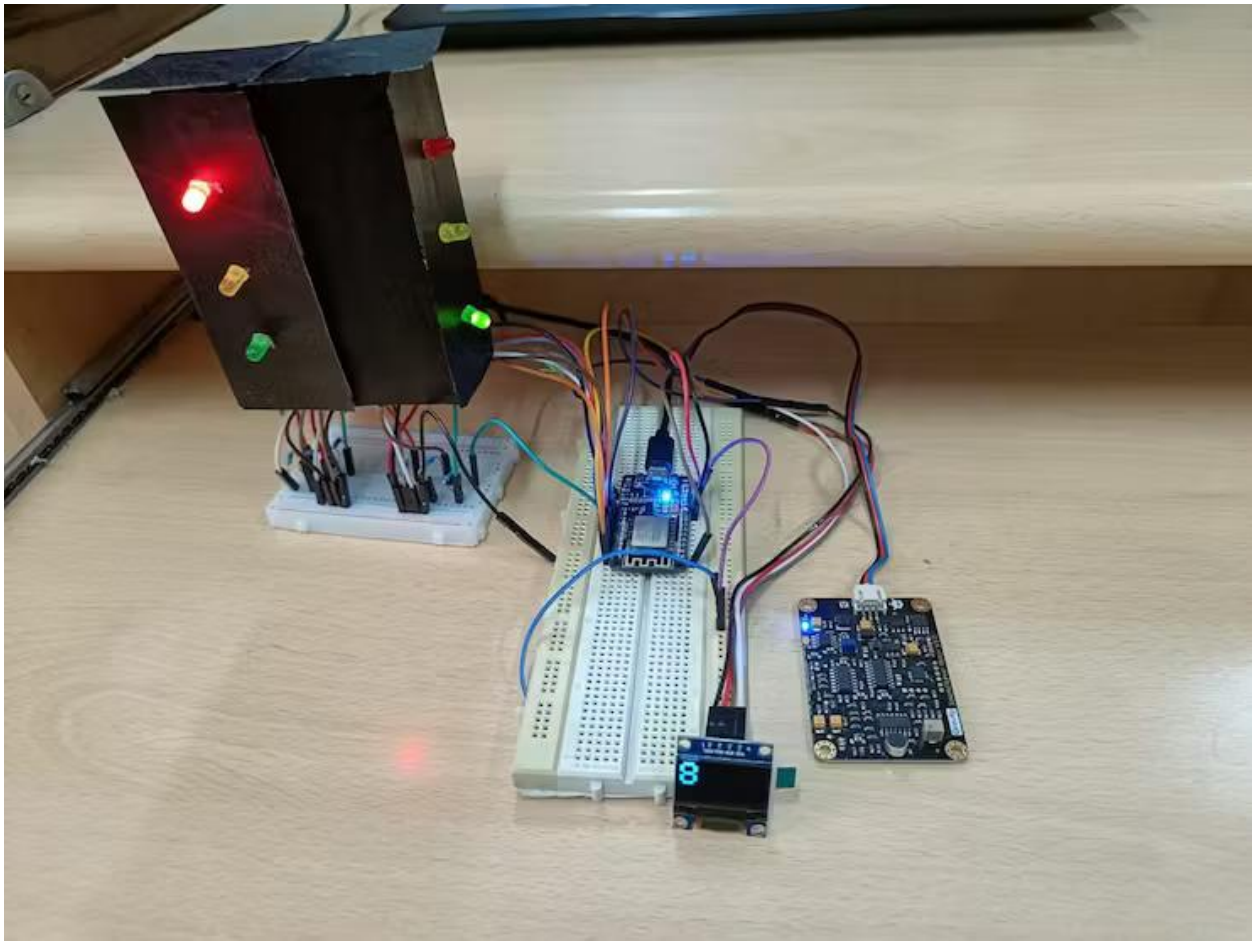


4.) LED Indicators (Quite / Moderate / High):

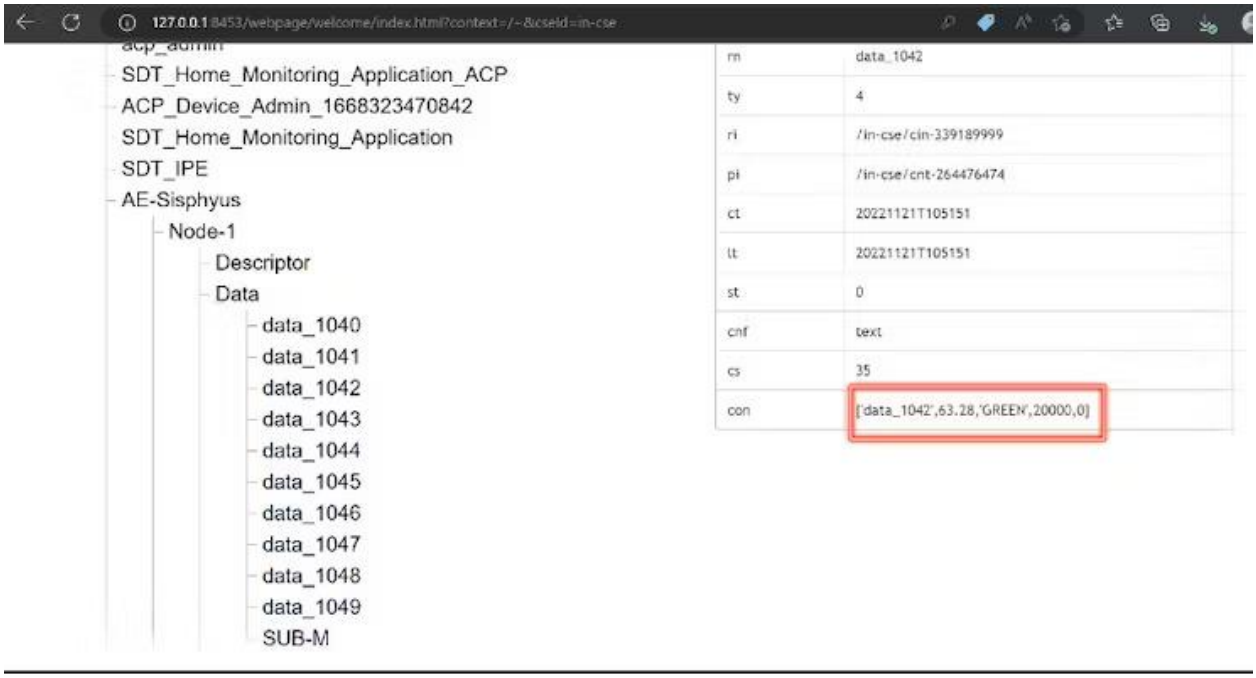
5.) Wi-Fi Module (ESP8266):

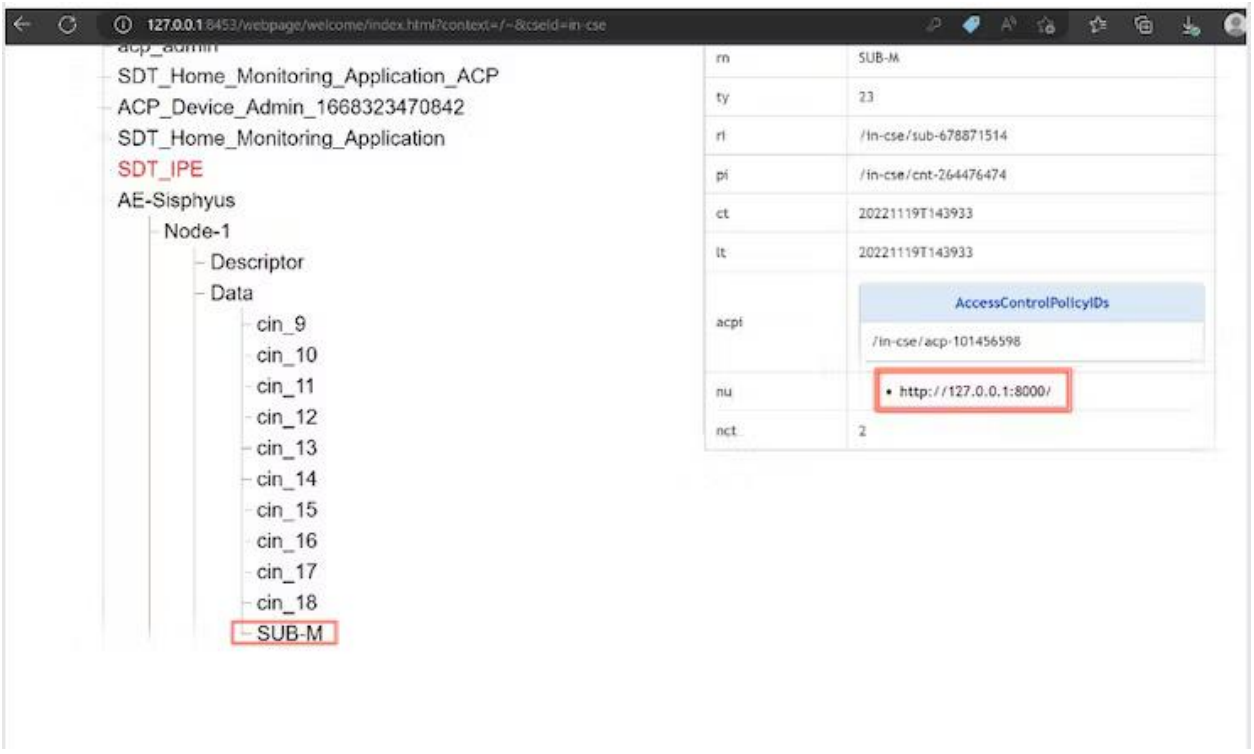
The project employs an **ESP8266-based Arduino board** as the central control unit, facilitating data processing and communication. It incorporates a sound sensor (microphone) for real-time noise level measurement. The system provides feedback through a **16x2 LCD display** and utilizes **LED indicators** for noise level categorization into "Quiet," "Moderate," and "High." **The Wi-Fi module** (ESP8266) ensures seamless data transmission to remote servers, enabling comprehensive noise pollution monitoring.

Prototype

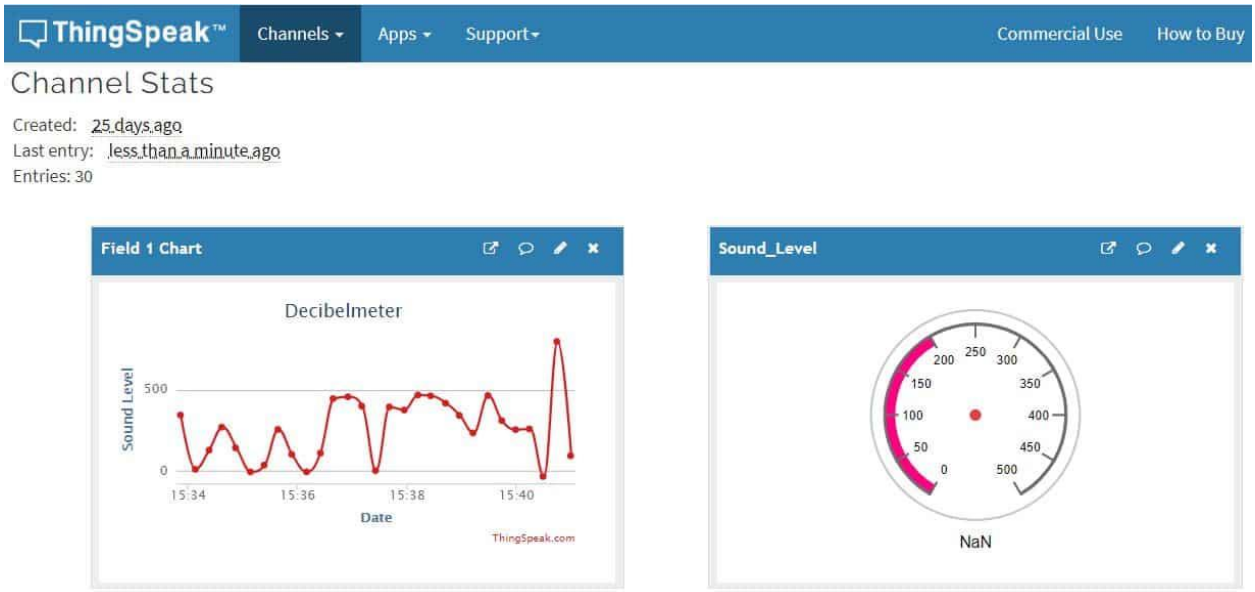


Data Sharing platform:





Thingspeak Dashboard:



Project in detail:

Project Overview:

The "OneM2M and IoT based Noise Pollution Monitoring System" addresses a pressing concern often overlooked—noise pollution. Despite its profound impact on public health, noise pollution tends to be underappreciated. In response to the escalating noise levels in urban areas, particularly near traffic junctions, this project introduces a comprehensive solution to both monitor and combat noise pollution effectively. The project's objective is to raise awareness of noise pollution's adverse effects and encourage commuters to reduce honking, especially in locations with heavy traffic.

Problem Statement:

Noise pollution, with its short and long-term implications for human health, is a pervasive issue. Regulatory guidelines for permissible noise levels exist, but adherence remains inconsistent. Urbanization and increased vehicular traffic have further exacerbated noise pollution. Honking, particularly at traffic junctions, stands as a primary source of noise pollution, necessitating urgent attention.

Solution:

The project proposes a multifaceted solution, primarily focusing on monitoring noise levels and implementing measures to discourage unnecessary honking. The key components of the project are a noise monitoring node deployed at traffic junctions and a prototype traffic light controller. Together, these components aim to create awareness about noise pollution and prompt behavioral change in commuters. The system is designed to collect real-time noise data, categorize noise levels, and adapt traffic signal timing based on noise conditions.

How It Works:

The system leverages the ESP8266 microcontroller, interfaced with the DF Robot Analog Sound Level Meter sensor, to measure noise levels in decibels with 'A' weighting (dB(A)), mirroring human sound perception. The traffic signal prototype, which controls signal transitions, operates using Arduino IDE. When noise levels exceed a predefined threshold during the red signal phase, the project imposes additional time as a penalty for excessive honking. The noise data, signal status, and countdown timer information are transmitted to the OM2M Eclipse server, and further data handling is achieved

through a Django server and PostgreSQL database. The data is then used to create a Grafana dashboard for in-depth data analysis and visualization.

Prerequisites:

To implement this project, specific prerequisites are required, including Arduino IDE 1.8.5 or later for sensor node code compilation and uploading, JAVA 1.8 for running the OneM2M platform, Eclipse OM2M v1.4.1 for project implementation, NodeJS v14, Postman, and Grafana v9.2 or later for data analysis and visualization.

Actuators - Display Message:

The project generates messages to be displayed at traffic junctions during the imposed amber signal phase. An OLED connected to the ESP8266 serves this purpose, providing commuters with a visual indication of the noise pollution situation.

Scenarios and Scope of the Prototype:

The system allows flexibility in adjusting the threshold values according to different times of the day and recommended noise levels based on WHO guidelines. Additionally, the traffic signal prototype can be extended to accommodate various intersection configurations, including three-way and four-way junctions. The OneM2M platform is versatile, capable of collecting data from multiple noise pollution monitoring systems across the city.

OneM2M Implementation:

The implementation of the OneM2M platform involves configuring the IoT platform, creating a resource tree using Postman, pushing data to the OM2M Eclipse server, and establishing subscriptions to a Django server.

High-Level Software/Hardware Architecture:

The project's architecture incorporates the DF Robot sound sensor, NodeMCU microcontroller, and an actuator (OLED) for display purposes. Data transmission occurs between these components and the OM2M Eclipse server, which subsequently connects to a Django server for data handling. Data is stored in a PostgreSQL database and made accessible through a Grafana dashboard for visualization and analysis.

Collecting Data from OM2M Server and Posting To PostgreSQL:

Data collected from the server is pushed to the OM2M platform, and subscriptions notify the Django server of modifications. Using Django, the data is sent in JSON format, allowing the creation of a PostgreSQL database.

Dashboard - Data Analysis:

The project employs Grafana to create a comprehensive dashboard for data visualization and analysis. Various panels are used to display noise pollution metrics, signal timing, and historical data trends.

Demonstration:

The project provides an in-depth demonstration, featuring video links showcasing the Arduino implementation and OneM2M platform deployment.

This project, aimed at tackling noise pollution at traffic junctions, stands as a significant and innovative solution that combines IoT technology, data analysis, and real-time intervention to create a quieter and healthier urban environment.