

AIR QUALITY MONITORING

Team Members:

Mukesh M (810021106051)

Sriram S (810021106081)

Vengatesan M (810021106091)

Vengatesan T (810021106092)

Sushmitha K (810021106083)

**AIR QUALITY MONITORING**

IOT\_Phase5 Project

# Introduction:

In this project we will see about the project's objectives, IoT device setup, platform development, and code implementation. We Included diagrams, schematics, and screenshots of the IoT devices and data-sharing platform and how the real-time air quality monitoring system can raise public awareness about air quality and health impacts.

Objectives:

The objectives of air quality monitoring using IoT (Internet of Things) include:

1. Real-Time Data: To provide real-time and continuous data on various air pollutants, enabling timely responses to changing air quality conditions.

2. Pollution Source Identification: To pinpoint the sources of pollution and identify trends, allowing for targeted pollution control measures and urban planning.

3. Public Awareness: To raise awareness among the public about air quality issues and their health impacts through easy access to air quality information.

4. Health Protection: To protect public health by providing early warnings and recommendations when air quality deteriorates to unsafe levels, especially for vulnerable populations.

5. Regulatory Compliance: To assist government agencies and industries in ensuring compliance with air quality regulations and standards.

6. Research and Analysis: To support scientific research and analysis of air quality data, aiding in the development of effective pollution control strategies.

7. Data-Driven Decision-Making: To provide decision-makers with data-driven insights for formulating policies and initiatives to improve air quality.

8. Environmental Sustainability: To contribute to the sustainable management of environmental resources by promoting reduced emissions and a cleaner environment.

9. Urban Planning: To guide urban development and infrastructure decisions, including the location of sensitive facilities, based on air quality considerations.

10. Global Monitoring: To contribute to a global network of air quality data, helping in international efforts to combat air pollution and climate change.

11. Adaptive Control: To enable dynamic control systems that respond to air quality data, such as adjusting traffic signals, reducing industrial emissions, or controlling ventilation in buildings.

12. Energy Efficiency: To optimize energy consumption in buildings and transportation based on air quality data, promoting energy efficiency and reducing greenhouse gas emissions.

# Details about components:

ARDUINO :



The Arduino Uno is the board we will be using for this project. Other Arduino boards may work for this project as well, but this board is all I needed. It’s a great price and we can purchase it on many different websites.

ESP8266 Wi-Fi chip:

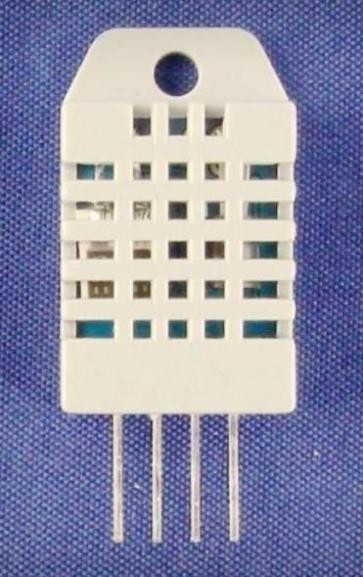


This is the chip that will allow our project to send data wirelessly. In order to program it, we must build a separate circuit and use a separate code then we will use for the main air quality monitor build.

**MQ-135 Sensor**:



The MQ-135 gas sensor is capable of detecting a bunch of different gases. We could build an air quality sensor for individual gases, such as carbon monoxide, but that would only give us a small snapshot of the overall air quality.

**DHT-22 Sensor**:

DHT22 output calibrated digital signal. It utilizes exclusive digital-signal-collecting- technique and humidity

sensing technology, assuring its reliability and stability.Its sensing elements is connected with 8-bit single-chip.

# ESP8266 Python Code (for forwarding data to ThingSpeak):

import serial import requests import time

ser = serial.Serial('COM4', 9600) # Replace 'COM4' with our ESP8266's serial port

def send\_to\_thingspeak(api\_key, field1, field2, field3): url =

f"https://api.thingspeak.com/update?api\_key={api\_key}&field1={field1}&field2={field2}&field3={f ield3}"

response = requests.get(url)

if response.status\_code == 200:

print("Data sent to ThingSpeak successfully") else:

print("Failed to send data to ThingSpeak")

while True:

data = ser.readline().decode('utf-8').strip() print(data)

if data.startswith("SENDING "): data = data[len("SENDING "):] values = data.split(",")

if len(values) == 3:

temperature, humidity, air\_quality = values send\_to\_thingspeak("OUR\_API\_KEY", temperature, humidity, air\_quality)

time.sleep(10) # Adjust the delay as needed

Make sure to replace `"OUR\_API\_KEY"` in the Python script with our actual ThingSpeak API key. Also,adjust the serial port name in the ESP8266 Python script to match our specific setup.

This code reads temperature, humidity, and air quality data from DHT22 and MQ135 sensors connected to the Arduino, sends it to the ESP8266 through serial communication, and then forwards it to ThingSpeak.

To connect an Arduino with a DHT22 temperature and humidity sensor and an MQ135 gas sensor to ThingSpeak through an ESP8266 WiFi module, we'll need to write code for both the Arduino and the ESP8266 module. Here's a basic example of the Arduino code and the Python script to collect data and send it to ThingSpeak:

# Arduino Code (for collecting sensor data):

#include "DHT.h"

#include "Adafruit\_Sensor.h" #include <MQ135.h>

#define DHTPIN 2 // DHT22 data pin #define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

MQ135 gasSensor = MQ135(A0);

void setup() { Serial.begin(9600);

ThingSpeak.begin(client); // Initialize ThingSpeak

}

void loop() {

float temperature = dht.readTemperature(); float humidity = dht.readHumidity();

float airQuality = gasSensor.getPPM();

Serial.print("Temperature: "); Serial.print(temperature); Serial.print("°C, Humidity: "); Serial.print(humidity); Serial.print("%, Air Quality: "); Serial.print(airQuality); Serial.println(" ppm");

// Send data to ESP8266 for forwarding to ThingSpeak Serial.print("SENDING ");

Serial.print(temperature); Serial.print(","); Serial.print(humidity);

Serial.print(","); Serial.print(airQuality); Serial.println(" TO ESP8266");

// Send data to ESP8266 here using Serial communication Serial.print("AT+CIPSEND="); Serial.println(data.length());

Serial.print("GET /update?api\_key=OUR\_API\_KEY&field1="); Serial.print(temperature);

Serial.print("&field2="); Serial.print(humidity); Serial.print("&field3="); Serial.print(airQuality); Serial.println(" HTTP/1.0");

Serial.println("Host: api.thingspeak.com");

Serial.println("Content-Type: application/x-www-form-urlencoded"); Serial.println("User-Agent: ESP8266");

Serial.println();

delay(10000); // Update every 10 seconds

}

# IOT DEVICE SETUP:

# 

# 1. Air Quality Sensors: These sensors measure various air pollutants such as particulate matter (PM), gases like carbon monoxide (CO), nitrogen dioxide (NO2), sulfur dioxide (SO2), ozone (O3), and volatile organic compounds (VOCs).

# 2. Meteorological Sensors: These sensors collect data on temperature, humidity, wind speed, wind direction, and

# atmospheric pressure, which are essential for understanding how weather conditions affect air quality.

# 3. Data Logger/Processor: A device or system that collects, stores, and processes the data from the sensors. It may include microcontrollers or dedicated data loggers.

# 4. Communication Modules: IoT devices need communication capabilities to transmit data to a central server or database. This can be achieved through Wi-Fi, cellular networks, LoRa, or other wireless communication protocols.

# 5. Power Supply: Depending on the location of the monitoring station, IoT devices may be powered by batteries, solar panels, or an external power source.

# 6. GPS Module: To provide location data, which is crucial for mapping air quality data to specific geographical areas.

# 7. Enclosure/Protection: To shield the IoT devices from environmental factors like rain, dust, and extreme temperatures.

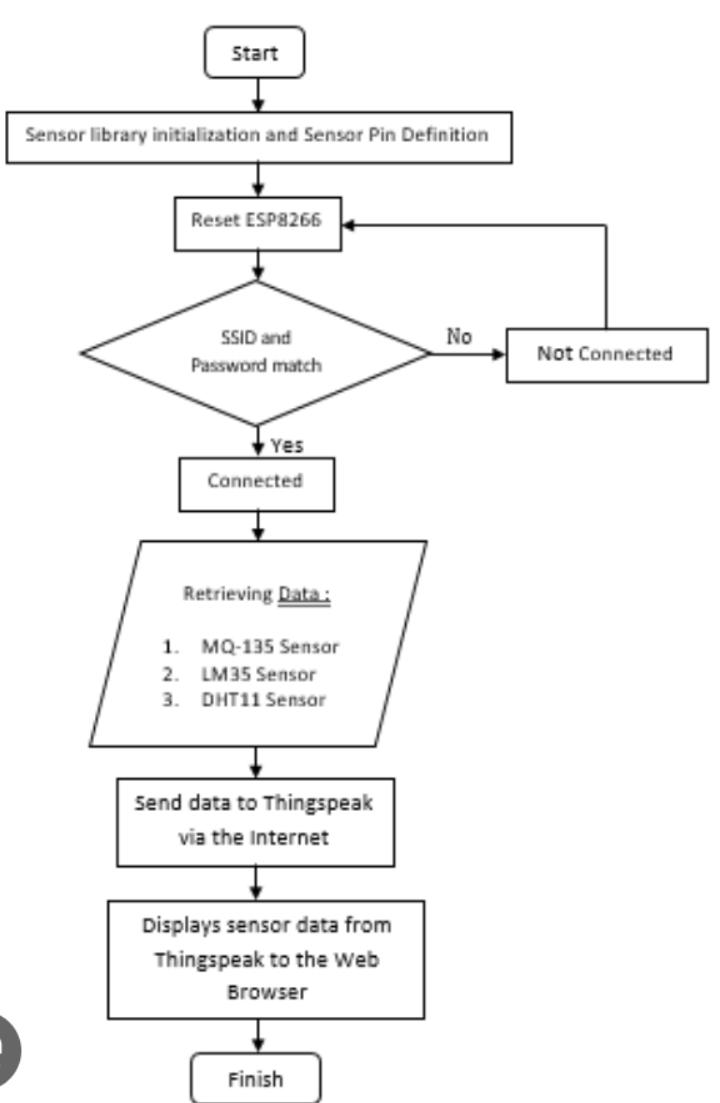
8. Remote Control and Monitoring: In some cases, IoT devices may include remote control features to adjust monitoring parameters and monitor device status remotely.

9. Data Transmission and Visualization Software: This software is used to collect, store, analyze, and visualize the data collected by IoT devices. It's typically cloud-based and allows real-time access to air quality information.

10. User Interface: A user-friendly interface, often accessible via a web application or mobile app, for users to access air quality data and receive alerts or notifications.

PLATFORM DEVELOPMENT:

Developing a platform for air quality monitoring involves creating the software and infrastructure necessary to collect, store, analyze, and present air quality data to users. Here are the key steps in platform development:

1. Define Requirements:

- Clearly define the objectives and requirements of the platform, including the types of data to be collected, the user interface, and any specific features like data visualization or alerts.

2. Select Technology Stack:

- Choose the technology stack for development, including programming languages, frameworks, and databases. The choice

depends on our project's needs and our development team's expertise.

3. Data Collection and Integration:

- Set up data connectors to collect data from monitoring devices. Ensure data is gathered efficiently and accurately. This may involve integrating with IoT devices through APIs or communication protocols.

4. Data Storage:

- Design a database structure to store air quality data securely.

Consider factors like data redundancy, scalability, and data retention policies.

5. Data Processing and Analysis:

- Develop algorithms and processes for data processing and analysis. This could include real-time data validation, aggregation, and the calculation of air quality indices.

6. User Interface (UI) Development:

* Create a user-friendly web application or mobile app for users to access air quality data. Design interactive dashboards for data visualization.

7. User Authentication and Access Control:

* Implement user authentication and access control mechanisms to ensure that only authorized users can access certain data and features.

8. Alerts and Notifications:

* Develop a system for generating and sending alerts and notifications to users when air quality reaches specific thresholds or when other events occur.

9. Mapping and Geographic Information System (GIS):

* If location data is collected, incorporate mapping and GIS functionality to visualize air quality data on maps.

10. Scalability and Performance Optimization:

* Ensure the platform is designed to scale to accommodate a growing amount of data and users. Optimize performance for fast data retrieval and response times.

11. Data Security:

* Implement robust security measures to protect data from unauthorized access and cyber threats. Encrypt sensitive information and regularly update security protocols.

12. Data Quality Control:

* Develop tools and processes for data quality control, including outlier detection and error correction.

13. Reporting and Data Export:

* Provide options for users to generate reports and export data for further analysis or regulatory compliance.

14. Integration with External Systems:

* If needed, integrate the platform with external systems or services, such as government air quality monitoring networks or weather data sources.

15. Testing and Quality Assurance:

* Thoroughly test the platform to identify and fix any issues or bugs. Conduct user testing to ensure the platform meets user expectations.

16. Documentation:

* Create comprehensive documentation for users, administrators, and developers to understand how to use, maintain, and troubleshoot the platform.

17. Deployment and Maintenance:

* Deploy the platform to a production environment and establish a maintenance plan to ensure ongoing functionality, updates, and support.

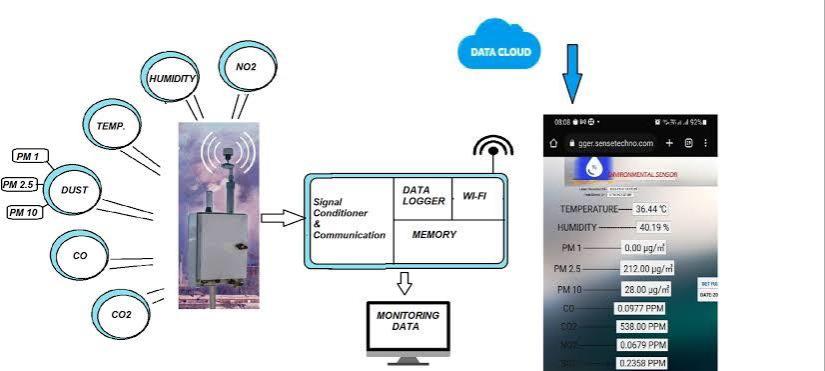
18. User Training:

* Train users and administrators on how to use the platform effectively and interpret the air quality data.

19. Monitoring and Analytics:

* Set up tools for monitoring platform performance and user engagement. Use analytics to make data-driven improvements.

20. Compliance and Regulation:

* Ensure the platform complies with any relevant data protection and regulatory requirements.
* 

USING WEB DEVELOPMENT CREATE A PLATFORM THAT DISPLAYS REAL-TIME AIR QUALITY DATA:

HTML code for webpage:

<!DOCTYPE html>

<html>

<head>

<title>Real-Time Air Quality Monitor</title>

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

</head>

<body>

<img src="https://www.ppsthane.com/wp-content/uploads/2023/02/IoT-based-

Air-Pollution-Monitoring-System.png" alt="image" height="340" width="400">

<h1>Real-Time Air Quality Monitor</h1>

<div id="data-container">

<div class="data">

<h2>Temperature</h2>

<p id="temperature">Loading...Getting data from Wifi module</p>

</div>

<div class="data">

<h2>Humidity</h2>

<p id="humidity">Loading...Getting data from Wifi module</p>

</div>

<div class="data">

<h2>Particulate Matter Level</h2>

<p id="pm-level">Loading...Getting data from Wifi module</p>

</div>

<div class="data">

<h2>Air Quality Level</h2>

<p id="aqi-level">Loading...Getting data from Wifi module</p>

</div>

</div>

<div id="refresh-button">

<button onclick="fetchAirQualityData()">Refresh Data</button>

</div>

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

</body>

</html>

CSS code:

body {

font-family: Arial, sans-serif; text-align: center;

}

h1 { color: lightsalmon;

}

.data { display: inline-block; margin: 20px; padding: 20px; background-color: lightblue; border: 1px solid #ccc; border-radius: 8px;

}

#refresh-button { margin-top: 20px; }

button { padding: 10px 20px;

font-size: 16px;

background-color: #0073e6;

color: white; border: 1px solid #ccc; cursor: pointer;

}

JavaScript code:

function fetchAirQualityData() {

fetch('http://our-esp8266-ip-address/data') // Replace with our ESP8266 IP

.then(response => response.json())

.then(data => { document.getElementById("temperature").textContent = `Temperature:

${data.temperature}°C`; document.getElementById("humidity").textContent = `Humidity:

${data.humidity}%`;

document.getElementById("pm-level").textContent = `Particulate Matter Level: ${data.pmLevel} µg/m³`; document.getElementById("aqi-level").textContent = `Air Quality Level: ${data.aqiLevel}`;

})

.catch(error => console.error('Error fetching data:', error));

}

// Initial data fetch on page load fetchAirQualityData();

// Set up an interval to periodically refresh the data (e.g., every 5 minutes) setInterval(fetchAirQualityData, 300000); // 300000ms = 5 minutes

Code to connect esp8266 wifi module with webpage:

#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

const char\* ssid = "OurNetworkSSID"; const char\* password = "OurNetworkPassword";

ESP8266WebServer server:

// Simulated air quality data (replace with actual data) float temperature = 25.0; float humidity = 50.0; float particulateMatter = 15.0;

String airQuality = "Good";

void setup() {

// Connect to Wi-Fi

WiFi.begin(ssid, password); while (WiFi.status() != WL\_CONNECTED) { delay(1000);

Serial.println("Connecting to WiFi...");

}

Serial.println("Connected to WiFi");

// Set up web server routes server.on("/data", HTTP\_GET, handleData);

server.begin();

Serial.println("HTTP server started");

}

void loop() { server.handleClient();

}

void handleData() {

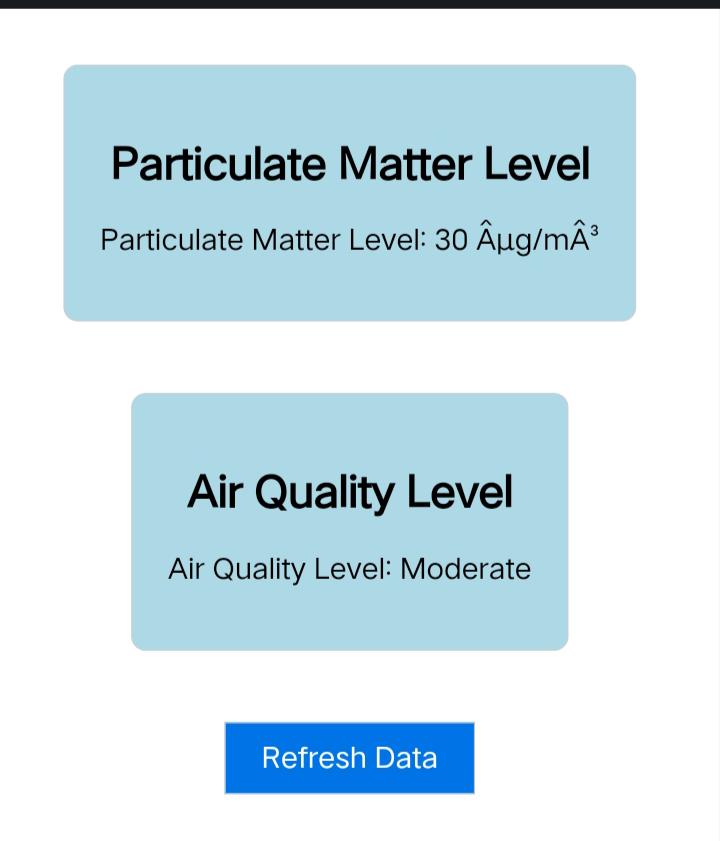
// Replace with code to read and send sensor data String data = "{";

data += "\"temperature\":\"" + String(temperature) + "\","; data += "\"humidity\":\"" + String(humidity) + "\",";

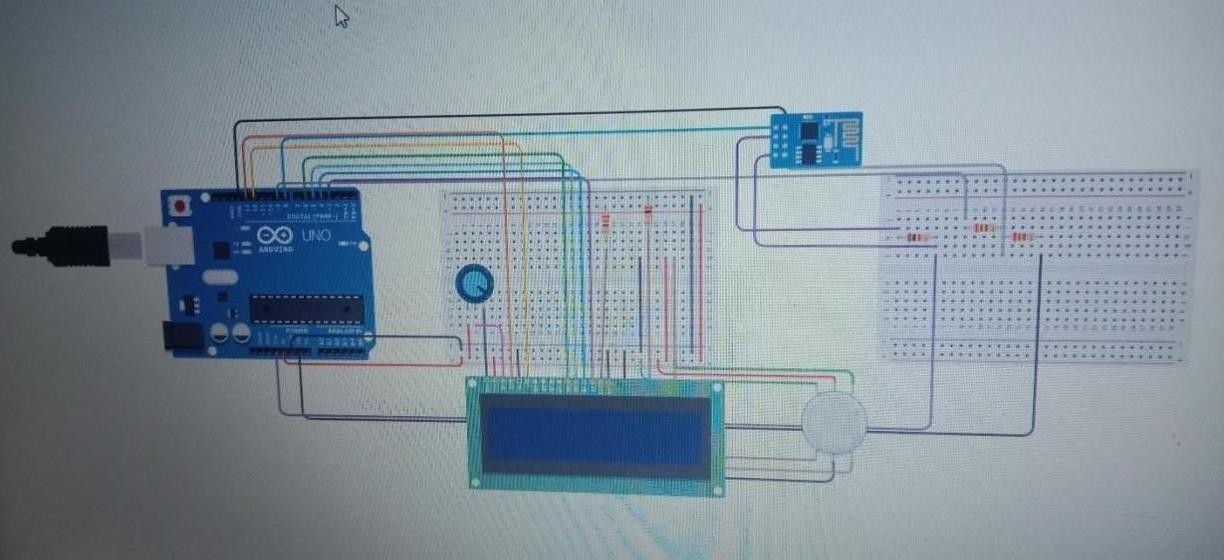
data += "\"particulateMatter\":\"" + String(particulateMatter) + "\","; data += "\"airQuality\":\"" + airQuality + "\""; data += "}";

server.send(200, "application/json", data);

}



IOT DEVICE CONNECTION:



•Connect the ESP8266 module to the Arduino as follows:

* ESP-01 VCC to 3.3V on the Arduino
* ESP-01 GND to GND on the Arduino
* ESP-01 TX to an Arduino RX pin (e.g., RX on the Arduino)
* ESP-01 RX to an Arduino TX pin (e.g., TX on the Arduino)
* GPIO0 on the ESP-01 should be connected to GND during programming.

•Connect the MQ-135 sensor to the Arduino as follows:

* MQ-135 VCC (or +) pin to Arduino 5V
* MQ-135 GND (or-) pin to Arduino GND
* MQ-135 AOUT pin to an analog input pin on the Arduino (e.g., A0)
* MQ-135 DOUT (or digital output) is optional and can be connected to a digital input pin if we want to use it, but it's not necessary for basic readings.
* - Connect the DHT22 sensor to the Arduino as follows:
  + DHT22 VCC pin to Arduino 5V
  + DHT22 GND pin to Arduino GND
  + DHT22 DATA pin to a digital pin on the Arduino (e.g., D2)
  + Place a 10kΩ resistor between the VCC and DATA pins of the DHT22 sensor.

Public awareness about air quality and health impacts:

A real-time air quality monitoring system can raise public awareness about air quality and health impacts in several ways:

1. Immediate Information: Real-time monitoring provides up-to-the-minute data on air quality, making people aware of current conditions and any sudden changes due to factors like pollution, weather, or industrial activity.

2. Visual Representation: User-friendly apps or websites can display real-time data using color-coded scales or visual indicators, making it easy for the public to understand air quality levels and associated health risks.

3. Alerts and Notifications: These systems can send alerts or notifications to users when air quality deteriorates significantly, prompting them to take precautions or modify their activities to reduce exposure to pollutants.

4. Geographic Specifics: Real-time monitoring can offer location-specific data, helping people identify areas with better or worse air quality. This can be particularly important for those with respiratory conditions or allergies.

5. Long-Term Trends: Over time, these systems collect data that allows the public to observe trends and seasonal variations in air quality, reinforcing the importance of long-term efforts to improve it.

6. Public Engagement: Real-time data can engage and empower the public to advocate for cleaner air and take actions, such as reducing emissions, using public transport, or supporting policies aimed at improving air quality.

7. Health Education: By linking air quality to health impacts (e.g., respiratory diseases, cardiovascular issues), the public becomes more informed about the tangible risks associated with poor air quality.

8. Policy Influence: Access to real-time air quality data can encourage citizens to demand stricter environmental regulations and government initiatives to address air pollution.

# Conclusion:

In conclusion, IoT-based air quality monitoring using Arduino presents a powerful and cost-effective solution for real-time environmental data collection. This system allows for continuous monitoring of air quality parameters, such as particulate matter, gases, and temperature, and enables data transmission to a central server for analysis. By leveraging Arduino's versatility and the connectivity of IoT, we can not only track air quality but also make informed decisions for public health, urban planning, and environmental protection. This technology has the potential to enhance our understanding of air quality and contribute to the creation of cleaner and healthier living environments.