**Air Quality Monitoring**

**Hardware Setup:**

* Choose appropriate air quality sensors (e.g., MQ series sensors) and connect them to a microcontroller. For example, use an Arduino board to collect sensor data.
* Set up a Wi-Fi module (e.g., ESP8266) to enable the microcontroller to connect to the internet.

**Sensor Data Collection:**

* Program the microcontroller to read sensor data periodically (e.g., every few seconds).
* Use relevant libraries and code to calibrate and interpret the sensor data.

**Data Transmission:**

* Use the microcontroller to transmit sensor data to a web server. You can implement this using HTTP or MQTT protocols.

**Web Development:**

* Create a web-based dashboard using HTML, CSS, and JavaScript to display air quality data.
* Set up a web server or use a cloud platform (e.g., AWS, Azure) to host your dashboard.

**Backend Server (Optional):**

* You can create a backend server using technologies like Node.js, Python (Flask/Django), or any other server-side language to handle data reception and storage.

**Data Visualization:**

* Utilize charting libraries like Chart.js or D3.js to present air quality data in graphs or charts.
* Display real-time data and historical data with timestamps.

**User Interface:**

* Create a user-friendly interface with interactive features for users to view data, set alerts, and customize the dashboard.

**Alerting System (Optional):**

* Implement an alert system that notifies users when air quality reaches predefined thresholds. This can be done using email, SMS, or push notifications.

**Data Storage:**

* Store historical data in a database, which can be SQL-based (e.g., MySQL) or NoSQL (e.g., MongoDB).

**User Authentication (Optional):**

* Implement user authentication if multiple users need access to the dashboard.

**Remote Access:**

Make sure the dashboard is accessible over the internet, so users can monitor air quality from anywhere.

**Documentation and Presentation:**

* Document the project, including the hardware setup, software code, and how to use the dashboard.
* Prepare a presentation to showcase the project's features and functionality.

**Testing and Calibration:**

* Test the system thoroughly and calibrate the sensors to ensure accurate readings.

**Deployment:**

* Deploy the entire system, including hardware and web dashboard, in a real-world environment for monitoring air quality.

**Creating a mobile app for an IoT Air Quality Monitoring project:**

It can be an excellent way to provide users with a convenient interface to access real-time air quality data. You can develop a mobile app using Python by leveraging frameworks like Kivy, which allows you to build cross-platform mobile applications. Here's a basic outline of how to create a simple mobile app for your project using Kivy:

**Prerequisites:**

* Python installed on your development machine.
* Install Kivy by running pip install kivy.

**Creating the Mobile App:**

**Set Up the Project Structure:** Create a project folder and organize your files. You may have separate folders for app code, assets (such as images and fonts), and any other resources.

**Design the User Interface:** Use Kivy's KV language or Python code to design the app's user interface. In your KV file (e.g., app.kv), define the layout, buttons, labels, and any other widgets needed to display air quality data.

#:kivy 2.0.0

BoxLayout:

orientation: 'vertical'

Label:

text: 'Air Quality Monitoring'

Label:

id: air\_quality\_label

text: 'Air Quality: N/A'

Button:

text: 'Refresh'

on\_press: app.refresh\_data()

**create a python code:**

Write the Python code for the mobile app using Kivy. This code will handle data retrieval and update the user interface.

from kivy.app import App

from kivy.uix.boxlayout import BoxLayout

from kivy.uix.label import Label

class AirQualityApp(App):

def build(self):

self.layout = BoxLayout(orientation='vertical')

self.air\_quality\_label = Label(text='Air Quality: N/A')

self.layout.add\_widget(self.air\_quality\_label)

return self.layout

def refresh\_data(self):

# Retrieve air quality data from your IoT system.

air\_quality\_data = self.retrieve\_air\_quality\_data()

self.air\_quality\_label.text = f'Air Quality: {air\_quality\_data}'

def retrieve\_air\_quality\_data(self):

# Implement logic to fetch data from your IoT system.

# You may use REST APIs or other data retrieval methods here.

# Replace this with actual data retrieval code.

# For demonstration purposes, a mock value is returned.

return 'PM2.5 25 µg/m³, PM10 40 µg/m³'

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

AirQualityApp().run()

**Connect to IoT Data:** In the retrieve\_air\_quality\_data method, implement the logic to fetch data from your IoT system. You may need to use APIs or data transmission protocols you set up earlier.

**Testing:** Run the app by executing the Python script. The app should display the air quality data and allow users to refresh it.

**Enhancements:** Depending on your project's complexity, you can add features like user authentication, historical data display, and alerts within the mobile app.

**Build and Deploy:** Once the app is complete, you can package it for the target mobile platforms (iOS and Android) using Kivy tools and then deploy it to app stores if needed.

To connect your mobile app with your IoT Air Quality Monitoring project, you'll need to set up communication between the mobile app and the IoT system. The most common method for this is using APIs (Application Programming Interfaces). Here's a high-level overview of how to establish this connection:

**Set Up an API for Data Retrieval:**

You need to create an API on the server or IoT device to provide access to the air quality data. You can implement this using various technologies, such as:

RESTful APIs: These are easy to implement and widely used for IoT projects. You'll expose endpoints (URLs) that the mobile app can query to retrieve data.

WebSocket: For real-time updates, you can implement a WebSocket API to push data to the app whenever there's a new reading.

**API Authentication and Security:**

Depending on your project's requirements, you may need to implement API authentication and security measures. This ensures that only authorized users can access the data.

**Mobile App Integration:**

In your mobile app (Python-based using Kivy or any other technology), you can use libraries like requests to send HTTP requests to the API endpoints. Here's an example of how you can retrieve data from a RESTful API in Kivy:

**python code:**

import requests

from kivy.app import App

from kivy.uix.label import Label

class AirQualityApp(App):

def build(self):

self.layout = BoxLayout(orientation='vertical')

self.air\_quality\_label = Label(text='Air Quality: N/A')

self.layout.add\_widget(self.air\_quality\_label)

self.refresh\_data() # Automatically fetch data on app startup

return self.layout

def refresh\_data(self):

try:

response = requests.get('https://your-api-url/data')

if response.status\_code == 200:

air\_quality\_data = response.json() # Assuming the API returns JSON data

self.air\_quality\_label.text = f'Air Quality: {air\_quality\_data}'

else:

self.air\_quality\_label.text = 'Error fetching data'

except requests.exceptions.RequestException:

self.air\_quality\_label.text = 'Network error'

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

AirQualityApp().run()

Tse**ting and Debugging:**

Test the mobile app to ensure it can successfully retrieve and display the air quality data. Debug any issues that may arise during testing.

**Real-Time Updates (Optional):**

If your IoT system supports real-time data updates, consider implementing a WebSocket API or server-sent events (SSE) to push updates to the mobile app without manual refresh.

**Handle Errors and Edge Cases:**

Implement error handling and edge case scenarios, such as when the mobile device is not connected to the internet, or the API server is down.

**Deployment:**· Once everything works as expected, package and deploy your mobile app to the target platforms (iOS and Android). Follow the guidelines for your chosen mobile app deployment method (e.g., Google Play Store for Android apps).

PYTHON PROGRAM FOR WOKWI:

import machine

import network

import urequests as requests # This is a MicroPython library for HTTP requests

ssid = "Your\_SSID"

password = "Your\_PASSWORD"

api\_key = "Your\_ThingSpeak\_API\_Key"

server = "api.thingspeak.com"

http\_port = 80

air\_quality\_pin = machine.ADC(0)

# Connect to Wi-Fi

sta\_if = network.WLAN(network.STA\_IF)

sta\_if.active(True)

sta\_if.connect(ssid, password)

while not sta\_if.isconnected():

pass

print("Connected to WiFi")

def send\_data\_to\_thingspeak(data):

url = f"/update?field1={data}&api\_key={api\_key}"

response = requests.get(f"http://{server}:{http\_port}{url}")

if response.status\_code == 200:

print("Data sent to ThingSpeak")

else:

print("Failed to send data to ThingSpeak")

while True:

t = air\_quality\_pin.read()

print("Air Quality =", t)

if t <= 500:

print("Fresh Air")

elif 500 < t <= 1000:

print("Poor Air")

else:

print("Very Poor")

# Send data to ThingSpeak

send\_data\_to\_thingspeak(t)

machine.delay(10000) # Delay between readings

CIRCUIT DIAGRAM: