Air Quality Monitoring

Documentation

Objectives:

The Air Quality Monitoring project aims to create a real-time air quality monitoring system using IoT devices to measure and report air quality parameters, such as particulate matter (PM2.5 and PM10), carbon monoxide (CO), nitrogen dioxide (NO2), and temperature. The system also includes a data-sharing platform to visualize and provide access to this data.

The main objectives are:

1. Monitor and collect air quality data from multiple locations.

2. Develop a user-friendly data-sharing platform to provide real-time air quality information.

3. Raise public awareness about the impact of air quality on health and the environment.

IoT Device Setup:

For this project, we use IoT devices equipped with sensors to monitor air quality. The setup includes the following components:

Air Quality: Sensors:These sensors measure various air quality parameters.

Microcontroller : Collects data from the sensors and transmits it to the cloud.

Internet Connectivity: The IoT devices connect to the internet through Wi-Fi or cellular networks.

Power Supply: Devices can be powered by batteries or AC power.

Platform Development:

The data-sharing platform is a crucial component of the project. It is developed using web technologies and cloud services. Here's an overview:

Cloud Infrastructure: Data from IoT devices is sent to cloud servers.

Database: Data is stored in a database for real-time and historical analysis.

Web Application: A web-based dashboard is created to visualize air quality data in real-time. Users can access it through a web browser.

Code Implementation:

The project is implemented using Python for both IoT device programming and web application development. Python libraries for IoT communication, database management, and web development are utilized. The codebase is structured as follows:

IoT Device Code: This code collects sensor data and sends it to the cloud.

Python code:

import time

import json

import paho.mqtt.client as mqtt

# Configuration for your MQTT broker (cloud platform)

mqtt\_broker\_address = "your-mqtt-broker-url"

mqtt\_port = 1883

mqtt\_topic = "air\_quality\_data"

# Simulated sensor data (replace with your actual sensor data)

air\_quality\_data = {

"PM2.5": 15.3,

"PM10": 20.1,

"CO": 0.3,

"NO2": 0.02,

"temperature": 25.5

}

# Connect to the MQTT broker

client = mqtt.Client("AirQualitySensor")

client.connect(mqtt\_broker\_address, mqtt\_port)

while True:

# Simulate sensor data collection

# Replace this with your actual sensor data acquisition code

# You should update the 'air\_quality\_data' dictionary with real sensor data

# Publish the sensor data to the MQTT topic

client.publish(mqtt\_topic, json.dumps(air\_quality\_data))

print("Data published:", air\_quality\_data)

time.sleep(60) # Data sent every 60 seconds (adjust as needed)

# Disconnect from the MQTT broker when done (this may never be reached in a real IoT device)

client.disconnect()

Server Code: Manages incoming data, stores it in the database, and communicates with the web application.

Python code:

from flask import Flask, request, jsonify

import sqlite3

app = Flask(\_\_name\_\_)

# Create an SQLite database

conn = sqlite3.connect('air\_quality\_data.db')

cursor = conn.cursor()

cursor.execute('''

CREATE TABLE IF NOT EXISTS air\_quality (

id INTEGER PRIMARY KEY AUTOINCREMENT,

pm25 REAL,

pm10 REAL,

co REAL,

no2 REAL,

temperature REAL

)

''')

conn.commit()

conn.close()

@app.route('/receive\_data', methods=['POST'])

def receive\_data():

try:

data = request.json

pm25 = data.get('PM2.5')

pm10 = data.get('PM10')

co = data.get('CO')

no2 = data.get('NO2')

temperature = data.get('temperature')

# Store data in the database

conn = sqlite3.connect('air\_quality\_data.db')

cursor = conn.cursor()

cursor.execute("INSERT INTO air\_quality (pm25, pm10, co, no2, temperature) VALUES (?, ?, ?, ?, ?)",

(pm25, pm10, co, no2, temperature))

conn.commit()

conn.close()

return jsonify({'message': 'Data received and stored successfully.'})

except Exception as e:

return jsonify({'error': str(e)})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

Web Application Code: Provides a user interface for data visualization.

Python code:

from flask import Flask, render\_template, request, jsonify

import sqlite3

app = Flask(\_\_name\_\_)

# Define a route to display air quality data

@app.route('/')

def index():

# Fetch air quality data from the database

conn = sqlite3.connect('air\_quality\_data.db')

cursor = conn.cursor()

cursor.execute("SELECT \* FROM air\_quality ORDER BY id DESC LIMIT 10")

data = cursor.fetchall()

conn.close()

# Prepare the data for visualization

timestamps = [entry[0] for entry in data]

pm25\_values = [entry[1] for entry in data]

pm10\_values = [entry[2] for entry in data]

co\_values = [entry[3] for entry in data]

no2\_values = [entry[4] for entry in data]

temperature\_values = [entry[5] for entry in data]

return render\_template('index.html', timestamps=timestamps, pm25=pm25\_values, pm10=pm10\_values, co=co\_values, no2=no2\_values, temperature=temperature\_values)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=8080)

HTML INDEX:

<!DOCTYPE html>

<html>

<head>

<title>Air Quality Data Visualization</title>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

</head>

<body>

<h1>Air Quality Data Visualization</h1>

<canvas id="air-quality-chart" width="800" height="400"></canvas>

<script>

var ctx = document.getElementById('air-quality-chart').getContext('2d');

var chart = new Chart(ctx, {

type: 'line',

data: {

labels: {{ timestamps | tojson }},

datasets: [

{

label: 'PM2.5',

borderColor: 'blue',

data: {{ pm25 | tojson }}

},

{

label: 'PM10',

borderColor: 'green',

data: {{ pm10 | tojson }}

},

{

label: 'CO',

borderColor: 'red',

data: {{ co | tojson }}

},

{

label: 'NO2',

borderColor: 'orange',

data: {{ no2 | tojson }}

},

{

label: 'Temperature',

borderColor: 'purple',

data: {{ temperature | tojson }}

}

]

},

options: {

responsive: true,

scales: {

x: {

beginAtZero: true

},

y: {

beginAtZero: true

}

}

}

});

</script>

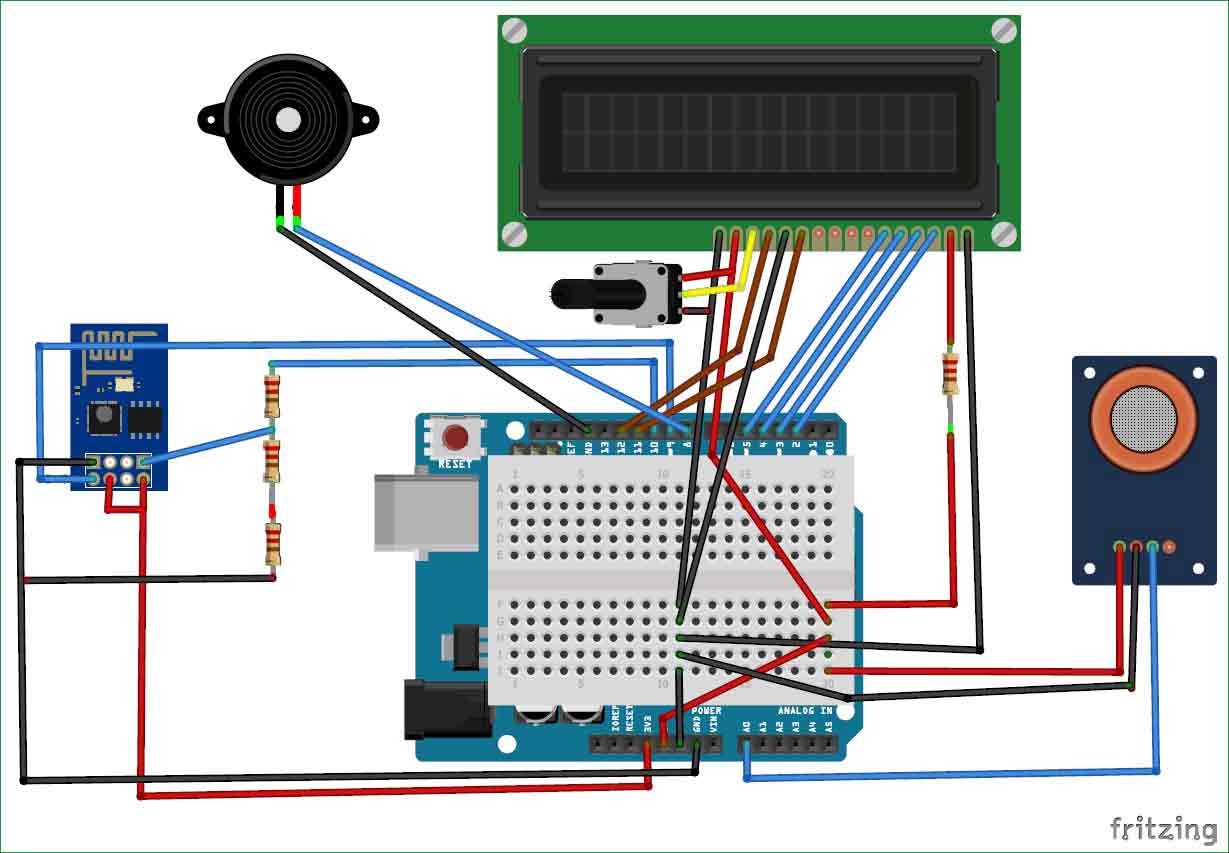
</body>

</html>

Diagrams and Screenshots:

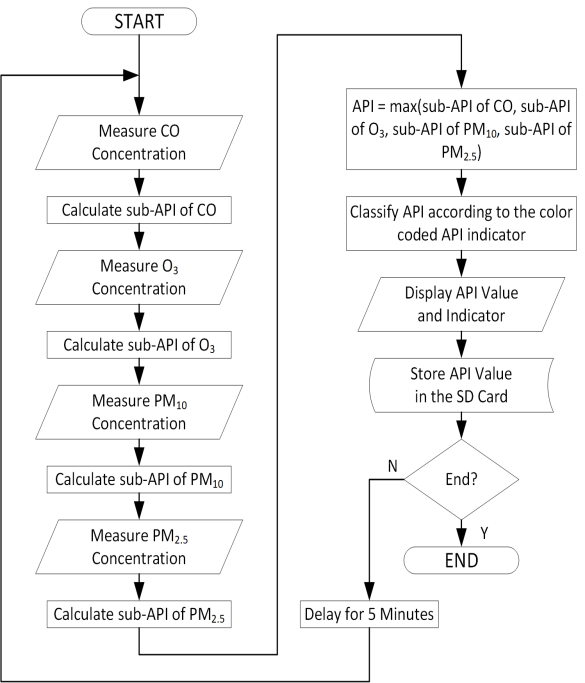
IoT Device Setup Diagram:

A schematic of the IoT device setup.



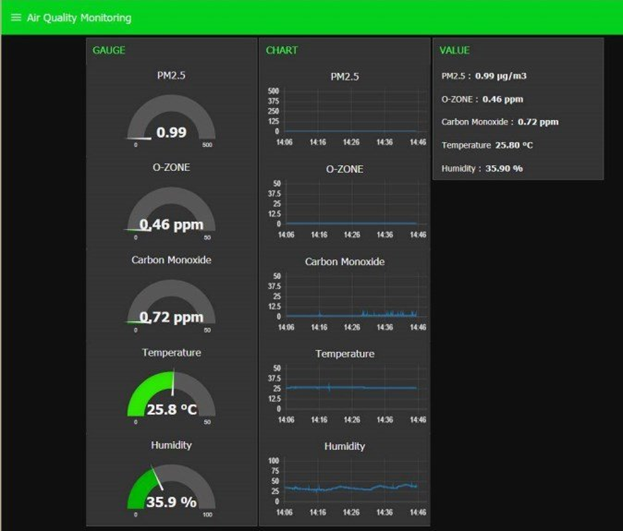
Data Flow Diagram:

Illustrating the flow of data from sensors to the web application.



Web Application Screenshots :

Screenshots of the web dashboard displaying air quality data.



Real-time Awareness:

The real-time air quality monitoring system can raise public awareness in the following ways:

1. Access to Data:

The public can access real-time air quality data from multiple locations, allowing them to make informed decisions regarding outdoor activities.

1. Health Impact:

The system can provide information about the health impacts of poor air quality, encouraging people to take precautions.

1. Environmental Awareness:

Users can see the effect of pollution on the environment, encouraging environmentally responsible behavior.

Project Replication Instructions

1. IoT Device Setup:

* Assemble the required sensors and a microcontroller.
* Install the necessary Python libraries for sensor data collection.

- Configure the device to connect to Wi-Fi or a cellular network.

- Replace simulated data with real sensor data collection logic.

2. Data-sharing Platform Development:

- Set up a cloud server .

- Create a database for storing air quality data.

- Develop a web application for data visualization using Python web frameworks .

- Modify the database configuration to use your chosen database system.

3. Integration:

- Modify the IoT device code to send data to the cloud server.

- Configure the cloud server to handle incoming data and store it in the database.

- Integrate the web application with the database to display real-time data.

- Customize the web application to suit your project requirements.

1. Example Outputs:

