IOT PROJECT—PHASE 3

PROJECT NAME—AIR QUALITY MONITORING

To start building your IOT air quality monitoring system, you can follow these steps:

Choose IOT Devices: Select appropriate sensors for measuring air quality parameters like pollution levels and particulate matter. Common sensors for this purpose include gas sensors, dust sensors, and temperature/humidity sensors.

Hardware Setup: Connect the selected sensors to your IOT device (e.g., Raspberry Pi, Arduino, or a specialized IoT board). Ensure that the connections are correctly established and the sensors are powered.

Install Necessary Software: Set up the development environment on your IoT device. This typically includes installing Python and any libraries required to interface with the sensors. For instance, you might use libraries like Adafruit CircuitPython for sensor integration.

Write Python Script: Develop a Python script on the IOT device to collect data from the sensors. This script should read sensor values at regular intervals and store them in variables.

Data Transmission: Use IOT protocols like MQTT, HTTP, or WebSocket to send the collected data to your chosen data-sharing platform. This could be a cloud service or a local server. Make sure you have the necessary credentials and configurations for the data transfer.

Data Validation and Quality Checks: Implement data validation and quality checks in your Python script to ensure the collected data is accurate and within expected ranges. You can set up threshold values for alerts.

Error Handling: Implement error-handling mechanisms in your script to manage any unexpected issues, such as network outages.

Data Visualization: You can consider building a dashboard or a data visualization interface to monitor the air quality data in real-time. There are various tools and platforms available for this purpose.

Testing and Calibration: Test your system thoroughly to ensure the accuracy and reliability of data. Calibration may be necessary for some sensors to ensure accurate readings.

Security: Ensure that your data transmission is secure. Use encryption and authentication mechanisms to protect your data during transmission.

Scaling and Expansion: As your project progresses, you may want to add more sensors, improve the data storage and analysis capabilities, and possibly integrate machine learning for predictive analytics.

Remember that building an IoT system can be a complex process, so it’s essential to plan and document each step carefully. Also, keep in mind any power requirements and data usage costs, especially if you’re using a cloud-based data-sharing platform.

Developing a Python script for an IoT device to send collected data to a data sharing platform typically involves using communication protocols and APIs relevant to the platform you’re sending data to. Below is a general example of how you can create a Python script to send data from an IoT device to a data sharing platform. Please note that the specifics will depend on the platform you’re using.

Assuming you want to send data using HTTP (RESTful API) to a generic data sharing platform, you can use libraries like requests to make HTTP requests. Here’s an example script:

Import requests

Import json

# Define the data to send

Data\_to\_send = {

“sensor\_id”: “12345”,

“temperature”: 25.5,

“humidity”: 50.2,

“timestamp”: “2023-10-18T14:30:00Z”

}

# Define the API endpoint and authentication (if required)

Api\_url = <https://yourdataplatform.com/api/endpoint>

Api\_key = “your\_api\_key” # Replace with your API key

# Set headers (if required)

Headers = {

“Content-Type”: “application/json”,

“Authorization”: f”Bearer {api\_key}”

}

# Convert data to JSON

Data\_json = json.dumps(data\_to\_send)

# Send the data to the platform

Try:

Response = requests.post(api\_url, data=data\_json, headers=headers)

# Check the response

If response.status\_code == 200:

Print(“Data sent successfully!”)

Else:

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

Except Exception as e:

Print(f”An error occurred: {str€}”)

In this script:

Replace data\_to\_send with the data collected by your IoT device.

Set api\_url to the API endpoint provided by your data sharing platform.

If your platform requires authentication, set api\_key and provide the necessary headers.

Convert the data to JSON format using json.dumps().

Send the data to the platform using the requests.post() method.

Make sure to install the requests library if you haven’t already by running pip install requests.

Replacing data\_payload with the actual data collected by your IoT device is a critical step in the Python script. This is where you format the sensor data into a structured JSON object that can be sent to your data-sharing platform’s API. Here’s how you can do it:

Read Sensor Data: Use the appropriate code to read data from your sensors. The specific code to read sensor data depends on the type of sensors you’re using and the libraries or modules available for them. Make sure that you’ve correctly initialized and configured your sensors earlier in the script.

Format Data Payload: Create a JSON object (dictionary in Python) that contains the sensor data you want to send. This should match the data format expected by your data-sharing platform. For example, if you have sensors for pollution levels and particulate matter, your data payload might look like this:

Python

Copy code

Sensor\_data = {

“timestamp”: time.time(),

“pm2.5”: sensor\_readings\_pm25, # Replace with actual PM2.5 data

“co2”: sensor\_readings\_co2, # Replace with actual CO2 data

# Add other sensor data as needed

}

Ensure that you replace “sensor\_readings\_pm25” and “sensor\_readings\_co2” with the actual variables or values obtained from your sensors. The “timestamp” should represent the time when the data was collected.

Add Additional Sensor Data: If you have more sensors measuring other parameters (e.g., temperature, humidity, GPS coordinates), add them to the sensor\_data dictionary as necessary.

Send Data: Finally, send the sensor\_data to your data-sharing platform’s API using the requests.post method, as shown in the previous example.

Here's a more concrete example of how to replace data\_payload with actual sensor data:

Python

Copy code

# Read sensor data

Sensor\_readings\_pm25 = sensor.read\_pm25() # Replace with actual reading code

Sensor\_readings\_co2 = sensor.read\_co2() # Replace with actual reading code

# Format data payload

Sensor\_data = {

“timestamp”: time.time(),

“pm2.5”: sensor\_readings\_pm25,

“co2”: sensor\_readings\_co2,

# Add other sensor data as needed

}

Make sure that you’ve correctly read and formatted data from your sensors to match the requirements of your data-sharing platform. The exact structure and content of sensor\_data may vary depending on the platform’s API specifications, so refer to their documentation for any specific formatting requirements.

Setting the api\_url variable to the correct API endpoint is a crucial step in configuring your IoT air quality monitoring system. The api\_url variable should point to the URL where your data-sharing platform’s API is hosted. Below are the details you need to consider when setting this variable:

API Endpoint URL: You will need to obtain the API endpoint URL from your data-sharing platform provider. This URL is typically provided when you set up an account or register your IoT project on the platform. It should look something like:

Arduino

Copy code

<https://api.example.com/your-data-endpoint>

Replace <https://api.example.com/your-data-endpoint> with the actual URL provided by your data-sharing platform.

Authentication: If your data-sharing platform requires authentication, you’ll need to include this information in the request headers. Common authentication methods include using API keys, OAuth tokens, or other forms of authentication. Make sure to set up the authentication headers in the script correctly, as shown in the previous example. Replace “Bearer YourAccessToken” with your actual authentication token.

Here's how you can set api\_url in your Python script with the correct API endpoint:

Python

Copy code

Api\_url = <https://api.example.com/your-data-endpoint> # Replace with your actual API endpoint URL

Remember that this URL is specific to the data-sharing platform you’re using, and it should be provided by the platform’s documentation

Convert the data to JSON format using json.dumps().

json.dumps() is a Python function that converts data into a JSON (JavaScript Object Notation) format. It’s often used to serialize data, making it easy to store or transmit. To use it, you'll need to import the json module in Python. Here's a basic example:

Python

Copy code

Import json

Data = {

“name”: “John”,

“age”: 30,

“city”: “New York”

}

Json\_data = json.dumps(data)

In this example, the data dictionary is converted into a JSON formatted string and stored in the json\_data variable. This string can then be saved to a file, sent over a network, or used in various ways to work with structured data.

Send the data to the platform using the requests.post() method.

To send data to a platform using the requests.post() method in Python, you’ll need to provide the necessary information and parameters in the request. Here are the details:

Import the requests library:

Make sure you have the requests library installed. If not, you can install it using pip:

Python

Copy code

Pip install requests

Import the library in your Python script:

Python

Copy code

Import requests

Define the URL of the platform you want to send data to:

Python

Copy code

url = ‘https://your-platform-url.com/api/endpoint’

Prepare the data you want to send. This can be in JSON format, form data, or other suitable formats:

For JSON data:

Python

Copy code

Data = {

‘key1’: ‘value1’,

‘key2’: ‘value2’,

}

For form data:

Python

Copy code

Data = {

‘param1’: ‘value1’,

‘param2’: ‘value2’,

}

Send the POST request with the data:

Python

Copy code

Response = requests.post(url, data=data)

Check the response from the platform:

Python

Copy code

If response.status\_code == 200:

# Request was successful

Print(‘Data sent successfully’)

Else:

# Request failed

Print(‘Failed to send data. Status code:’, response.status\_code)

Remember to replace ‘https://your-platform-url.com/api/endpoint’ with the actual URL of the platform you want to send data to and adapt the data format to what the platform expects. Additionally, you may need to provide headers, authentication, or other specific parameters depending on the platform’s API documentation.

PROJECT BY

R.SWETHA

21EC144

923321106044

ECE 3RD YEAR