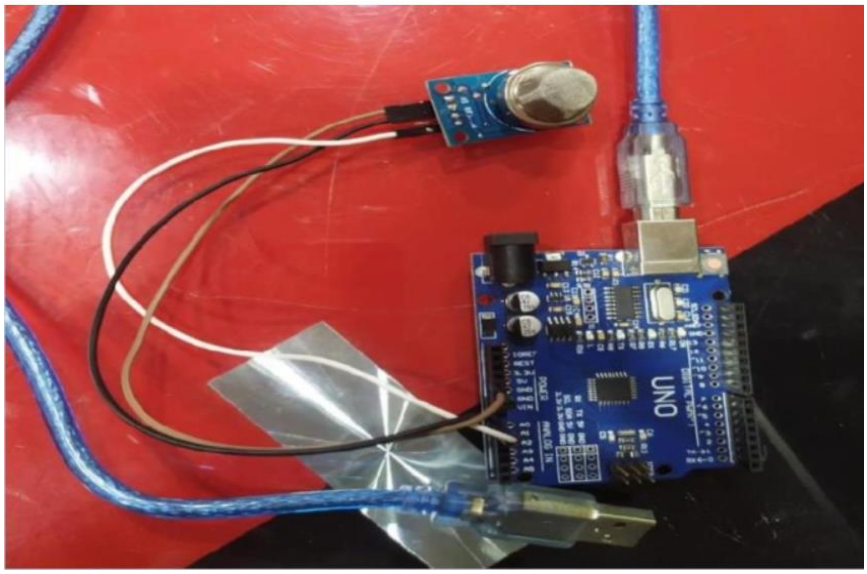
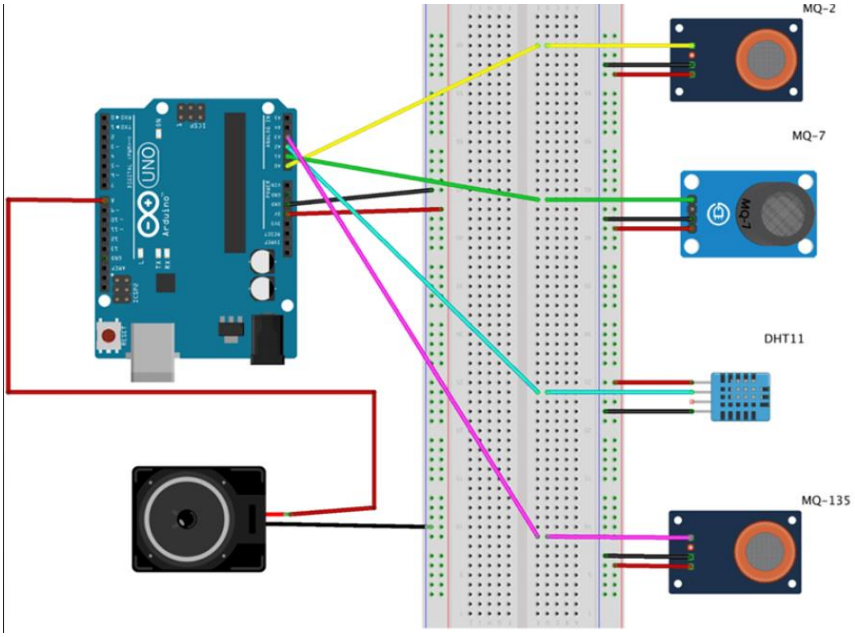
PROJECT DOCMENTATION AND SUBMISSION

OBJECTIVE:

This project aims to develop a comprehensive solution for providing real-time air quality information to the public. Leveraging IoT technology and Python integration, the project encompasses the design and implementation of an air quality monitoring system with sensors for key pollutants. The data collected is seamlessly integrated into a user-friendly platform, offering an intuitive interface for the public to access upto- date air quality information. Through a design thinking approach, the project ensures user empathy, iterative development, and community engagement. The platform not only serves as a source of information but also aims to raise awareness about air quality issues, empowering individuals to make informed decisions for their well-being and the environment.The project involves setting up IOT devices to measure air quality parameters and make the data publicly available for raising awareness about air quality and its impact on public health. The objective is to create a platform that provides real-time air quality information to the public. This project includes defining objectives, designing the IOT monitoring system, developing the data sharing platform and integrating them using IOT technology and python.The primary objectives of this project are to enhance public awareness regarding air quality and its consequential effects on public health, and second, to establish a comprehensive air quality monitoring system. The system's core objectives involve the real-time collection of air quality data, encompassing critical parameters like particulate matter (PM2.5 and PM10), various gases (CO, NO2, SO2, O3), and environmental conditions such as temperature and humidity. The acquired data will be made readily accessible to the public through an intuitive and user-friendly platform, which will employ data visualization techniques, including charts, graphs, and maps, to facilitate comprehension.

IOT device Setup:





PLATFORM DEVELOPMENT:

ARDUINO IDE:

Arduino is a popular platform for building electronic projects, and you can use it to interface with gas sensors like the MQ-6 sensor to collect data on the concentration of a specific gas, such as LPG (liquefied petroleum gas) or butane. Here are the general steps to use Arduino software to get data from an MQ-6 sensor:

1. *Gather Components*:
   - Arduino board (e.g., Arduino Uno or Arduino Nano).
   - MQ-6 gas sensor.
   - Breadboard and jumper wires.
   - A power supply for the sensor (MQ-6 sensors usually require 5V).

2. *Connect the Sensor*:
   - Connect the VCC (5V) and GND pins of the MQ-6 sensor to the corresponding 5V and GND pins on the Arduino.
   - Connect the analog output (AO) pin of the sensor to one of the analog input pins on the Arduino, such as A0.

3. *Install Arduino Software*:
   - Download and install the Arduino IDE (Integrated Development Environment) on your computer if you haven't already.

4. *Write Arduino Code*:
   - Open the Arduino IDE and write code to interface with the MQ-6 sensor. You can use the analogRead() function to read the sensor's output voltage from the analog pin connected to the sensor.

- Convert the analog value to a gas concentration using a calibration curve specific to your sensor. The datasheet for the MQ-6 sensor should provide information on how to calibrate it.

5. *Upload Code to Arduino*:
   - Connect your Arduino board to your computer using a USB cable.
   - Select the appropriate board and COM port from the Arduino IDE's "Tools" menu.
   - Click the "Upload" button to upload the code to the Arduino.

6. *Monitor Data*:
   - Open the Arduino IDE's serial monitor to see the gas concentration readings printed in real-time.

MQTT

MQTT stands for Message Queuing Telemetry Transport. It is an extremely simple and lightweight messaging protocol designed for limited devices and networks with high latency, low bandwidth or unreliable networks. Its design principles are designed to reduce the network bandwidth and resource requirements of devices and ensure security of supply.
With MQ Telemetry Transport, resource-constrained IoT devices can send or publish information on a specific topic to a server that acts as an MQTT message broker.
The MQTT broker is the center of every Publish / Subscribe protocol. Depending on the implementation, a broker can manage up to thousands of simultaneously connected MQTT clients. The broker is responsible for receiving all messages, filtering the messages, determining who subscribed to each message and sending the message to those subscribed clients. The Broker also holds the sessions of all persistent clients, including subscriptions and missed messages. Another task of the Broker is the authentication and authorization of clients

DHT11 Temperature sensor:

The DHT11 Temperature sensor is one of the low cost and small-sized sensors of its type. It is lab calibrated, stable and its signal output is digital.It is highly reliable when it comes to temperature & humidity sensing technology. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component and can connect to a high-performance 8-bit microcontroller. It’s single-wire serial interface and 4-pin single row pin package makes system integration easy .The sensor converts the resistance measurement to humidity on the IC mounted to the back of the unit and transmits the readings directly to the Arduino.components that will be needed to connect the temperature sensor with tinkercad are TMP36, Arduino Uno, and Mini Breadboard. The 3 pin must be connected to the data (7), ground (GND), and voltage (5V).

Technical specifications:

Humidity Measurement Range: 20-90%RH
Temperature Measurement Range: 0-50℃
Humidity Accuracy: ±5%RH
Temperature Accuracy: ±2℃

The relative humidity is measured by the electrical resistance between two electrodes. The humidity sensing component of the DHT11 is a moisture holding substrate with the electrodes applied to the surface. The ions are released by the substrate as water vapor is absorbed by it,

which in turn increases the conductivity between the electrodes. The change in resistance between the two electrodes is proportional to the relative humidity.

The sensor converts the resistance measurement to humidity on the IC mounted to the back of the unit and transmits the readings directly to the Arduino. The temperature readings from the DHT11 come from a surface mounted NTC temperature sensor built into the unit. The DHT11 uses one signal wire to transmit sensor readings to the Arduino digitally. The power comes from separate 5V and ground wires

Setting up Arduino with DHT11:

DHT11 pin-1 is connected to 5v power supply pin on Arduino Uno
DHT11 pin-2 is connected to digital pin-A0 on Arduino Uno
DHT11 pin-3 is NC
DHT11 pin-4 is connected to GND (Ground) pin on Arduino Uno

CODE AND IMPLEMENTATION:

ARDUINO CODED FEEDED INSIDE THE ARDUINO UNO

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// Replace with your WiFi credentials
const char* ssid = "wifi";
const char* password = "123456789";

// Replace with your MQTT broker IP or hostname
const char* mqtt_server = "broker.emqx.io";

// Replace with your MQTT credentials if required
const char* mqtt_user = "";
const char* mqtt_password = "";

WiFiClient espClient;
PubSubClient client(espClient);

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

```
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void setup() {
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);

  // Replace with the actual device ID
  String device_id = " Device_1234";
  float temperature = 25.5;
  float humidity = 48.3;
  String pollution_level = "Moderate";
  float particulate_matter = 15.2;

  String json_data = "{";
  json_data += "\"temperature\":" + String(temperature, 2) + ",";
  json_data += "\"humidity\":" + String(humidity, 2) + ",";
  json_data += "\"pollution_level\":\"" + pollution_level + "\",";
  json_data += "\"particulate_matter\":" + String(particulate_matter, 2);
  json_data += "}";

  char char_data[json_data.length() + 1];
  json_data.toCharArray(char_data, json_data.length() + 1);


  String mqtt_topic = "airquality/" + device_id;

  if (client.connect(device_id.c_str(), mqtt_user, mqtt_password)) {
    Serial.println("Connected to MQTT broker");
    client.publish(mqtt_topic.c_str(), char_data);
    Serial.println("Published data to MQTT topic");
  }
}
```

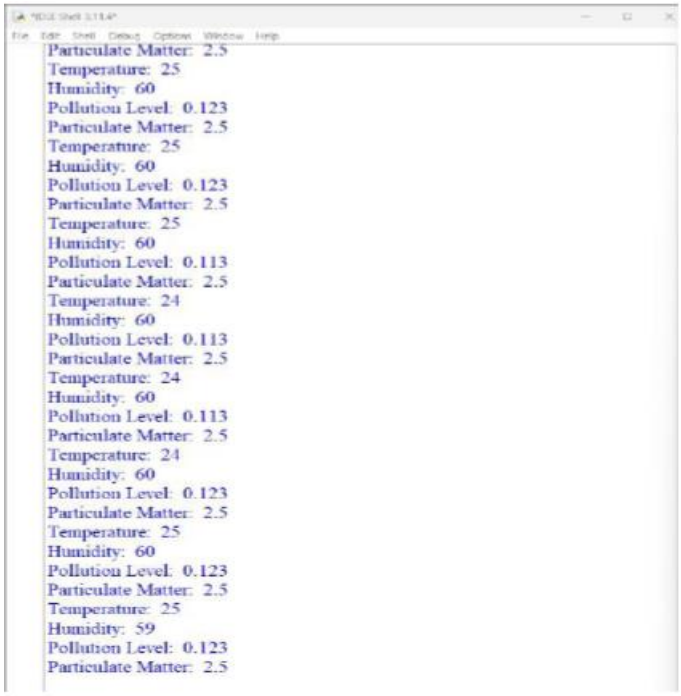
PYTHON CODE TO FETCH THE DATA FROM MQTT SERVER

```
import paho.mqtt.client as mqttClient
from threading import Thread
import json
import mysql.connector
import datetime

class Mqtt:
```

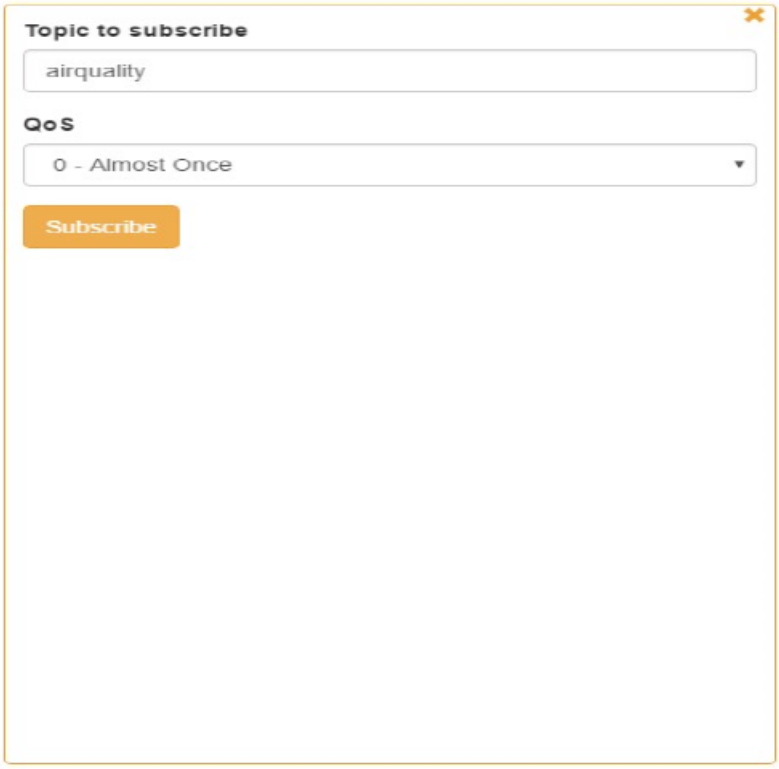
```
    def __init__(self):
        self.db = mysql.connector.connect(
            host="localhost",
            user="root",
            password="",  # Replace with your MySQL password
            database="demo")

        mqttclient = mqttClient.Client("52244535477668454")
        mqttclient.on_connect = self.on_connect
        mqttclient.on_message = self.on_message

        # Set MQTT username and password if required
        mqttclient.username_pw_set(username="", password="")

        mqttstatus = mqttclient.connect("broker.emqx.io", 1883, 60)
        mqttclient.subscribe("airquality", 2)
        mqttclient.loop_forever()

    def upload(self, msg):
        mqtt_msg = str(msg.payload.decode("utf-8"))  # Convert payload to string
        mqtt_msg = mqtt_msg.replace("'", "\"")
        try:
            data = json.loads(mqtt_msg)

            temperature = data.get('temperature')
            humidity = data.get('humidity')
            pollution_level = data.get('pollution_level')
            particulate_matter = data.get('particulate_matter')

            # Print the received data
            print("Temperature: ", temperature)
            print("Humidity: ", humidity)
            print("Pollution Level: ", pollution_level)
            print("Particulate Matter: ", particulate_matter)

            # Insert the data into the database
            cursor = self.db.cursor()
            insert_query = "INSERT INTO sensor_data (temperature, humidity, pollution_level,
particulate_matter, timestamp) VALUES (%s, %s, %s, %s, NOW())"
            data_to_insert = (temperature, humidity, pollution_level, particulate_matter)
            cursor.execute(insert_query, data_to_insert)
            self.db.commit()
            cursor.close()

        except json.JSONDecodeError as e:
            print("Error decoding JSON:", e)

    def on_connect(self, mqttclient, userdata, flags, rc):
        if rc == 0:
```

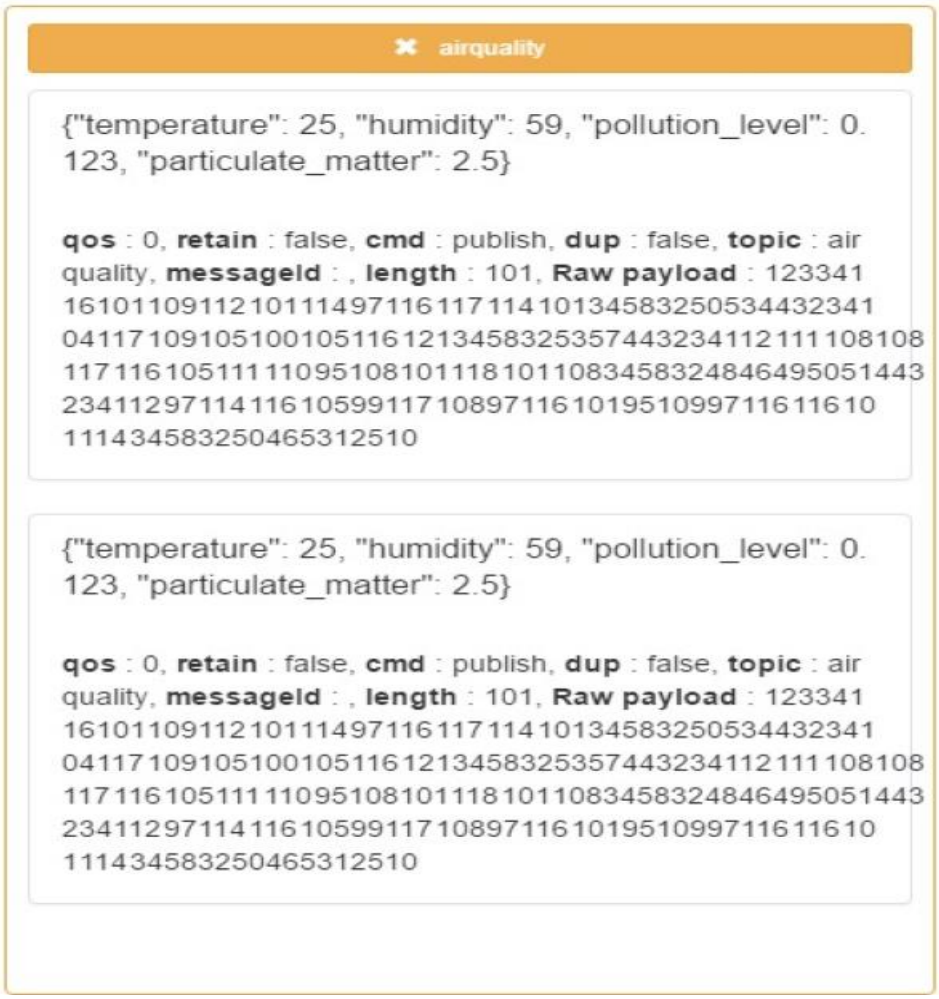
```
            print("Connected!")
        else:
            print("Connection failed")

    def on_message(self, mqttclient, userdata, msg):
        Thread(target=self.upload, args=(msg,)).start()

if __name__ == '__main__':
    Mqtt()
```

Output from MQTT:

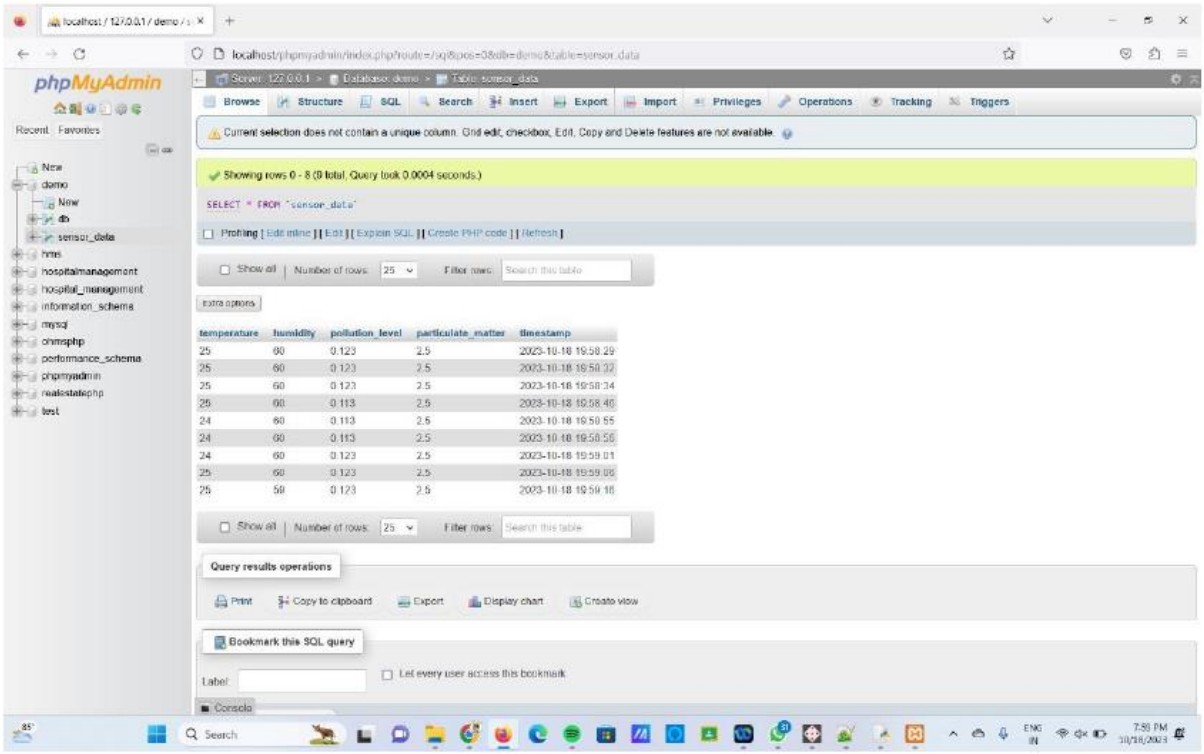


MQTT


Topic to subscribe
airquality
QoS
0 - Almost Once
Subscribe



DATABASE CONNECTION:



WAMP:

WampServer, short for Windows, Apache, MySQL, and PHP, is a popular open-source software stackprimarily designed for Windows operating systems. It provides a convenient and straightforward wayfor web developers to create a local web development environment on their Windows-based computers. This environment enables developers to build, test, and debug web applications without the need for a live web server, making it an essential tool in the web development process.

The core components of WampServer include Apache, the web server that handles HTTP requests; MySQL, a relational database management system for data storage; and PHP, a server-side scriptinglanguage used to create dynamic web applications. By bundling these components, WampServer simplifies the installation and configuration process, allowing developers to set up their developmentenvironment quickly.

WampServer offers several key advantages, including the ability to switch between different versions of PHP to match project requirements, an easy-to-use user interface for configuring server settings, and the ability to run local web applications with ease. Whether you're a beginner looking to learn webdevelopment or an experienced developer working on new projects, WampServer can significantly streamline your workflow by providing a complete, local web server environment on your Windows machine.

PHPMYADMIN:

phpMyAdmin is a versatile and widely-used tool for managing MySQL databases through a user- friendly web-based interface. It offers a wealth of features that streamline database administration anddevelopment tasks. With phpMyAdmin, users can perform a wide range of actions, from creating andaltering database structures to executing SQL queries and managing user privileges. This makes it an invaluable resource for web developers, database administrators, and anyone working with MySQL databases.

One of phpMyAdmin's standout features is its ability to simplify database management tasks. Userscan effortlessly create new databases, tables, and fields, and they can efficiently import and export data in various formats. The SQL query editor allows for custom data retrieval and manipulation, providing fine-grained control over the database's content. Furthermore, the tool's user managementcapabilities enable administrators to set access permissions and secure their databases.

Another advantage of phpMyAdmin is its database design functionality. Users can visually design database structures, define relationships between tables, and set data types and constraints. This makesthe tool especially useful for those who need to plan and implement complex database schemas.
Additionally, phpMyAdmin's support for multiple languages and its extensive community of userscontribute to its widespread adoption and accessibility across different regions and cultures.

CODE:

```
<!DOCTYPE html>
<html>
<head>
    <title>Air Quality Monitoring</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Real-Time Air Quality Monitoring</h1>
        <div id="data-container">
            <!-- Real-time data will be displayed here -->
        </div>
    </div>
    <script src="script.js"></script>
</body>
```

```
<style>
body {
    font-family: Arial, sans-serif;
}

.container {
    text-align: center;
    margin: 20px;
}

#data-container {
    border: 1px solid #ccc;
    padding: 20px;
}
</style>
</html>
```

Script.js

```
function fetchDataAndDisplay() {
    fetch("fetch_data.php")
        .then(response => response.json())
        .then(data => {
            const dataContainer = document.getElementById("data-container");

            if (data.error) {
                dataContainer.innerHTML = `<p>${data.error}</p>`;
            } else {
                const timestamp = new Date(data.timestamp).toLocaleString();

                const newData = `
                    <h2>Real-Time Air Quality Data</h2>
```

```
                <p>Timestamp: ${timestamp}</p>
                <p>Temperature: ${data.temperature} °C</p>
                <p>Humidity: ${data.humidity} %</p>
                <p>Pollution Level: ${data.pollution_level}</p>
                <p>Particulate Matter: ${data.particulate_matter} µg/m³</p>
            `;

            dataContainer.innerHTML = newData;
        }
    })
    .catch(error => {
        console.error("Error fetching data: " + error);
    });
}

// Fetch and display data on page load and every 5 seconds
fetchDataAndDisplay();
setInterval(fetchDataAndDisplay, 5000);
```

fetch_data.php

```
<?php
$db = new mysqli("localhost", "root", "", "demo");

if ($db->connect_error) {
    die("Connection failed: " . $db->connect_error);
}

$sql = "SELECT * FROM sensor_data ORDER BY timestamp DESC LIMIT 1";
$result = $db->query($sql);

if ($result->num_rows > 0) {
```

```
    $row = $result->fetch_assoc();
    $data = [
        "temperature" => $row["temperature"],
        "humidity" => $row["humidity"],
        "pollution_level" => $row["pollution_level"],
        "particulate_matter" => $row["particulate_matter"],
        "timestamp" => $row["timestamp"],
    ];
    echo json_encode($data);
} else {
    echo json_encode(["error" => "No data available"]);
}

$db->close();
?>
```

EXPLANATION

Database connection:

- This section establishes a connection to a MySQL database. It specifies the database server (`localhost`), the username (`root`), and an empty password (in this example, no password). Thedatabase name is set to `"airquality"`.

- It uses the `mysqli` library to create a database connection object (`$conn`).

1. Connection Error Handling:

   - The `if ($conn->connect_error)` condition checks if there was an error in establishing the databaseconnection. If an error occurs, the script terminates and displays an error message with the reason for the connection failure.

2. SQL Query:

   - The SQL query selects specific columns (temperature, humidity, pollution_level, and particulate_matter) from a table named "air." This table is assumed to be within the specified database.

3. Query Execution and Result Handling:

   - The `query` method is used to execute the SQL query. The result is stored in the `$result` variable.

4. Display Data in HTML Table:

   - If there are rows in the result set (i.e., if `$result->num_rows` is greater than 0), it generates anHTML table with column headers (Temperature, Humidity, Pollution Level, Particulate Matter).

   - Inside a loop, it fetches each row of data using `$result->fetch_assoc()`, and for each row, itgenerates an HTML table row (`<tr>`) with the corresponding data in the table cells (`<td>`).

- The loop iterates through all the rows in the result set.

5. No Results Message:

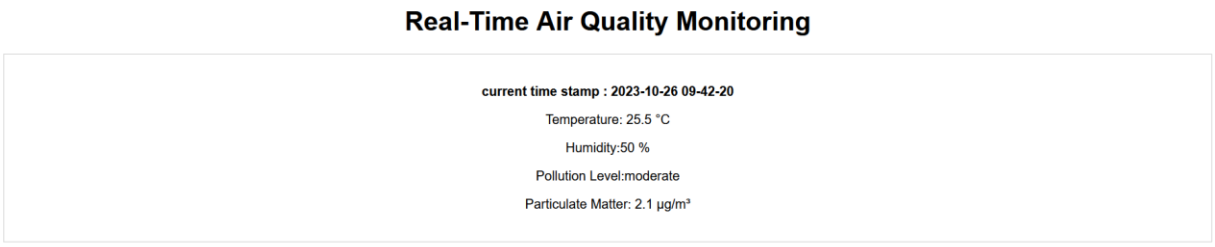
   - If there are no results (i.e., no rows in the result set), it displays a message indicating "0 results."

6. Database Connection Closure:

   - The `$conn->close()` method is used to close the database connection when the data retrieval anddisplay are complete.

   This PHP code connects to a MySQL database, retrieves data from a specific table, and formats the data into an HTML table for display on a web page. The data displayed in the HTML table depends onwhat's stored in the "air" table of the "airquality" database.

Output :



PUBLIC AWARENESS RAISED BY AIR QUALITY MONITORING SYSTEM AND ITS HEALTH IMPACT

Immediate Access to Data:

Real-time monitoring systems continuously collect data on air quality parameters like particulate matter (PM2.5 and PM10), ozone (O3), nitrogen dioxide (NO2), sulfur dioxide (SO2), carbon monoxide (CO), and volatile organic compounds (VOCs). This data is then made available to the public via various platforms such as websites, mobile apps, or even

public displays. Having immediate access to this data allows the public to see the current state of their local air quality.

Transparency:

By making air quality data readily available, these systems promote transparency and accountability among government agencies and industries responsible for air pollution. The public can hold these entities accountable for their environmental impact and demand improvements when air quality is subpar.

Health Impacts Information:

Real-time monitoring systems often provide information on the health impacts of the current air quality. For example, they can warn the public about increased risks of respiratory problems, heart disease, and other health issues associated with poor air quality. This information can educate individuals about the direct consequences of air pollution on their health.

Color-Coded Index:

Many monitoring systems use color-coded indices (e.g., the Air Quality Index, or AQI) to simplify the understanding of air quality. These indices categorize air quality levels into different color bands, ranging from "Good" to "Hazardous." The use of visual cues makes it easier for the public to quickly assess the current situation.

Alerts and Notifications:

Real-time monitoring systems can send alerts and notifications to the public when air quality reaches certain levels. This proactive approach helps individuals take precautionary measures, such as staying indoors, reducing outdoor activities, or wearing masks, during times of poor air quality.

Trend Analysis:

These systems often provide historical data and trend analysis, allowing the public to understand the seasonal and long-term variations in air quality. This data can highlight patterns and trends, encouraging discussions and policy actions.

Community Engagement:

Air quality monitoring can engage communities and local organizations in environmental issues. Communities can use the data to advocate for cleaner air and participate in initiatives to reduce pollution sources.

Policy Advocacy: Armed with real-time air quality data, concerned citizens and environmental organizations can advocate for stronger environmental regulations and policies. This can lead to more stringent emission standards, cleaner energy sources, and sustainable transportation options.

Behavioral Change:

Public access to real-time air quality data can prompt individuals to make personal lifestyle changes, such as reducing vehicle emissions, minimizing wood-burning stoves, and choosing cleaner energy sources.

Educational Tools: Air quality monitoring systems can serve as educational tools in schools, informing the younger generation about the importance of clean air and its impact on public health, which can lead to greater awareness and advocacy in the future.

Immediate Health Concerns:

Real-time air quality monitoring provides the public with information about the current state of the air they are breathing. This information is critical for individuals, especially those with respiratory conditions like asthma or cardiovascular issues, as they can make informed decisions to protect themselves when air quality is poor.

Respiratory Health:

Monitoring systems allow the public to be aware of elevated levels of particulate matter (PM2.5 and PM10), ozone (O3), and other pollutants known to exacerbate respiratory problems. When air quality is poor, individuals can take precautions, such as wearing masks or limiting outdoor activities, to reduce their exposure and potential respiratory distress.

Cardiovascular Health:

Real-time data informs the public about the cardiovascular risks associated with air pollution. People can take precautions and adapt their daily routines during periods of high pollution to mitigate the risk of heart attacks and strokes.

Preventative Measures:

Public awareness of the health impacts of air pollution encourages individuals to take preventive measures to reduce their risk, such as keeping indoor air clean, using air purifiers, and seeking medical advice when necessary.

CONCLUSION:

The development and implementation of a real-time air quality monitoring system, as outlined in this project, have yielded significant benefits for public awareness and health. By leveraging IoT technology and Python integration, this system has provided immediate access to vital air quality data, fostering transparency, accountability, and community engagement. It demonstrates the positive impact of technology when harnessed for the betterment of public health and the environment. As we move forward, the lessons learned from this project can serve as an inspiration for similar initiatives aimed at creating healthier and more informed communities.