**Phase 4 project**

**Project Title: AIR QUALITY MONITORING**

**Project ID:** proj\_223737\_Team\_3

**College:** Gnanamani College of Technology

**College code:** 6208

**Branch:** B.Tech-Information Technology

**Year:** IIIrd year

Team Members:

Balachandar.K(620821205006)

Jana.V(620821205020)

Kathirvanan.S(620821205028)

Pravinkumar.K(620821205041)

Srikanth.S(620821205056)

**AIR QUALITY MONITORING**

**Definition:**

An IoT-based air pollution monitoring system is an ideal solution that can provide real-time data and insights about the air quality in a particular area. An IoT based air pollution monitoring system consists of several hardware and software components that works together to collect and process data.

**PHASE 4**

Creating a real-time air quality data platform involves multiple components and technologies. Here's a simplified outline of how you can design such a platform using web development technologies:

**1. Data Source (IoT Devices):**

* Set up IoT devices like air quality sensors to collect data.
* These devices should be capable of sending data to a server in real-time.

**2. Backend (Server):**

* Use a server-side technology like Node.js or Python (Django/Flask) to receive and process data from the IoT devices.
* Implement WebSockets for real-time data updates. You can use libraries like Socket.io (Node.js) or Django Channels (Python).
* Store data in a database (e.g., PostgreSQL or MongoDB).

**3. APIs:**

* Create RESTful or WebSocket-based APIs to communicate between the IoT devices, server, and frontend.
* Ensure security and authentication mechanisms for data transmission.

**4. Frontend (Web App):**

* Use HTML, CSS, and JavaScript to create a user interface for displaying air quality data.
* Fetch real-time data from the server using APIs.
* Implement data visualization using libraries like D3.js, Chart.js, or Leaflet for maps.
* Update the UI in real-time as new data arrives.
* Provide user-friendly features like data filtering, search, and historical data views.

**5. User Authentication:**

* Implement user authentication to secure access to the platform.
* Allow users to create accounts and log in to access personalized data.

**6. Data Analytics:**

* Implement data analysis and alerting systems for abnormal air quality conditions.
* Send notifications to users when air quality levels reach predefined thresholds.

**7. Deployment:**

* Host your web application and server on a cloud platform like AWS, Azure, or Google Cloud.
* Ensure scalability to handle a growing number of IoT devices and users.

**8. Testing and Optimization:**

* Thoroughly test the platform to ensure its reliability and performance.
* Optimize the frontend and backend for speed and efficiency.

**9. Documentation and Support:**

* Provide clear documentation for users and administrators.
* Offer customer support and troubleshoot issues.

**10. Maintenance:**

* Regularly update and maintain the platform to ensure it remains up-to-date and secure.

**Creating a website:**

Creating a real-time air quality data platform using HTML, CSS, and JavaScript is a simplified web development task. Here's a basic example of how you can set up a frontend to display real-time air quality data:

**1. HTML:**

Create the structure of your webpage. You might have a layout like this:

<!DOCTYPE html>

<html>

<head>

<title>Real-time Air Quality</title>

<link rel="stylesheet" type="text/css" href="styles.css">

</head>

<body>

<div id="air-quality">

<h1>Real-time Air Quality</h1>

<p>Air Quality: <span id="aqi">Loading...</span></p>

</div>

<script src="script.js"></script>

</body>

</html>

**2. CSS (styles.css):**

Style your webpage to make it visually appealing:

body {

font-family: Arial, sans-serif;

background-color: #f2f2f2;

text-align: center;

}

#air-quality {

background-color: #fff;

padding: 20px;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);

}

h1 {

color: #333;

}

#aqi {

font-size: 24px;

color: #008000;

}

**3. JavaScript (script.js):**

Use JavaScript to fetch and display real-time air quality data. For simplicity, we'll simulate the data:

// Simulated real-time data (replace this with actual data retrieval logic)

function fetchAirQuality() {

return Math.floor(Math.random() \* 301); // Simulated Air Quality Index (AQI)

}

function updateAirQuality() {

const aqiElement = document.getElementById('aqi');

const aqi = fetchAirQuality();

aqiElement.textContent = aqi;

// Update the AQI color based on thresholds (customize as needed)

if (aqi <= 50) {

aqiElement.style.color = '#008000'; // Good (Green)

} else if (aqi <= 100) {

aqiElement.style.color = '#FFFF00'; // Moderate (Yellow)

} else if (aqi <= 150) {

aqiElement.style.color = '#FFA500'; // Unhealthy for Sensitive Groups (Orange)

} else {

aqiElement.style.color = '#FF0000'; // Unhealthy (Red)

}

}

// Periodically update air quality data (every 5 seconds in this example)

setInterval(updateAirQuality, 5000);

updateAirQuality(); // Call once to display initial data

In this example, we're using JavaScript to periodically fetch and display simulated air quality data, updating the AQI value and its color based on defined thresholds.