Cloud-Enabled Industrial Pollution Monitoring & Data Management Systems

Dr Mohini Vyawahare, Rohit. D. Raut, Prof Sandhya Patil

Priyadarshini College of Engineering, Nagpur, Maharashtra

Priyadarshini College of Engineering, Nagpur, Maharashtra

Priyadarshini College of Engineering, Nagpur, Maharashtra

mohini.vyawahare@pcenagpur.edu.in

rohitraut2623@gmail.com

sandhyapatil7864@gmail.com

Abstract: The project "Cloud-Enabled Industrial Pollution Monitoring & Data Management Systems" addresses the critical need for accurate and localized air quality monitoring in industrial environments. Leveraging Internet of Things (IoT) technologies, the project aims to revolutionize pollution monitoring systems by providing real-time and comprehensive data tailored to specific industry needs.

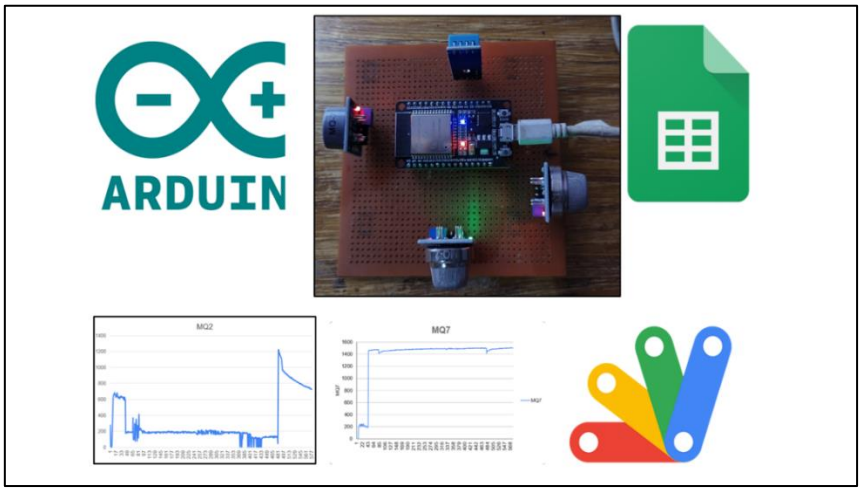
Keywords: Industrial Pollution Monitoring, Internet of Things (IoT), ESP32, Pollution Monitoring Systems, Data Management, Environmental Impact Assessment.

1. Introduction

In today's rapidly evolving industrial landscape, the integration of Internet of Things (IoT) has become paramount. Not only does IoT revolutionize operational efficiency and productivity, but it also plays a pivotal role in addressing pressing environmental concerns. By leveraging advanced IoT infrastructure, industries are empowered to combat environmental pollution more effectively than ever before. This is particularly crucial in heavy industries such as power plants, where the emission of harmful gases poses significant challenges. Furthermore, IoT technologies align seamlessly with modern business goals cantered around Economic Social Governance (ESG), Carbon Neutrality, and Sustainable Development Goals (SDGs). It is within this context that the project "Cloud-Enabled Data Management for Industrial Air Quality Monitoring Systems" emerges, aiming to bridge the gap in current pollution monitoring systems. By providing real-time and comprehensive data on air quality parameters, this project not only empowers stakeholders but also adds a new dimension by involving data scientists in the planning process, recognizing data as the new oil. Moreover, it facilitates the management of carbon credits and environmental impact assessment, thus contributing to a more sustainable and environmentally conscious industrial ecosystem.

At the heart of this project lies the utilization of ESP32 technology, renowned for its versatility and efficiency in IoT applications. The project architecture comprises five layers: the Sensor Data Layer, responsible for collecting raw data from various sensors including DHT11, MQ135, MQ7, and MQ2; the Data Acquisition and Processing Layer, where collected data undergoes preprocessing and filtering; the Communication Layer, facilitating seamless data transmission between the ESP32 module and the Cloud Server Layer; and finally, the User Interface Layer, providing stakeholders with intuitive access to real-time air quality data.'

Existing air quality monitoring systems, typically overseen by pollution control boards, often fall short in providing accurate and localized data, particularly for industries. These existing systems rely on Air Quality Index (AQI) monitors placed in regions distanced from industrial facilities, resulting in data that may not accurately reflect on-site pollution levels. In contrast, the proposed project seeks to integrate directly with industrial infrastructure, providing relevant and independent data tailored to specific industry needs and geographic locations. This approach promises to revolutionize air quality monitoring within industrial settings, paving the way for more effective pollution management strategies.



2. Literature Review

Industrial air quality monitoring has become an increasingly critical area of focus in recent years due to growing environmental concerns and regulatory requirements. Various studies have highlighted the adverse effects of air pollution on public health, ecosystems, and climate change, emphasizing the need for robust monitoring systems.

Gas sensors like the MQ135, MQ2, and MQ7 play a crucial role in detecting specific air pollutants emitted by industrial processes. Research conducted by H. Parikh et al. (2020) [1] has explored the application of MQ135 sensors for monitoring indoor air quality, highlighting their effectiveness in detecting harmful gases such as CO2 and CO. Similarly, studies by S. Singh et al. (2019) [2] have investigated the performance of MQ2 and MQ7 sensors in detecting combustible gases and carbon monoxide, respectively, demonstrating their utility in industrial environments.

The integration of IoT technologies, such as the ESP32 microcontroller, enables real-time data collection, processing, and transmission in industrial air quality monitoring systems. Research by A. Al-Fuqaha et al. (2015) [3] discusses the role of IoT in environmental monitoring, emphasizing its potential for enhancing data accuracy, scalability, and accessibility. Furthermore, studies by H. Kim et al. (2019) [4] have highlighted the advantages of using ESP32-based sensor nodes for environmental monitoring applications, citing their low power consumption, wireless connectivity, and compatibility with various sensors.

Cloud computing platforms, such as Google Apps Script and Google Sheets, provide scalable and cost-effective solutions for data storage, analysis, and visualization in industrial air quality monitoring. Literature by P. Bhatia et al. (2017) [5] discusses the use of cloud-based platforms for environmental data management, emphasizing their role in facilitating real-time decision-making and stakeholder collaboration.

3. Problem Statement

The existing practices of collecting pollution data from industries are outdated, relying on methodologies that lack accuracy and fail to address the dynamic nature of industrial emissions. This outdated approach raises several critical issues:

Inaccurate Data Collection: The fault lines in the accuracy of the collected data create uncertainties in understanding the true extent of industrial pollution. This can lead to ineffective mitigation strategies and an incomplete understanding of environmental impact.

Regional Monitoring Injustice: The current air quality monitoring system operates on a regional scale, neglecting the specific geographic location of industries. This oversight fails to capture the nuances of localized pollution levels, hindering targeted pollution control efforts.

Off-Site Data Sampling: Data sampling of pollutants relies on the geographic region rather than on-site measurements, resulting in a mismatch between the reported data and the actual emissions from industrial facilities. Limited Parameters: The current data scope is restricted, primarily focusing on

AQI (Air Quality Index) in parts per million (ppm) and humidity. This limitation overlooks crucial parameters, providing an incomplete picture of the diverse pollutants emitted by industries.

Lack of Comprehensive Data Storage: The absence of a proper data storage mechanism exacerbates the challenges in managing and utilizing pollution data effectively. This gap hinders the development of informed decision-making processes.

Industry-Lacking Pollution Monitoring Systems: Many industries lack dedicated pollution monitoring systems, leaving them uninformed about their own environmental impact. This deficiency inhibits industries from implementing targeted pollution reduction strategies.

Outdated Technologies: The current approach often relies on outdated technologies such as GSM (Global System for Mobile Communications) and WSN (Wireless Sensor Networks), hindering the adoption of more advanced and efficient IoT infrastructure.

Underutilization of IoT Infrastructure: The wide-scale adoption of IoT infrastructure in industries remains a critical need. The gap between the potential benefits of IoT and its limited implementation in pollution monitoring creates inefficiencies in data collection and analysis.

Lack of Integration: A substantial gap exists between independent pollution monitoring, data storage, and visualization. Additionally, the conventional methods Current methods fall short of leveraging the potential of new-age IoT and data analytics, preventing enterprises from reaping the full advantages of technology breakthroughs.

4. Project objective

I. Develop an IoT-based air quality monitoring system utilizing cutting-edge sensor technologies and connectivity standards.
II. Establish seamless communication and data transmission between sensor nodes and a centralized data processing unit.
III. Implement advanced analytics and visualization techniques to derive actionable insights from the collected sensor data.
IV. Incorporate machine learning algorithms to enhance the accuracy and predictive capabilities of the monitoring system.
V. Provide stakeholders with intuitive dashboards and cloud interfaces for accessing real-time air quality data and insights.

5. Project Proposal

Our project proposes the deployment of a comprehensive IoT infrastructure comprising sensor nodes, communication modules, data processing units, and cloud-based analytics platforms. The sensor nodes, equipped with advanced gas sensors such as MQ135, MQ2, MQ7, and environmental sensors like DHT11, will be strategically deployed across industrial facilities to capture real-time data on air quality parameters. These sensor nodes will communicate wirelessly with gateway devices using protocols like Wi-Fi enabling seamless data transmission to cloud-based servers.

On the cloud side, advanced data analytics algorithms will be employed to process and analyse the collected sensor data in real-time. Through intuitive dashboards and visualization tools, stakeholders will gain actionable insights into pollution levels, enabling informed decision-making and proactive pollution control measures. Additionally, the system will support remote monitoring and management capabilities, allowing stakeholders to access real-time data and receive timely alerts on air quality anomalies via web or mobile interfaces

6. Methodology

1. Hardware

I. ESP32 Microcontroller Board

The ESP32 is a versatile and powerful microcontroller developed by Espressif Systems. It is used in various IoT (Internet of Things) applications due to its robust features and capabilities.

Architecture of ESP32

1. Power Pins: -

Pins	Description
VCC	This is supplying voltage pin with 3.3v rating.
Vin	This pin enables the ESP Module to power from external source.
GND	This pin connected to ground pin of power supply with 0v rating.

2. General Purpose Input Output (GPIO) Pins

The ESP32 microcontroller features a significant number of GPIO (General Purpose Input Output) pins that can be configured for various purposes including digital input/output, PWM (Pulse Width Modulation) output, I2C, SPI, UART, and more. The exact number and configuration of GPIO pins may vary depending on the specific ESP32 module or development board being used.

In general, the ESP32 typically has several dozen GPIO pins, labelled GPIO0 through GPIO39. These pins can be used for various purposes and are often multiplexed with other functionalities. Some GPIO pins may also have specific hardware capabilities such as capacitive touch sensing.

GPIO Pins	ESP32 Pin
GPIO1	TX0
GPIO2	D2
GPIO3	RXD40
GPIO4	D5
GPIO5	D12
GPIO12	D13
GPIO13	D14
GPIO14	D15
GPIO15	RX2
GPIO16	TX2
GPIO17	D18
GPIO18	D19
GPIO19	D21
GPIO21	D22
GPIO22	D23
GPIO23	D25
GPIO24	D26
GPIO25	D27
GPIO26	D32

GPIO32	D33
GPIO33	D34
GPIO34	D35
GPIO35	VCC
GPIO39	Vin

D Pin = Digital pin

TX Pin = Transmitter pin

RX = Reviver Pin

3. I2C (Inter-Integrated Circuit) Interface of ESP32:

The ESP32 microcontroller includes hardware support for the I2C (Inter-Integrated Circuit) communication protocol, allowing it to communicate with other devices using I2C. Here's a brief overview of the I2C interface on the ESP32:

I2C Pins: The ESP32 typically has dedicated pins for I2C communication. These pins are usually labeled SDA (data line) and SCL (clock line). You'll need to connect these pins to the corresponding SDA and SCL pins on the device you want to communicate with.

Hardware I2C Controller: The ESP32 features built-in hardware support for I2C communication, making it efficient and reliable for interfacing with I2C devices. The hardware I2C controller handles the timing and protocol requirements of I2C communication, freeing up the CPU for other tasks.

I2C Library Support: The ESP32 SDK provides a built-in I2C library that allows you to easily configure and communicate with I2C devices in your firmware code. Additionally, there are various third-party libraries and frameworks available for working with I2C on the ESP32, which can simplify development and provide additional features.

I2C Speed: The ESP32 supports various I2C bus speeds, including standard (100 kHz) and fast (400 kHz) modes. Some ESP32 modules may support even higher speeds depending on the specific hardware configuration.

I2C Configuration: In your ESP32 firmware code, you'll typically need to configure the I2C pins, initialize the I2C bus, and then use functions provided by the I2C library to communicate with specific I2C devices. This involves specifying the I2C address of the target device, writing data to the device, reading data from the device, and handling any errors that may occur during communication.

I2C Communication	**ESP Pin**
SCL (Serial Clock Line)	D17
SDL (Serial Data Line)	D16

4. DAC Pins of ESP32

The ESP32 microcontroller typically features built-in DAC (Digital-to-Analog Converter) pins that allow it to output analog voltage signals. These pins can be used for various applications such as audio output, analog signal generation, and more. However, it's important to note that not all ESP32 modules or development boards may have DAC pins available, and the exact number and configuration of DAC pins may vary depending on the specific module or board.

In general, the ESP32 commonly features two DAC pins, which are usually labelled DAC1 and DAC2. These pins can output analog voltages range of 0 to 3.3 volts, corresponding to digital values between 0 and 255 (8-bit resolution).

When using the DAC pins on the ESP32, it's essential to configure them properly in your firmware code and ensure that the connected circuitry can handle the specified output voltage range. Additionally, some

ESP32 modules or development boards may provide additional features or options for DAC functionality, so it's recommended to refer to the datasheet or specific documentation for your particular ESP32 module or board for more detailed information about DAC pins and their usage.

DAC Pins	ESP Pin
DAC1	D25
DAC2	D26

5. Touch Pins of ESP32

The ESP32 microcontroller features built-in capacitive touch sensing capabilities, allowing it to detect touch or proximity without the need for physical contact with traditional buttons or switches. These touch sensing capabilities are often implemented using specialized touch sensor pins.

The touch sensor pins on the ESP32 are typically labeled T0 to T8, indicating multiple touch input channels. However, the exact number and configuration of touch pins may vary depending on the specific ESP32 module or development board being used.

Touch Pins	ESP32 Pins
Touch0	D4
Touch2	D2
Touch3	D15
Touch4	D13
Touch5	D12
Touch6	D14
Touch7	D27
Touch8	D33
Touch9	D32

6. Transmitter and Receiver Pins of ESP32

The ESP32 microcontroller does not have dedicated "transmitter" and "receiver" pins in the same sense as hardware serial communication pins like those found on traditional microcontrollers. Instead, the ESP32 typically uses its General-Purpose Input/Output (GPIO) pins for various communication protocols, including UART (Universal Asynchronous Receiver Transmitter), SPI (Serial Peripheral Interface) and I2C (Inter-Integrated Circuit).

For UART communication, which is commonly used for serial communication between the ESP32 and external devices, you would typically use two GPIO pins:

UART TX (Transmit) Pin: This pin is used to transmit data from the ESP32 to an external device. It is often labeled as "TX" or "TXD" and connected to the receive pin (RX) of the external device.

UART RX (Receive) Pin: This pin is used to receive data from an external device to the ESP32. It is often labeled as "RX" or "RXD" and connected to the transmit pin (TX) of the external device.

When configuring UART communication on the ESP32, you would specify these GPIO pins as the transmit and receive pins for the UART interface.

For SPI and I2C communication, the ESP32 also uses specific GPIO pins for data transfer, but they are not explicitly labeled as "transmitter" and "receiver" pins. Instead, they are designated as the data input/output pins for the respective communication protocol.

It's essential to refer to the datasheet or specific documentation for the ESP32 module or development board you're using to determine the GPIO pins used for UART, SPI, or I2C communication and their corresponding functions. Additionally, you'll need to configure the communication protocol and GPIO pins appropriately in your firmware code to establish communication with external devices.

Transmitter/Receiver Pins	ESP32 Pins

TX	D22
Rx	D21

Processor Core:

The ESP32 features one or two Xtensa LX6 32-bit microprocessor cores running at clock speeds up to 240 MHz. These cores execute instructions and perform computations for various tasks.

Memory:

ROM: The ESP32 has 448 KB of ROM (Read-Only Memory) for storing boot firmware and other essential code.

SRAM: It includes 520 KB of SRAM (Static Random-Access Memory) for data storage and temporary program execution.

RTC SRAM: An additional 16 KB of SRAM is reserved for the Real-Time Clock (RTC), which maintains time-related data even when the device present in deep sleep mode.

Wireless Connectivity:

Wi-Fi: The ESP32 supports 802.11b/g/n Wi-Fi standards with data rates of up to 150 Mbps, providing wireless network connectivity.

Bluetooth: It is compliant with Bluetooth v4.2 BR/EDR (Basic Rate Enhanced Data Rate) and Bluetooth Low Energy (BLE) specifications, enabling communication with Bluetooth-enabled devices.

Peripheral Interfaces:

GPIOs: The ESP32 offers 34 programmable GPIOs (General-Purpose Input/Output) for digital communication and control, supporting various external devices and sensors.

ADC: Analog-to-Digital Converters provide analog input and output capabilities for interfacing with analog sensors and actuators.

SPI/I2C/I2S/UART: Multiple serial communication interfaces such as SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), I2S (Inter-IC Sound), and UART (Universal Asynchronous Receiver-Transmitter) facilitate communication with external peripherals.

SDIO/eMMC/SD: Host and slave interfaces for SD (Secure Digital), eMMC (Embedded MultiMediaCard), and SDIO (Secure Digital Input Output) communication support external memory storage and data transfer.

Ethernet: An Ethernet MAC (Media Access Control) interface with dedicated DMA (Direct Memory Access) and IEEE 1588 support enables wired network connectivity.

Others: Additional interfaces include touch sensors, PWM (Pulse Width Modulation), motor control, and more, expanding the device's versatility for various applications.

Clocks and Timers:

The ESP32 integrates internal and external oscillators for clock generation, along with timer modules for scheduling tasks, generating precise timing signals, and managing power modes.

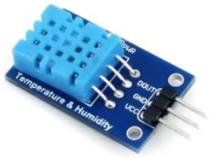
Power Management:

Power management features allow fine-resolution control over power consumption by adjusting clock frequencies, duty cycles, and power modes to optimize energy efficiency.

Security Features:

Security features include secure boot, flash encryption, and cryptographic hardware acceleration for encryption algorithms (AES), hashing (SHA-2), digital signatures (RSA, ECC), and random number generation (RNG), ensuring data integrity and confidentiality.

II. DHT11



The DHT11 Temperature & Humidity Sensor is a flexible device that can accurately and reliably monitor temperature and humidity in a variety of applications. The integrated resistive-type humidity measuring component and NTC temperature measurement component, together with a high-performance 8-bit microprocessor, allow for exact data gathering. One of its most notable characteristics is the calibrated digital signal output, which ensures constant and dependable transfer of temperature and humidity data. The sensor uses an innovative digital signal acquisition approach., providing not only accuracy but also a fast response to changes in environmental conditions. With a focus on long-term stability, the DHT11 excels in maintaining reliable performance over extended periods. Its anti-interference capabilities make it suitable for deployment in environments with potential electromagnetic disturbances. Furthermore, the DHT11 strikes a balance between quality and cost-effectiveness, offering a cost-efficient solution for applications requiring temperature and humidity monitoring. Overall, the DHT11 stands as a reliable choice for users seeking an affordable yet accurate sensor for their temperature and humidity sensing needs.

Technical Specifications

Item	Measurement Range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20-90%RH 0-50 ℃	±5%RH	±2℃	1	4 Pin Single Row

Circuit

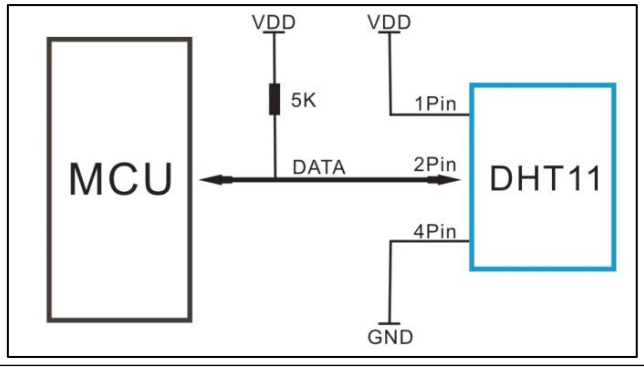


Fig:- Interfacing Circuit of ESP32 With DHT11 Sensor

III. MQ135

The MQ135 Gas Sensor is a versatile device designed to detect a wide range of gases, including carbon dioxide (CO2), carbon monoxide (CO), ammonia (NH3), nitrogen oxides (NOx), sulfur dioxide (SO2), and various volatile organic compounds (VOCs). Its primary function is to measure the concentration of these gas in the surrounding environment and provide a corresponding output signal.

One of the key features of the MQ135 sensor is its sensitivity to a broad spectrum of gases, making it suitable for diverse applications such as air quality monitoring, industrial safety, and environmental monitoring. The sensor operates based on the principle of chemical detection, where the presence of specific gases causes changes in the sensor's conductivity.

The MQ135 sensor consists of a sensitive layer composed of tin dioxide (SnO2) nanoparticles, which undergo changes in resistance when exposed to different gases. This change in resistance is measured and converted into an output voltage signal, which can be further processed and analyzed by a microcontroller or other electronic devices.

The sensor's sensitivity and selectivity to different gases can be adjusted by varying the operating temperature and applying appropriate voltage levels. Additionally, the MQ135 sensor is equipped with a heating element to stabilize its performance and ensure accurate gas detection.

Structure & Configuration

	Parts	Materials
1	Gas sensing layer	SnO_2
2	Electrode	Au
3	Electrode line	Pt
4	Heater coil	Ni-Cr alloy
5	Tubular ceramic	Al_2O_3
6	Anti-explosion network	Stainless steel gauze (SUS316 100-mesh)
7	Clamp ring	Copper plating Ni
8	Resin base	Bakelite
9	Tube Pin	Copper plating Ni

IV. MQ2

The MQ2 Gas Sensor is used module designed to detect various gases in the environment, including but not methane (CH4), carbon monoxide (CO), and alcohol vapours. It operatesMQ2 based on the principle of chemical detection, where the presence of specific gases causes changes in the sensor’s conductivity.

The MQ2 sensor consists of a sensitive layer composed of tin dioxide (SnO2) nanoparticles, which exhibit variations in resistance when exposed to different gases. This change in resistance is proportional to the concentration of the target gas, allowing the sensor to provide a corresponding output signal.

One of the key features of the MQ2 sensor is its versatility in detecting a wide range of gases, making it suitable for applications such as gas leak detection, industrial safety monitoring, and air quality assessment. Its sensitivity and selectivity to different gases can be adjusted by controlling the operating temperature and applying appropriate voltage levels.

The sensitive substance of the MQ-2 gas sensor is SnO2, which has a low conductivity in clean air. When the target flammable gas exists, the sensor's conductivity increases as the gas concentration rises. Please utilize a simple electro circuit to translate the change in conductivity to the corresponding output signal of gas concentration. MQ-2 gas sensor has a high sensitivity to propane and hydrogen, but it can also detect methane and other combustible steam. It is low-cost and suited for a variety of applications.

Technical Specification

Model No.	MQ-2
Sensor Type	Semiconductor
Standard Encapsulation	Bakelite (Black Bakelite)
Detection Gas	Combustible gas and smoke
Concentration	300-10000ppm (Combustible gas)

Circuit

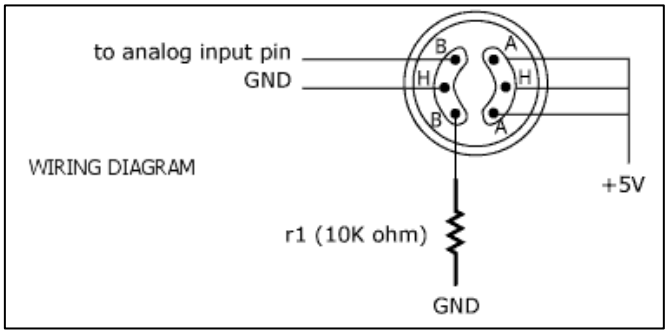


Fig: - H - Pins Out of the two H pins, one is linked to the supply and the other to ground.

A-Pins the A and B pins are interchangeable. These pins will connect to the supply voltage.

B-Pins the A and B pins are interchangeable. One pin will function as an output, while the other will be pulled to ground.

V. MQ7

The MQ7 Gas Sensor is a robust module designed to detect carbon monoxide (CO) gas in the environment. It operates based on the principle of chemical detection, where the presence of CO gas causes changes in the sensor's conductivity.

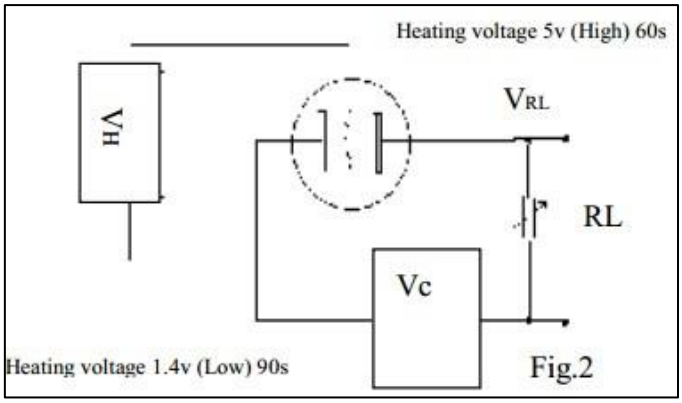
The MQ7 sensor consists of a sensitive layer composed of tin dioxide (SnO2) nanoparticles, which exhibit variations in resistance when exposed to CO gas. This change in resistance is proportional to the concentration of CO gas, allowing the sensor to provide a corresponding output signal.

One of the key features of the MQ7 sensor is its high sensitivity to CO gas, making it suitable for applications such as indoor air quality monitoring, industrial safety, and automotive exhaust detection. It can detect CO concentrations in the range of a few to several hundred parts per million (ppm), depending on the specific model and configuration.

The sensor module typically includes a preheating circuit to stabilize its performance and ensure accurate gas detection. It also features analog and digital output pins for interfacing with microcontrollers or other electronic devices, enabling real-time monitoring and data analysis.

The MQ7 Gas Sensor offers a cost-effective and reliable solution for CO gas detection and monitoring applications, providing valuable insights into indoor air quality and potential health hazards associated with CO exposure. Its ease of use, sensitivity, and wide detection range make it a popular choice for both residential and industrial applications.

Circuit



2. Software Code

Algorithm

Initialization:

Include necessary libraries (WiFi.h, HTTPClient.h, DHT.h).

Define pin connections for sensors and sensor types.

Specify Wi-Fi network credentials (SSID and password) and the Google Apps Script endpoint URL.

Initialize serial communication, DHT sensor, and connect to Wi-Fi network in the setup function.

Main Loop:

Continuously loop through the following steps:

Read temperature and humidity data from the DHT11 sensor.

Read gas sensor data (MQ135, MQ2, MQ7).

Print sensor readings (temperature, humidity, MQ135, MQ2, MQ7) to the serial monitor.

Call the sendData function to send sensor data to the Google Apps Script endpoint.

Wait for 5 seconds before repeating the loop.

sendData Function:

Create an HTTPClient object.

Construct a data string containing sensor readings in the required format.

Send an HTTP POST request to the Google Apps Script endpoint with the data.

If the request is successful (HTTP response code > 0), print the response from the server to the serial monitor.

If there is an error in the request, print an error message to the serial monitor.

connectToWiFi Function:

Attempt to connect the ESP32 device to the specified Wi-Fi network.

Loop for a maximum of 10 attempts, waiting for the device to connect to Wi-Fi.

If the connection is successful, print a confirmation message to the serial monitor. Otherwise, print a failure message.

End of Algorithm

Cloud Interface Algorithm

Initialize Variables:

Initialize the result variable to 'Ok'.

Initialize the sheet_id variable with the ID of the target Google Sheets spreadsheet.

Get the active sheet of the target spreadsheet.

Handle Incoming GET Request:

When a GET request is made to the web app URL:

Capture the event object e.

Log the event object for debugging purposes.

Check Parameter Existence:

Check if the parameter property of the event object is undefined:

If true, set result to 'No Parameters'.

Process Parameters and Update Spreadsheet:

If parameters are present:

Calculate the new row index for appending data by getting the last row and incrementing it by 1.

Initialize an empty array rowData to store data.

Get the current date and time and add them to rowData.

Iterate over each parameter in the event object:

Remove surrounding quotes from the parameter value.

Assign the value to the corresponding index in rowData based on the parameter name.

Construct a new range in the spreadsheet starting from the new row.

Set the values of the range to rowData.

Return Response:

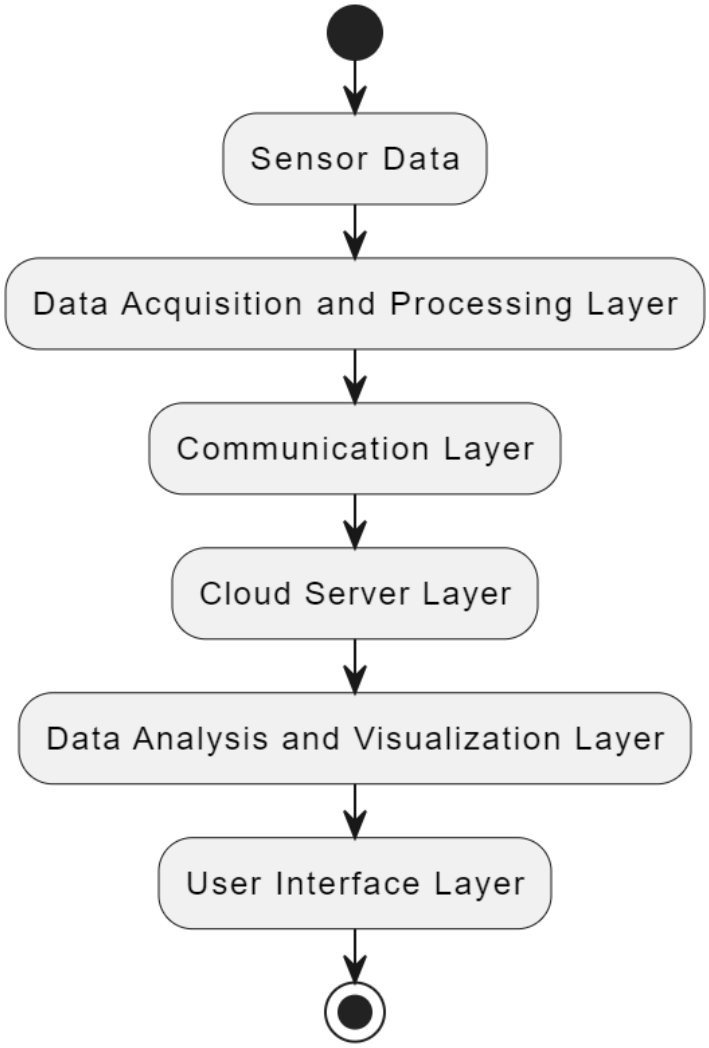
Return a text output containing the value of result.

Utility Function stripQuotes(value):

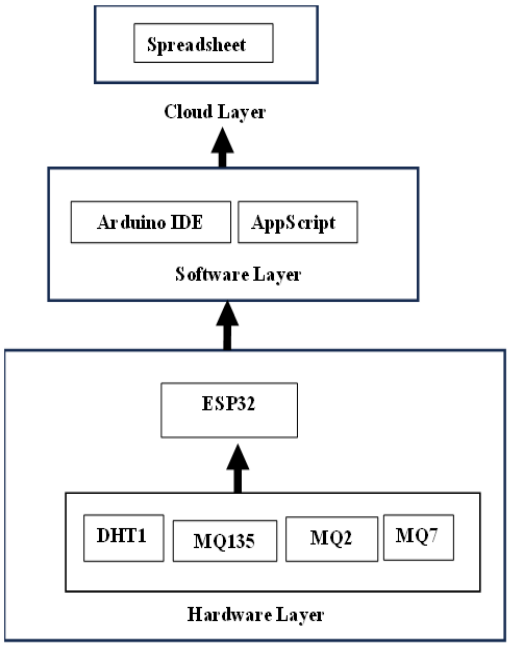
Define the utility function stripQuotes(value) to remove surrounding quotes from a string value.

End of Algorithm

7. Architecture



8. Project Flowchart



9. Layers

User Interface Layer:

The User Interface Layer serves as the gateway for user interaction with the system, providing real-time data display and visualization tools for monitoring environmental parameters**.**

Data Analysis and Visualization Layer:

Responsible for processing and analysing data stored in the cloud server, this layer generates visualizations such as charts and graphs to derive meaningful insights from sensor readings.

Cloud Server Layer:

Cloud Server Layer hosts the Google Apps Script endpoint for handling sensor data. It receives, processes, and stores data in Google Sheets, ensuring centralized data management and accessibility.

Communication Layer:

Facilitating data transmission between IoT devices and the cloud server, the Communication Layer establishes Wi-Fi connections and sends processed sensor data for further processing**.**

Data Acquisition and Processing Layer:

This layer is responsible for reading sensor data from various sensors, processing it, and preparing it for transmission to the cloud server through the Communication Layer.

Sensor Layer:

Comprising specific sensors such as DHT11, MQ135, MQ2, and MQ7, this layer measures temperature, humidity, and gas concentrations, providing raw data for processing.

Hardware Layer:

The Hardware Layer includes the ESP32 microcontroller board, serving as the hardware interface for sensors and communication with the cloud server, enabling seamless integration of IoT devices.

10. Protocol Stack Overview in Industrial Air Pollution Monitoring System

Layer	Protocol / Technology	Description
Physical Layer	Ethernet	A robust wired networking technology for seamless device connection in local area networks (LAN).
	Wi-Fi (IEEE 802.11)	Standardizing wireless local area networking, this protocol relies on radio waves for efficient data transmission.
Data Link Layer	Ethernet (IEEE 802.3)	Defining both physical and data link layer protocols, this technology is fundamental for wired LAN protocols.
	Wi-Fi (IEEE 802.11)	Enabling the transmission of data packets wirelessly, this layer is pivotal for maintaining connectivity in wireless networks.
Network Layer	Internet Protocol (IP)	Serving as the backbone, IP addresses and routes packets across networks with unmatched precision.
	Transmission Control Protocol (TCP)	Ensuring reliable, ordered, and error-checked data delivery, TCP is indispensable for applications across interconnected hosts.
	User Datagram Protocol (UDP)	A lightweight, connectionless protocol ideal for real-time applications, providing rapid data transmission where strict reliability is not the top priority.
Transport Layer	Hypertext Transfer Protocol (HTTP)	Facilitating the exchange of hypermedia documents between web servers and clients, HTTP is the cornerstone of web communication.
	Hypertext Transfer Protocol Secure (HTTPS)	An extension of HTTP, HTTPS enhances security through SSL/TLS, ensuring encrypted communication and secure server identification.
Session Layer	HTTP/HTTPS Sessions	Managing sessions using cookies and tokens, ensuring stateful connections between clients and servers, and enabling a seamless user experience.
Presentation Layer	JSON	A lightweight, human-readable data-interchange format, JSON facilitates easy data exchange between applications.
	XML	Utilized for structured data representation, XML is commonly employed in configuration files and web services.
Application Layer	Google Apps Script	Empowering lightweight application development in the Google Suite, this scripting language is key to creating web applications and automating tasks in various Google services.

Layer	Protocol / Technology	Description
Cloud Computing Layer	Representational State Transfer (REST)	REST is an architectural approach for distributed systems, establishes guidelines for developing web services with well-defined operations and resource representations.

Physical Layer:

Ethernet: In your project, Ethernet may not be directly applicable since you're likely utilizing Wi-Fi (IEEE 802.11) for wireless connectivity. However, if you choose to incorporate wired connections in certain scenarios, Ethernet would be relevant for establishing physical connections between devices within your monitoring system.

Data Link Layer:

Wi-Fi (IEEE 802.11): This layer is crucial for wireless communication between devices in your monitoring system. Wi-Fi enables the transmission of data packets over radio waves, allowing sensors, microcontrollers (such as ESP32), and other components to communicate with different hardware and the cloud server.

Network Layer:

Internet Protocol (IP): IP addressing and routing are essential for transmitting data packets across networks. In your project, IP ensures that data collected from sensors is properly addressed and routed to its destination, such as the cloud server for storage and analysis.

Transmission Control Protocol (TCP) : Depending on your specific requirements, you may use either TCP for communication between devices and the cloud server. TCP provides reliable, ordered, and error-checked data delivery, which is suitable for applications where data integrity is critical. On the other hand, UDP offers a lightweight, connectionless communication option that may be preferred for real-time data transmission.

Transport Layer:

Hypertext Transfer Protocol (HTTP): HTTP or HTTPS protocols are used for communication between the ESP32 microcontroller and the cloud server. HTTP is suitable for transmitting sensor data to the server, while HTTPS ensures secure and encrypted communication, safeguarding sensitive information such as user credentials and sensor readings**.**

Session Layer:

HTTP/HTTPS Sessions: Sessions are managed using cookies and session tokens to maintain stateful connections between the ESP32 microcontroller and the cloud server. This allows for continuous data transmission and ensures that each interaction between the devices is part of a coherent session.

Presentation Layer:

JSON (JavaScript Object Notation): JSON is used for structuring and representing the data exchanged between the ESP32 microcontroller and the cloud server. It provides a lightweight and human-readable format for transmitting sensor readings, allowing for easy parsing and interpretation by both devices**.**

Application Layer:

Google Apps Script: Google Apps Script serves as the application layer in your project, providing a scripting environment for developing lightweight applications within the Google Suite ecosystem. In your case, Google Apps Script is used to create custom scripts that handle data received from the ESP32 microcontroller, store it in Google Sheets, and perform any necessary data analysis or visualization tasks.

Cloud Computing Layer:

Representational State Transfer (REST): RESTful principles are applied in your project for designing the communication interface between the ESP32 microcontroller and the cloud server. This architecture allows for the creation of well-defined operations (HTTP methods) and resource representations (JSON data format), enabling seamless integration and interaction between the devices in your monitoring system.

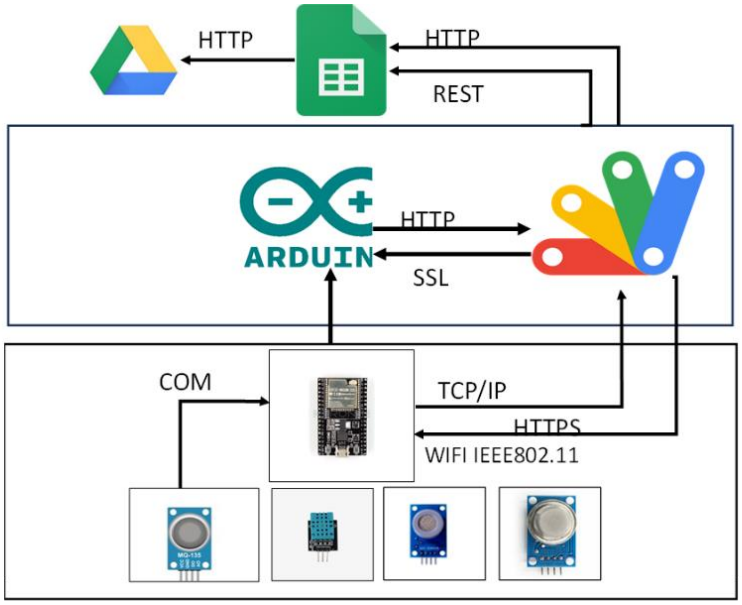


Fig: - Flowchart of Protocols governing Project

11. Data Flow

The data flow within the system begins with the sensors, namely the DHT11, MQ135, MQ2, and MQ7, which are connected to the ESP32 microcontroller. These sensors capture environmental parameters such as temperature, humidity, and various gas concentrations. The ESP32 reads data from these sensors periodically and processes it accordingly. Once the data is collected, the ESP32 establishes a connection to the local Wi-Fi network using the provided credentials. Subsequently, the ESP32 sends the processed sensor data over the Wi-Fi network to a designated Google Apps Script endpoint. This endpoint acts as an intermediary, receiving the data from the ESP32 and handling it accordingly. The Google Apps Script processes the received data and stores it in a Google Sheets spreadsheet hosted on Google Drive. The spreadsheet serves as a centralized location for logging and organizing the collected sensor data. Through this flow of data, the system enables real-time monitoring and analysis of environmental parameters, facilitating informed decision-making and environmental management.

Fig: - illustrate the flow of Data Between different layers

12. Circuit Diagram: -

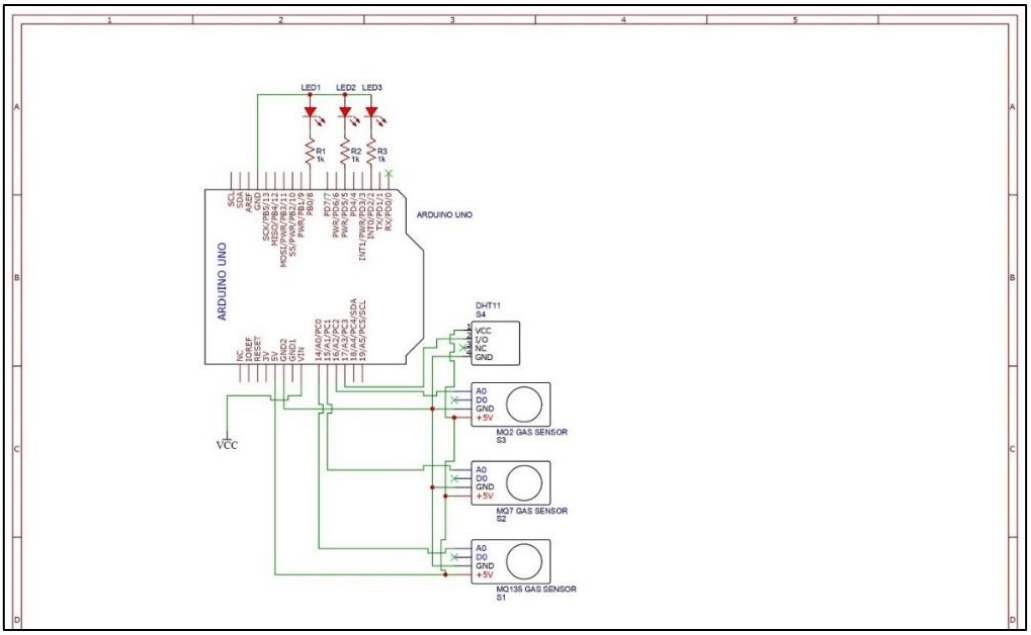


Fig: The above diagram is the illustration of the Implementation on Circuit

13. Conclusion

In conclusion, the project "Industrial Air Pollution Monitoring using Cloud Computing" represents a significant advancement in the field of environmental monitoring and management. By leveraging IoT technology and cloud computing resources, the project offers a comprehensive solution for real-time monitoring of air quality in industrial settings. The integration of sensors such as DHT11, MQ135, MQ2, and MQ7 with the ESP32 microcontroller enables accurate and reliable measurement of key parameters including temperature, humidity, CO2, CO, and CH4 levels.

Through the use of Wi-Fi connectivity, the ESP32 facilitates communication between the sensors and cloud-based platforms, allowing for the transmission of sensor data to a Google Sheets spreadsheet hosted on Google Drive. This data logging mechanism enables continuous monitoring of air pollution levels over time, providing valuable insights for environmental analysis and decision-making.

The project addresses the limitations of traditional air pollution monitoring systems, such as GSM modules and WSN networks, by offering a cost-effective, scalable, and efficient solution. By utilizing cloud Architecture for data storage and data analysis, the project ensures centralized data management, accessibility, and flexibility.

Furthermore, the project demonstrates the potential of IoT technology in addressing Industry challenges and promoting sustainable development practices. By empowering industries with real-time air quality monitoring capabilities, the project contributes to the monitoring of gases released from the industry.

In summary, "Industrial Air Pollution Monitoring using Cloud Computing" presents a robust and innovative approach to industrial air Pollution monitoring, with implications for various industries and sectors. The project highlights the importance of leveraging emerging technologies to address pressing environmental issues and underscores the role of IoT and cloud computing in shaping a more sustainable future.

14. Result

The results of the project “Cloud-Enabled Industrial Pollution Monitoring & Data Management Systems" demonstrate the successful implementation of a comprehensive air quality monitoring system. Key findings and outcomes of the project include:

Real-time Monitoring: The system enables real-time monitoring of temperature, humidity, CO2, CO, and CH4 levels in industrial environments. Data from sensors such as DHT11, MQ135, MQ2, and MQ7 are collected continuously and transmitted to a cloud-based platform for analysis.

Data Logging and Storage: Sensor data is logged and stored in a Google Sheets spreadsheet hosted on Google Drive. This data logging mechanism ensures the preservation of historical data for trend analysis and decision-making.

	A	B	C	D	E	F	G
1	Date	Time	Temperatur	Humidity	MQ135	MQ7	MQ2
2	25-02-2024	13:45:15	nan	55	0	0	279
3	25-02-2024	13:45:23	nan	55	0	0	0
4	25-02-2024	13:45:38	nan	55	0	0	0
5	25-02-2024	13:45:47	nan	55	0	0	0
6	25-02-2024	13:45:56	nan	55	0	0	0
7	25-02-2024	13:46:04	nan	55	0	0	80
8	25-02-2024	13:47:43	27	55	0	6	208
9	25-02-2024	13:47:52	27	55	0	128	508
10	25-02-2024	13:48:01	27	55	0	206	617
11	25-02-2024	13:48:10	27	55	0	205	654
12	25-02-2024	13:48:18	27	55	0	224	656
13	25-02-2024	13:48:27	27	55	0	239	661
14	25-02-2024	13:48:36	27	55	0	238	687
15	25-02-2024	13:48:45	27	55	5	240	659
16	25-02-2024	13:48:53	27	55	0	214	654
17	25-02-2024	13:49:02	27	55	0	208	660
18	25-02-2024	13:49:22	27	55	13	225	624
19	25-02-2024	13:49:31	27	55	12	239	643
20	25-02-2024	13:49:40	27.1	55	0	216	640
21	25-02-2024	13:49:48	27	55	8	248	681
22	25-02-2024	13:49:59	27.1	55	0	212	671
23	25-02-2024	13:50:09	27.1		0	240	631
24	25-02-2024	13:50:17	27.1		0	210	630
25	25-02-2024	13:50:26	27.1		13	217	619

Fig: - The Data logging in the Google Spreadsheet

Accuracy and Reliability: The project demonstrates the accuracy and reliability of sensor readings, ensuring precise measurement of air pollution parameters. Calibration and testing procedures validate the performance of the sensors and the overall monitoring system.

```
���Temperature: 29.00
Humidity: 37.00
MQ135 Value: 427
MQ2 Value: 1535
MQ7 Value: 913
HTTP Response Code: 302
<HTML>
<HEAD>
<TITLE>Moved Temporarily</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
<H1>Moved Temporarily</H1>
The document has moved <A HREF="https://script.googleusercontent.com/macros/echo?user_content_key=QtSWEV8r_0uGCHdBM1itcGH6
```

Fig: - The script is the Serial Monitor data which is fetch by the Arduino IDE from

Connectivity and Accessibility: Wi-Fi connectivity provided by the ESP32 microcontroller enables seamless communication between the sensors and the cloud server. This connectivity ensures remote access to sensor data from any location with internet access.

```
HTTP Response Code: 302
<HTML>
<HEAD>
<TITLE>Moved Temporarily</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
<H1>Moved Temporarily</H1>
The document has moved <A HREF="https://script.googleusercontent.com/macros/echo?user_content_key=QtSWEV8r_0uGCHdBM1itcGH6XJI2a3QQATU
</BODY>
</HTML>
```

Fig: - The Above serial monitor script connected to Cloud and sending data to the Spreadsheet via HTTPS

Data Analysis and Visualization: Sensor data stored in the Google Sheets spreadsheet can be analysed using various tools and techniques. Data visualization methods such as charts, graphs, and dashboards provide insights into air quality trends and patterns over time.

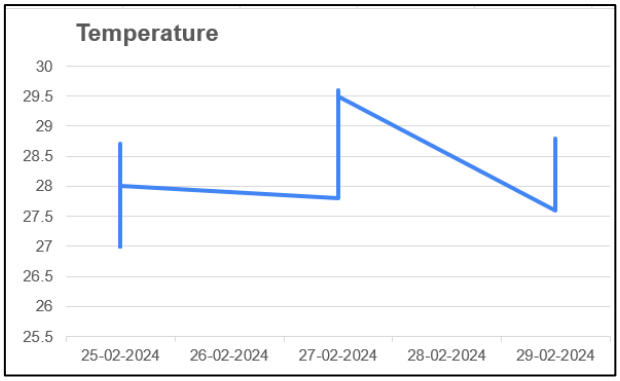


Fig: - The Data Visualization of Temperature collected by DHT11 Sensor

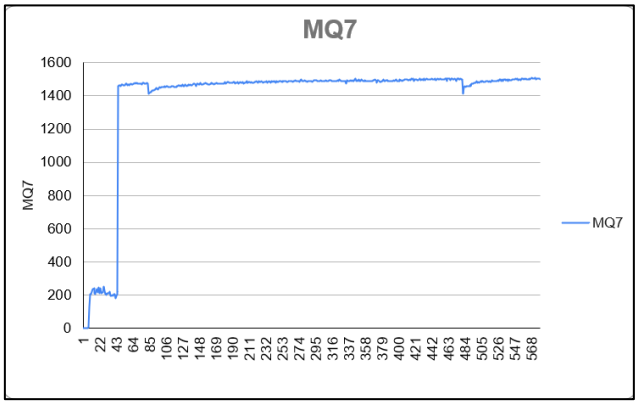


Fig: - Data Visualization of MQ7 Sensor Data

Environmental Impact: By providing real-time monitoring of air pollution levels, the project contributes to environmental protection and public health. Early detection of pollutant levels allows for timely intervention and mitigation measures to minimize environmental impact.

References:

1. Parikh, H., R. Desai, D., & Iyer, S. (2020). IoT Based Indoor Air Quality Monitoring System. In 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (pp. 1-6). IEEE.
2. Singh, S., Gupta, R., & Choudhary, A. (2019). Carbon Monoxide Monitoring System Using IoT. In 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS) (pp. 1-5). IEEE.

3. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys & Tutorials, 17(4), 2347-2376.
4. Kim, H., & Eom, H. (2019). A Study on Environment Monitoring Using Wireless Sensor Network based on ESP32. In 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC) (pp. 1-5). IEEE.
5. Bhatia, P., & Singh, V. (2017). Cloud based IoT for Environmental Monitoring. In 2017 2nd International Conference on Computing, Communication and Automation (ICCCA) (pp. 1-5). IEEE.