Air Quality Management

Phase-5: Project Documentation & Submission

Problem Definition:

Air pollution is one of the biggest threats to the present-day environment. Everyone is being affected by air pollution day by day including humans, animals, crops, cities, forests and aquatic ecosystems. Besides that, it should be controlled at a certain level to prevent the increasing rate of global warming. This project aims to design an IOT-based air quality monitoring system using the internet from anywhere using a computer or mobile to monitor the air quality of the surroundings and environment. There are various methods and instruments available for the measurement and monitoring quality of air. The IoT-based air quality monitoring system would not only help us to monitor the air quality but also be able to send alert signals whenever the air quality deteriorates and goes down beyond a certain level

Design Thinking:

1. **Project Objective:**

   Implement a real-time monitoring system to continuously collect data on key air quality parameters. Enables quick detection of changes in air quality, allowing for prompt response to emerging issues. Measure and analyse critical air quality parameters such as particulate matter (PM2.5, PM10), carbon dioxide (CO2), ozone (O3), nitrogen dioxide (NO2), sulphur dioxide (SO2), temperature, and humidity. Provides a comprehensive understanding of the air quality profile, helping to identify sources of pollution and potential health risks. These objectives collectively contribute to the development and implementation of a comprehensive IOT-based air quality management system that addresses environmental concerns, protects public health, and supports sustainable development.

2. **IOT Sensor Design:**

   Designing IOT sensors for air quality monitoring involves careful consideration of the specific parameters you want to measure, the environmental conditions, power consumption constraints, and communication requirements. Below are key aspects to consider in the design of IOT sensors for air quality monitoring Choose sensors based on the air quality parameters of interest (e.g., PM2.5, CO2, NO2).
   Consider sensor accuracy, sensitivity, and response time.
   Evaluate the suitability of sensors for outdoor or indoor deployment. Decide on the power source, considering factors such as battery life and maintenance requirements.

Explore energy-efficient options, and assess the feasibility of renewable energy sources (solar, wind) where applicable.

3. **Real-Time Transit information platform:**

   Select communication protocols based on the application and network infrastructure (e.g., MQTT, CoAP, LoRa, NB-IoT).Consider the range and data throughput

requirements. Determine whether data processing will be done locally (edge computing) or in the cloud. Consider the storage capacity and the ability to handle real-time Ensure compatibility with the selected IOT platforms for seamless integration. Implement standard data formats and communication protocols for interoperability. analytics Implement a calibration mechanism to ensure sensor accuracy over time. Include self-diagnostic features to identify and compensate for sensor drift**.**

4. **Integration Approach:**
   Determine whether data processing will be done at the edge (on the sensor or gateway) or in the cloud. Consider the advantages of edge computing for real-time processing and cloud computing for in-depth analytics. If using edge computing, deploy IOT gateways to collect and pre-process data from sensors.
   Ensure gateways are compatible with both the sensors and the communication infrastructure. Choose a cloud platform for data storage, analysis, and visualization (e.g., AWS, Azure, Google Cloud).Integrate IOT devices with cloud services using platform-specific APIs or protocols.

Overview of the innovations:

Create a cost-effective, IOT-based air quality monitoring station that can be easily deployed in various locations for real-time air quality data collection

IOT Sensors:

Particulate Matter (PM) Sensor: Measures PM2.5 and PM10 concentrations.

Gas Sensors: Detect pollutants like NO2, SO2, CO, and others.

Microcontroller:

Use a microcontroller (e.g., Arduino, Raspberry Pi) to interface with sensors, process data, and transmit it to the cloud.

Communication Module:

Incorporate a communication module (Wi-Fi, Lora, or cellular) to send data to a centralized server or cloud platform.

Cloud Platform:

Utilize a cloud platform (such as AWS, Azure, or Google Cloud) to collect and store air quality data.

Dashboard:

Create a simple web or mobile dashboard that displays real-time air quality data collected from the monitoring stations.

Innovation Steps:

Sensor Deployment:

Deploy the smart air quality monitoring station in strategic locations, such as urban centre, parks, or industrial areas using PM2.5 etc.

Real-time Data Collection:

The sensors continuously measure air quality parameters, and the microcontroller processes this

data in real-time. Combine data from multiple sensors to provide a comprehensive view of air quality. Advanced algorithms can be used to fuse and analyse data to generate meaningful insights.

Data Transmission to Cloud:

Use the communication module to send processed data to the cloud platform securely. Create a user-friendly mobile app and web interface that allows users to access real-time air quality data from anywhere.

Geolocation and Mapping:

Use GPS data to map air quality across different locations. This feature can help users identify pollution hotspots and make informed decisions about their activities.

Cloud-Based Analytics:

Implement basic analytics on the cloud platform to identify patterns and trends in air quality data.

User-Friendly Dashboard:

Create a user-friendly dashboard accessible through a web or mobile app, allowing users to view real-time air quality information.

Automated Alerts:

Set up automated alerts on the dashboard or through notifications to inform users when air quality levels exceed predefined thresholds.

Public Engagement:

Share the air quality data with the public through the dashboard, raising awareness and encouraging individuals to take informed actions.

Scalability:

Design the system for scalability, allowing easy deployment of additional monitoring stations as needed.

Air Quality Index (AQI) Calculation:

Calculate and display AQI values, which provide a simple and standardized way to communicate air quality to the public.

IOT Connectivity:

Use low-power, long-range IOT communication protocols like LoRa or NBIoT for efficient data transmission, especially in remote areas.

Public Data API:

Offer an API for developers and researchers to access and use the collected air quality data for various applications.

Benefits:

. Affordability

. Accessibility

. Real-Time Monitoring

. Community Awareness

. Healthier Environment

. Public Health

. Early Warning Systems

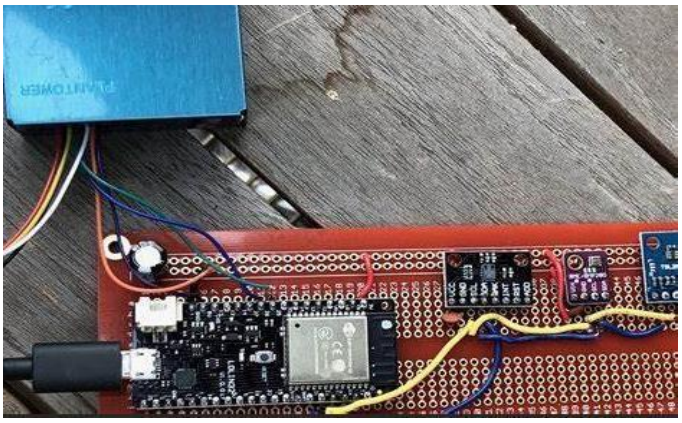
. Climate Goals

. Compliance and Regulation

Sensors we used for the detection of air quality:

For Particulate Matter (PM) sensors, two commonly used models are the Plantower PMS5003 and PMS7003. Both of these sensors are known for their reliability and accuracy in measuring fine particulate matter in the air. The below explains about the materials for measuring AQ.

Plantower PMS5003:



Features:

Measures PM1.0, PM2.5, and PM10 particulate concentrations.

High sensitivity and fast response time.

Suitable for indoor and outdoor air quality monitoring.

Usage:

Often used in air quality monitoring devices, weather stations, and environmental monitoring systems.

Plantower PMS7003:

Features:

Measures PM1.0, PM2.5, and PM10 particulate concentrations.

Compact and lightweight design.

UART and PWM communication options.

Usage:

Widely used in air quality monitoring applications, IoT devices, and environmental research

PM1.0, PM2.5, and PM10 refer to different size fractions of particulate matter in the air. These are commonly used terms in air quality monitoring to describe the sizes of particles being measured

PM1.0 (Particulate Matter 1.0):

Size Range: Particles with aerodynamic diameters less than or equal to 1.0 micrometers (µm).

Features:

Represents the smallest particles measured in air quality monitoring.

Includes fine and ultrafine particles, which can penetrate deeper into the respiratory system.

Usage:

Monitoring ultrafine particles that may have adverse health effects.

Used in research and studies focusing on the smallest particle sizes.

PM2.5 (Particulate Matter 2.5):

Size Range: Particles with aerodynamic diameters less than or equal to 2.5 micrometers (µm).

Features:

Represents fine particles that can be inhaled into the respiratory system.

Includes particles from various sources, such as combustion processes and natural sources.

Usage:

Widely used in air quality monitoring due to its relevance to human health.

Regulatory standards often specify limits for PM2.5 concentrations.

PM10 (Particulate Matter 10):

Size Range: Particles with aerodynamic diameters less than or equal to 10 micrometers (μm).

Features:

Represents larger particles, including coarse particles.

Includes particles from both human-made and natural sources.

Usage:

Used in air quality monitoring to assess the overall level of inhalable particles.

Regulatory standards often specify limits for PM10 concentrations.

These are the different sizes of particles present in the air

Now, next to this we are going to place these sensors in outdoor for better identification of the air quality:

Outdoor Placement:

Street-Level Monitoring:

Placing PM sensors at street level is common in urban areas to monitor traffic-related pollution. This can help assess the impact of vehicular emissions on air quality.

SOFTWARE COMPONENTS:

Air quality monitoring using IoT typically involves a combination of software components for data collection, processing, visualization, and communication. Here are the key software components used in an air quality monitoring system:

Sensor Drivers and Libraries:

These software components are responsible for interfacing with the air quality sensors. They provide the necessary drivers and libraries to communicate with the sensors and retrieve data.

IoT Middleware or Cloud Platform:

IoT Platforms: Platforms like AWS IoT, Google Cloud IoT, Microsoft Azure IoT, and IBM Watson IoT provide scalable, cloud-based solutions for data management, analytics, and device management.

IoT Edge Computing: Some data processing and analysis can be done at the edge (on the device) to reduce latency and minimize data transmission.

Alerting and Notification Systems:

Software that can trigger alerts and notifications based on predefined thresholds or anomaly detection. Notifications can be sent via email, SMS, or push notifications to mobile apps

.

Database Management Systems (DBMS):

Used for storing historical air quality data. Popular options include SQL databases (e.g., PostgreSQL, MySQL) or NoSQL databases (e.g., MongoDB).

Components:

1. Arduino board (e.g., Arduino Uno or Arduino Nano)
2. Air quality sensor (e.g., MQ series sensor)
3. Wi-Fi module (e.g., ESP8266 or ESP32) for IoT connectivity
4. A computer or Raspberry Pi for running the Python program
5. Blynk or a similar IoT platform for data visualization (optional)

Steps:

1. Hardware Setup:

>Connect the air quality sensor to the Arduino following the sensor's datasheet and your chosen Arduino model.
> Connect the Wi-Fi module to the Arduino, ensuring it can connect to your Wi-Fi network.

2. Arduino Code:

Write an Arduino sketch to read data from the air quality sensor and send it to the IoT platform (e.g., Blynk). Make sure to include the necessary libraries for your sensor and Wi-Fi module.
Here's a simplified example using the Adafruit CCS811 air quality sensor and an ESP8266 Wi-Fi module:

```
#include &lt;Wire.h&gt;
#include &lt;Adafruit_CCS811.h&gt;
#include &lt;ESP8266WiFi.h&gt;
#include &lt;BlynkSimpleEsp8266.h&gt;
char auth[] = &quot;Your_Blynk_Auth_Token&quot;;
char ssid[] = &quot;Your_WiFi_SSID&quot;;
char pass[] = &quot;Your_WiFi_Password&quot;;
Adafruit_CCS811 ccs;
void setup() {
Serial.begin(115200);
Blynk.begin(auth, ssid, pass);
ccs.begin();
}
void loop() {
if (!ccs.available()) {
return;
}
```

```
if (!ccs.readData()) {
return;
}
float temperature = ccs.calculateTemperature();
float humidity = ccs.calculateHumidity();
int eCO2 = ccs.geteCO2();
int TVOC = ccs.getTVOC();
Serial.print(&amp;quot;eCO2: &amp;quot;);
Serial.println(eCO2);
Serial.print(&amp;quot;TVOC: &amp;quot;);
Serial.println(TVOC);
Blynk.virtualWrite(V1, eCO2); // Send eCO2 data to virtual pin V1
Blynk.virtualWrite(V2, TVOC); // Send TVOC data to virtual pin V2
Blynk.run();
delay(1000); // Delay between readings
```

Code that integrates sensor to the environment:

```
import time

import requests

import random

# Data-sharing platform configuration (replace with your platform's details)

api_url = "https://platform.openai.com/account/api-keys"

api_key = "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"


def generate_simulated_data():
    # Simulate PM2.5 and PM10 values
    pm25 = random.uniform(5, 20)
    pm10 = random.uniform(10, 30)
    return pm25, pm10


def send_data_to_platform(pm25, pm10):
    payload = {"api_key": api_key, "field1": pm25, "field2": pm10}
    try:
        response = requests.post(api_url, data=payload)
        print("Simulated data sent successfully")
    except Exception as e:
        print(f"Error sending simulated data: {e}")
```

```
if __name__== "_main_":
    try:
        while True:
            pm25, pm10 = generate_simulated_data()
            print(f"Simulated PM2.5: {pm25} µg/m³, PM10: {pm10} µg/m³")
            send_data_to_platform(pm25, pm10)
            time.sleep(300) # Adjust this based on your data-sharing platform's limitations
    except KeyboardInterrupt:
            print("Script terminated by user.")
```

Output:

```
Simulated PM2.5: 15.018356831988099 µg/m³, PM10: 20.09326754324212
µg/m³
Simulated data sent successfully
Simulated PM2.5: 11.843433202538392 µg/m³, PM10: 25.414319872755364
µg/m³
Simulated data sent successfully
Simulated PM2.5: 13.656690136913404 µg/m³, PM10: 22.554439980378543
µg/m³
Simulated data sent successfully
Simulated PM2.5: 15.180759281165448 µg/m³, PM10: 25.21239236030451
µg/m³
Simulated data sent successfully

Simulated PM2.5: 8.562569641229786 µg/m³, PM10: 25.159651799641956

µg/m³

Simulated data sent successfully

Simulated PM2.5: 19.934493649751055 µg/m³, PM10: 24.840927601852826

µg/m³
```

Building the project by developing the data-sharing platform.

Data-sharing platform:

After building the initial data display platform using Arduino or similar IoT devices, the next levels of development for a data-sharing platform typically involve enhancing the platform's functionality, scalability, and security

Data Storage and Database Integration:

Implement a database to store historical air quality data. This will allow you to analyze trends over time.

Real-Time Data Streaming:

Enable real-time data streaming capabilities to provide users with immediate access to the latest air quality information.

User Authentication and Authorization:

Implement user authentication and authorization to ensure that only authorized users can access the data-sharing platform.

Data Visualization:

Enhance the user interface with interactive data visualizations such as charts, graphs, and maps to provide a richer user experience.

Geospatial Integration:

Integrate geospatial features to allow users to view air quality data on a map, which can be especially useful for monitoring air quality in different locations.

Alerting and Notifications:

Implement alerting and notification systems that inform users when air quality reaches certain thresholds or when specific gases are detected at harmful levels.

Data Analysis and Reporting:

Provide data analysis tools that allow users to generate reports and insights from the collected data.

Scalability and Performance:

Ensure that the platform can handle a growing volume of data and users. Consider using cloud-based solutions for scalability.

Security and Privacy:

Implement strong security measures to protect data integrity and user privacy. This includes encryption, access controls, and compliance with data protection regulations.

Collaboration and Sharing Features:

Add features that allow users to collaborate and share data with others, fostering a community of users interested in air quality.

Continuous Monitoring and Maintenance:

Regularly monitor and maintain the platform to ensure data accuracy and system reliability.

Documentation and Support:

Create user and developer documentation to assist users and other developers in using and integrating your platform.

Compliance with Environmental Standards:

Ensure that your platform adheres to environmental standards and regulations related to air quality monitoring.

Creating a platform that displays real-time air quality data. Design the platform to receive and display air quality data sent by the IOT devices.

Code that has been used for creating a web page are:

HTML (index.html): This file contains the structure and content of your web page.

JavaScript (app.js): This file contains the JavaScript code that simulates air quality data and displays it on your web page.

CSS (styles.css): This file contains the styling rules to control the appearance of your web page.

HTML code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="styles.css">
    <title>Air Quality Data Platform</title>
</head>
<body>
    <div class="container">
        <h1>Air Quality Data Platform</h1>
        <button id="simulateButton">Simulate IoT Data</button>
        <div id="dataDisplay"></div>
    </div>
    <script src="app.js"></script>
</body>
</html>
```

CSS:

```
body {
    font-family: Arial, sans-serif;
```

```
    margin: 0;
    padding: 0;
}

.container {
    max-width:    600px;
    margin:  50px  auto;
    text-align: center;
}

button {
    padding: 10px 20px;
    font-size: 16px;
    cursor: pointer;
}

#dataDisplay {
    margin-top: 20px;
    padding: 20px;
    border: 1px solid #ddd;
    border-radius: 8px; text-
    align: left;
}
```

Java script:

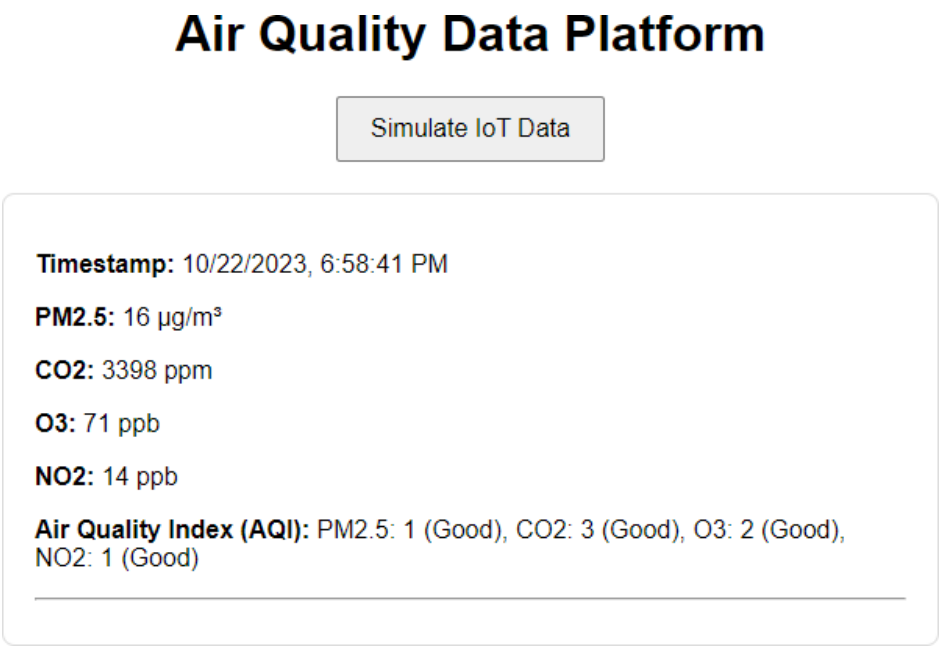
```
document.addEventListener('DOMContentLoaded', () => {
    const simulateButton = document.getElementById('simulateButton'); const
    dataDisplay = document.getElementById('dataDisplay');

    simulateButton.addEventListener('click', () => { const
        simulatedData = generateSimulatedData();
        displayData(simulatedData);
    });

    function generateSimulatedData() {
        const timestamp = new Date().toLocaleString(); const
        airQualityData = {
            PM25: Math.floor(Math.random() * 50),
            CO2: Math.floor(Math.random() * 5000), O3:
            Math.floor(Math.random() * 100), NO2:
            Math.floor(Math.random() * 50)
        };

        return { timestamp, airQualityData };
    }

    function displayData(data) {
        const newDataElement = document.createElement('div');
        newDataElement.innerHTML = `<p><strong>Timestamp:</strong>
```

```
${data.timestamp}</p>
                        <p><strong>PM2.5:</strong> ${data.airQualityData.PM25} µg/m³</p>
```

```
                        <p><strong>CO2:</strong> ${data.airQualityData.CO2} ppm</p>
                        <p><strong>O3:</strong> ${data.airQualityData.O3} ppb</p>
                        <p><strong>NO2:</strong> ${data.airQualityData.NO2} ppb</p>
                        <p><strong>Air Quality Index (AQI):</strong>
${getSpecificAirQuality(data.airQualityData)}</p>
                        <hr>`;

        dataDisplay.prepend(newDataElement);
    }


    function getSpecificAirQuality(airQualityData) {
        const pm25AQI = calculateAQI(airQualityData.PM25, [0, 12, 35, 55, 150]);
        const co2AQI = calculateAQI(airQualityData.CO2, [0, 400, 1000, 2000, 5000]);
        const o3AQI = calculateAQI(airQualityData.O3, [0, 54, 70, 85, 100]);
        const no2AQI = calculateAQI(airQualityData.NO2, [0, 25, 50, 100, 200]);

        const pm25Level = getAirQualityLevel(pm25AQI); const
        co2Level = getAirQualityLevel(co2AQI); const o3Level =
        getAirQualityLevel(o3AQI);
        const no2Level = getAirQualityLevel(no2AQI);

        return `PM2.5: ${pm25AQI} (${pm25Level}), CO2: ${co2AQI} (${co2Level}), O3:
${o3AQI} (${o3Level}), NO2: ${no2AQI} (${no2Level})`;
    }


    function calculateAQI(concentration, levels) { for
        (let i = 0; i < levels.length - 1; i++) {
            const [low, high, lowIndex, highIndex] = [levels[i], levels[i + 1], i, i + 1]; if
            (concentration >= low && concentration <= high) {
                return Math.round(((highIndex - lowIndex) / (high - low)) * (concentration - low) +
lowIndex);
            }

        }

        return 0; // Default to the lowest level
    }


    function getAirQualityLevel(aqi) { if
        (aqi <= 50) {
            return "Good";
        } else if (aqi <= 100) { return
            "Moderate";
        } else if (aqi <= 150) {
            return "Unhealthy for Sensitive Groups";
        } else if (aqi <= 200) { return
            "Unhealthy";
        } else if (aqi <= 300) { return
            "Very Unhealthy";
        } else {
            return "Hazardous";
        }
```

```
    }
});
```

The above code gives the website as:



Conclusion:

Thus, to prevent the air quality and to analyse the quality of air we have innovated a sensor through which we can monitor the quality on daily manner and also enables us to take some preventive measures to avoid the air pollution

The webhost that has been responsible for launching our webpage which has air quality is as follows:

https://shruthisrini.neocities.org/AQM/

we have used neocities as the platform to post our webpage as of to be viewed by everyone.