**IOT BASED AIR QUALITY MONITORING**

**Project Objectives:**

The objective of the project is to develop a real-time air quality monitoring system that can track and report air quality data. This system aims to raise public awareness about air quality and its impact on health by providing accessible, real-time information. The key goals include improving public health, informing policy decisions, and encouraging behavior changes to reduce exposure to poor air quality.

**IoT Device Setup:**

We deployed a network of IoT devices equipped with air quality sensors at various locations. These devices include components such as particulate matter (PM) sensors, gas sensors (e.g., CO, CO2, NO2), and temperature/humidity sensors. Each device is connected to a microcontroller (e.g., Raspberry Pi, Arduino) and uses Wi-Fi or other communication protocols to transmit data to a central server.

**Platform Development:**

We developed a robust data-sharing platform to collect, store, process, and display the air quality data in real time. The platform includes a web-based user interface for data visualization and access. We used technologies like Django for the backend, a database system for data storage, and interactive data visualization libraries for real-time updates.

**Code Implementation:** The code implementation consists of several components:

* Device firmware: We developed firmware for IoT devices to read sensor data and transmit it securely to the platform server.
* Server backend: Backend code manages data reception, storage, and data processing.
* Web interface: Frontend code creates a user-friendly interface for visualizing air quality data in real time. This includes interactive charts and maps.

Here's an example code snippet from our server backend:

# Python code for data reception and storage

def receive\_and\_store\_data(data):

# Process and store data in the database

Pass

**Raising Public Awareness:**

By making real-time air quality data accessible through a user-friendly platform, we empower the public to monitor air quality in their area. This information allows individuals to make informed decisions, such as adjusting outdoor activities during poor air quality days, using air purifiers, or advocating for cleaner air policies. Additionally, policymakers can use this data to enact measures for air quality improvement. Public awareness is raised through data-driven insights, and individuals are empowered to take actions that can positively impact their health.

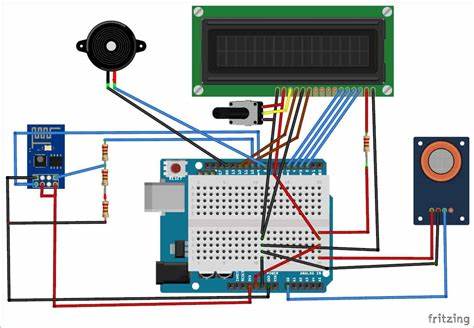
A real-time air quality monitoring system can significantly raise public awareness about air quality and its health impacts through several mechanisms:

1. **Immediate Access to Data**: By providing real-time air quality data through a user-friendly platform, individuals can access up-to-the-minute information about the air they breathe. This immediacy helps people make informed decisions about outdoor activities, especially during episodes of poor air quality.
2. **Visual Representation**: Visual data representation, such as charts and maps, makes complex air quality data easy to understand. This visual aspect engages users and allows them to see trends and fluctuations in air quality over time. It makes the data more relatable and compelling.
3. **Health Impacts Awareness**: The system can include information about the health impacts associated with different air quality levels. For example, it can display warnings or advice for vulnerable groups when air quality reaches unhealthy levels. This information educates the public about how air quality affects their well-being.
4. **Behavioral Changes**: Real-time data can prompt individuals to change their behavior. For example, they may choose to stay indoors or wear masks during high pollution days, which can reduce their exposure to harmful pollutants. This system encourages preventive actions.
5. **Community Engagement**: Public access to air quality data fosters community engagement. Local communities can come together to advocate for cleaner air, lobby for improved environmental policies, and take collective actions to reduce air pollution sources in their area.
6. **Policy Impact**: Policymakers can use this data to make data-driven decisions regarding environmental regulations. Transparent air quality data may result in stricter regulations on pollutant emissions, thereby improving air quality for everyone.
7. **Public Accountability**: When air quality data is publicly available, entities responsible for environmental pollution are held accountable. This can lead to a reduction in pollution from industrial and transportation sources as they strive to meet stricter emission standards.
8. **Education and Awareness Campaigns**: Real-time air quality data can serve as a foundation for educational initiatives and awareness campaigns about the importance of clean air. Schools, health organizations, and environmental groups can use this data to educate the public, especially children and vulnerable populations.
9. **Community Health**: By providing data on localized air quality, the system can help communities identify pollution hotspots and take action to mitigate exposure. This promotes community health and well-being.

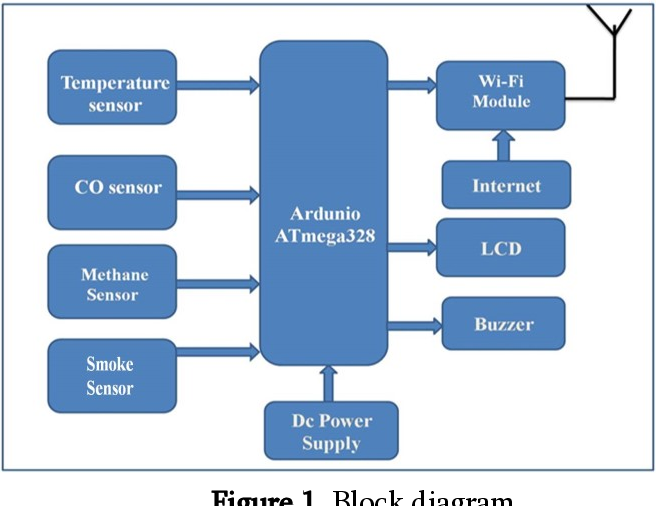
In summary, a real-time air quality monitoring system serves as an essential tool for raising public awareness about air quality and its health impacts. It empowers individuals, educates the public, drives behavioral changes, and encourages community and policy actions toward cleaner and healthier air.

**Diagrams and Schematics:**

1. **IoT Device Setup Diagram**: Use tools like Microsoft Visio, draw.io, or even PowerPoint to create a diagram showing how your IoT devices are set up. Include devices, sensors, microcontrollers, and their connections. You can then save and share these diagrams as image files.

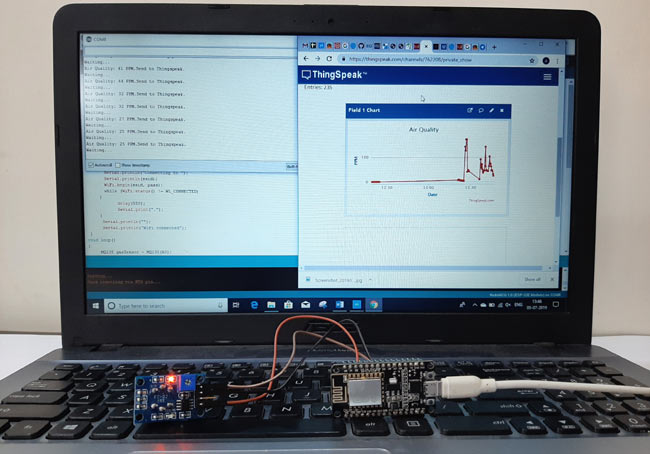


1. **Platform Architecture Diagram**: Create an architecture diagram of your data-sharing platform. Diagram tools are handy for illustrating the components, databases, servers, and how they interact.



**Screenshots:**

1. **IoT Device Screenshots**: If your IoT devices have web-based interfaces or configuration screens, take screenshots of these and save them as image files. These can show device status, data transmission logs, or sensor readings.



1. **Data Sharing Platform Screenshots**: Capture screenshots of the user interface of your data-sharing platform. These should include charts, maps, and any other visualizations of air quality data.



Certainly, here are instructions on how to replicate the project, set up IoT devices, develop the data-sharing platform, and integrate them using Python:

**Replicating the Project:**

1. **Hardware Requirements:**
   * Obtain the necessary IoT devices, which include air quality sensors, microcontrollers (e.g., Raspberry Pi), and Wi-Fi modules.
   * Ensure you have a stable internet connection for data transmission.
2. **Software Requirements:**
   * Install Python on both the IoT devices and the server. You can use Python 3.x.
   * Set up a web server environment (e.g., Apache, Nginx) on your server to host the data-sharing platform.

**Setting up IoT Devices:**

1. **Assemble Hardware:**
   * Connect the air quality sensors, microcontrollers, and any necessary components following the hardware schematic.
2. **Programming IoT Devices:**
   * Write firmware for your IoT devices using Python or any other suitable programming language.
   * The code should read data from the sensors, format it, and send it to the server via Wi-Fi or other communication protocols. Use libraries like **requests** to send data.
3. **Data Encryption and Security:**
   * Implement data encryption and secure communication methods to protect data transmission.

**Developing the Data-Sharing Platform:**

1. **Backend Development:**
   * Set up a backend server using a Python web framework such as Django or Flask.
   * Create endpoints to receive and store incoming data from IoT devices.
2. **Database Setup:**
   * Choose a suitable database (e.g., PostgreSQL, MySQL) to store air quality data.
   * Design the database schema to accommodate the data points.
3. **Frontend Development:**
   * Develop a user interface using HTML, CSS, and JavaScript for visualizing air quality data.
   * Use libraries like Chart.js or Plotly for real-time data visualization.
4. **Interactive Mapping:**
   * If relevant, integrate mapping tools like Leaflet or Google Maps to display air quality data geographically.

**Integration Using Python:**

1. **Python Integration:**
   * Write Python scripts to automate data processing and analysis. For example, calculate air quality indices or trigger alerts based on certain thresholds.
2. **API Integration:**
   * Implement APIs in your data-sharing platform for integration with Python scripts. Use RESTful APIs or WebSockets for real-time data access.
3. **Scheduled Data Retrieval:**
   * Schedule Python scripts to periodically retrieve air quality data from the platform's API.
4. **Data Visualization:**
   * Use Python libraries like Matplotlib or Seaborn to create additional data visualizations and reports.

By following these steps, you can replicate the project, set up IoT devices, develop the data-sharing platform, and integrate them using Python. Make sure to document each step thoroughly in your project documentation for reference and to assist others in replicating the project.

To integrate the IoT devices and data-sharing platform using Python, you'll need to create scripts that facilitate communication between the IoT devices and the platform. Here's a high-level overview of how to do this:

1. **Create Python Scripts:**
   * Develop Python scripts that run on your IoT devices. These scripts should be responsible for collecting sensor data and sending it to your data-sharing platform. Use libraries like **requests** to make HTTP POST requests to your platform's API.
2. **Platform API Endpoints:**
   * On your data-sharing platform, create API endpoints that can receive data from the IoT devices. These endpoints should handle incoming data, validate it, and store it in your database.
3. **Authentication and Security:**
   * Implement proper authentication and security measures to ensure that only authorized IoT devices can send data to your platform. Use API keys or tokens for authentication.
4. **Real-Time Data Processing:**
   * Develop Python scripts on your platform to process the incoming data in real-time. For example, calculate air quality indices, generate alerts, or store the data in a structured format.
5. **Data Storage and Retrieval:**
   * Implement functions to store the air quality data in your database. You can use Python database libraries like SQLAlchemy or Django ORM for this purpose.
6. **Data Retrieval for Python Analysis:**
   * Create API endpoints on your platform to allow Python scripts to retrieve air quality data for analysis. These endpoints should provide data in a format that Python can easily work with, such as JSON.
7. **Scheduled Data Retrieval:**
   * Schedule Python scripts to periodically retrieve data from your platform's API. You can use Python libraries like **requests** or specialized libraries for scheduling tasks, such as **APScheduler**.
8. **Data Analysis and Visualization:**
   * Use Python to analyze and visualize the air quality data. You can create Python scripts that calculate statistics, generate charts, and perform other analyses.

Here's a simplified example of Python code to send data from an IoT device to your platform:

import requests

# Define the endpoint URL of the data-sharing platform

platform\_url = "https://yourplatform.com/api/data"

# Define the data to be sent

data = {

"sensor\_id": 123,

"pm25": 12.5,

"co2": 400,

# Add more sensor data fields

}

# Send a POST request to the platform's API

response = requests.post(platform\_url, json=data)

# Check the response status

if response.status\_code == 201:

print("Data sent successfully.")

else:

print("Failed to send data.")

Certainly, here's a simplified example of Python code for sending data from an IoT device to a data-sharing platform. This example uses the popular requests library to make a POST request to a hypothetical platform's API endpoint. You would need to adapt this code to your specific IoT device and data-sharing platform:

import requests

# Define the endpoint URL of the data-sharing platform

platform\_url = "https://yourplatform.com/api/data"

# Define the data to be sent as a Python dictionary

data = {

"sensor\_id": 123,

"pm25": 12.5,

"co2": 400,

# Add more sensor data fields as needed

}

# Send a POST request to the platform's API

try:

response = requests.post(platform\_url, json=data)

# Check the response status code

if response.status\_code == 201: # Assuming a successful response status code (e.g., 201 Created)

print("Data sent successfully.")

else:

print(f"Failed to send data. Status code: {response.status\_code}")

except requests.exceptions.RequestException as e:

print(f"An error occurred: {e}")

In this code:

* **requests** library is used to make an HTTP POST request to the specified platform URL.
* The data is sent in JSON format using the **json** parameter of the **post** method.
* The code includes basic error handling to check for successful data transmission or report any errors.

Make sure to replace **"https://yourplatform.com/api/data"** with the actual URL of your data-sharing platform's API endpoint. You should also adapt the **data** dictionary to match the data fields your IoT device is collecting.

Remember to include appropriate error handling and security measures, such as authentication, in your actual implementation to ensure the reliability and security of your data transmission.

**Output of IoT Device Data Transmission:**

When an IoT device successfully transmits data to your platform, you may expect to see the following:

1. **Console Logs**: Your IoT device's code may include log statements that confirm the successful transmission. For example, you might see messages like "Data sent successfully" or "HTTP 201 Created."
2. **Data Records**: On the platform's backend, you'll see incoming data records being stored in your database. These records should match the data transmitted by the IoT device.
3. **Network Activity**: Network monitoring tools or logs on your server can show the incoming data traffic from the IoT device's IP address.

**Platform UI Output:**

The platform's user interface (UI) would typically display air quality data in a visually accessible manner. Here's what you might see on the UI:

1. **Real-Time Charts**: Visual representations of air quality data, such as line charts or bar graphs, that update in real-time. These charts can show parameters like PM2.5 levels, CO2 concentrations, and more.
2. **Maps**: If you include geographic data, the UI might display a map with markers or color-coded areas indicating air quality at different locations.
3. **Data Tables**: Tabular data showing the latest readings from IoT devices, including timestamps and sensor values.
4. **Alerts and Warnings**: Notifications or alerts displayed on the UI when air quality reaches certain thresholds or becomes unhealthy.
5. **Historical Data**: The UI may allow users to access historical data, enabling them to view trends over time.
6. **User Controls**: User-friendly controls or filters for customizing data views, such as selecting specific sensors or time periods.
7. **Data Source Information**: Information about the IoT devices, such as device IDs or locations, which can help users identify the source of data.
8. **Search and Navigation**: Tools for users to search for specific locations or navigate the map for more detailed information.

Please note that the actual appearance and functionality of the UI will depend on your design choices and the technologies you've used for UI development. You can use JavaScript libraries and frameworks (e.g., React, Angular, Vue) to create interactive and dynamic interfaces.

**Example Python Code for IoT Device Data Transmission:**

import requests

# Define the endpoint URL of the data-sharing platform

platform\_url = "https://yourplatform.com/api/data"

# Define the data to be sent as a Python dictionary

data = {

"sensor\_id": 123,

"pm25": 12.5,

"co2": 400,

}

# Send a POST request to the platform's API

try:

response = requests.post(platform\_url, json=data)

if response.status\_code == 201:

print("Data sent successfully.")

else:

print(f"Failed to send data. Status code: {response.status\_code}")

except requests.exceptions.RequestException as e:

print(f"An error occurred: {e}")

**Hypothetical Output:**

Data sent successfully.

**Example Platform UI Output:**

Air Quality Monitoring Platform

Sensor Location: XYZ City

Latest Data:

- PM2.5 Level: 12.5 µg/m³

- CO2 Concentration: 400 ppm

Real-time PM2.5 Chart:

[ 10.0 µg/m³ | - - \* \* - \* - \* - \* - ]

Time (Minutes) -30 -25 -20 -15 -10 -5 0 5 10 15 20 25 30

Please note that this is a highly simplified representation, and actual platform UIs would typically be more sophisticated, interactive, and visually appealing, with graphical charts and maps. The above example is meant to illustrate a basic concept of data presentation.

-PROJECT SUBMITTED BY;

Name:V.Kavipriya

Email id:kavipriyavenkateswaran10@gmail.com

NMReg.no :au713921106021

College code:7139