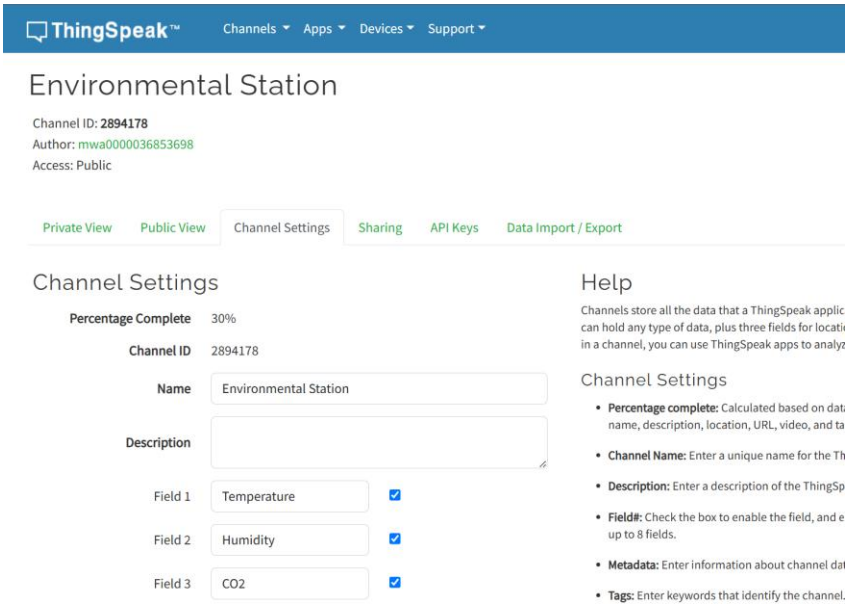
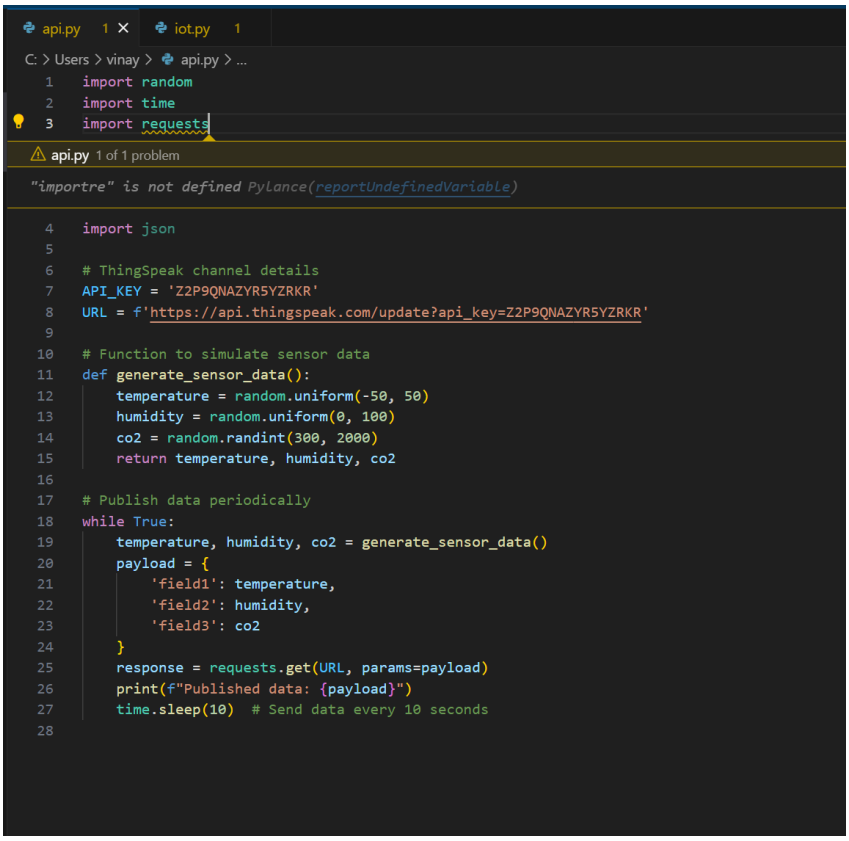
Vinay Menon	IoT Assignment 3	792225320

Created a Channel with 3fields: Temperature, Humidity, CO2. Noted down WRITE_API_KEY, channel ID, username etc.

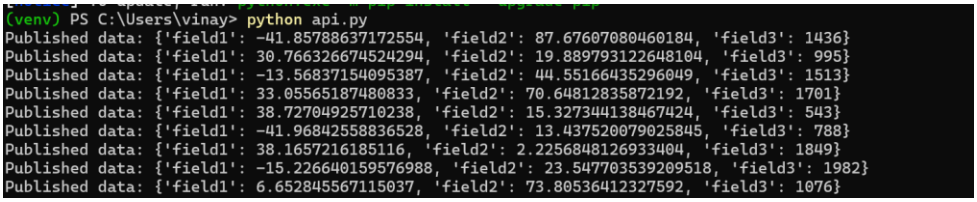


This code sends random values based on sensor field and value limits .



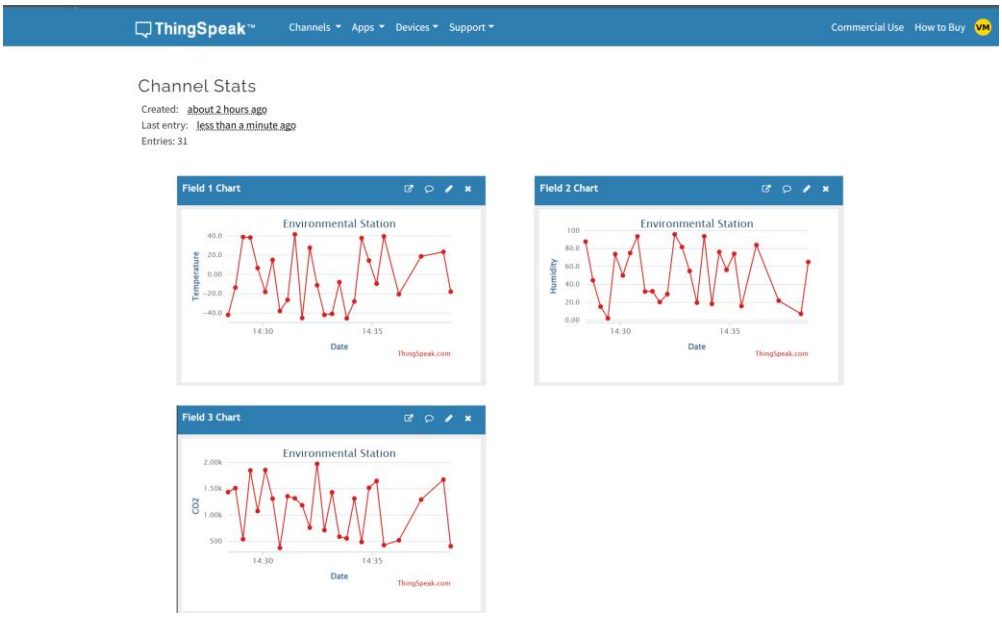
```
api.py 1 ×  iot.py 1
C: > Users > vinay > api.py > ...
1  import random
2  import time
3  import requests
api.py 1 of 1 problem
"importre" is not defined Pylance(reportUndefinedVariable)
4  import json
5
6  # ThingSpeak channel details
7  API_KEY = 'Z2P9QNAZYR5YZRKR'
8  URL = f'https://api.thingspeak.com/update?api_key=Z2P9QNAZYR5YZRKR'
9
10 # Function to simulate sensor data
11 def generate_sensor_data():
12     temperature = random.uniform(-50, 50)
13     humidity = random.uniform(0, 100)
14     co2 = random.randint(300, 2000)
15     return temperature, humidity, co2
16
17 # Publish data periodically
18 while True:
19     temperature, humidity, co2 = generate_sensor_data()
20     payload = {
21         'field1': temperature,
22         'field2': humidity,
23         'field3': co2
24     }
25     response = requests.get(URL, params=payload)
26     print(f"Published data: {payload}")
27     time.sleep(10)  # Send data every 10 seconds
28
```

We see the code running and values being published

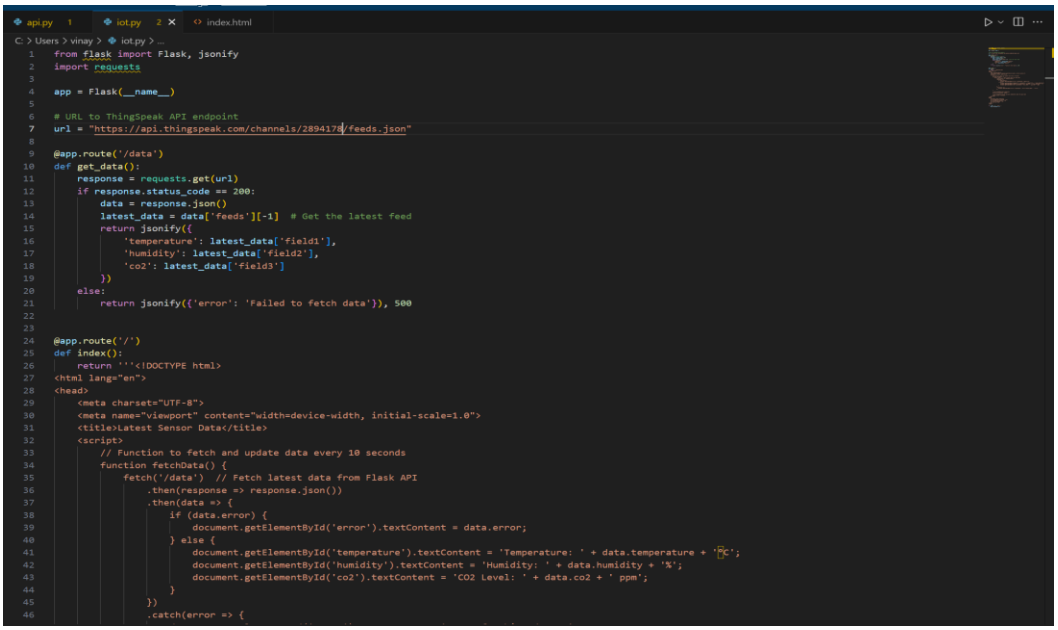


```
(venv) PS C:\Users\vinay> python api.py
Published data: {'field1': -41.85788637172554, 'field2': 87.67607080460184, 'field3': 1436}
Published data: {'field1': 30.766326674524294, 'field2': 19.889793122648104, 'field3': 995}
Published data: {'field1': -13.56837154095387, 'field2': 44.55166435296049, 'field3': 1513}
Published data: {'field1': 33.05565187480833, 'field2': 70.64812835872192, 'field3': 1701}
Published data: {'field1': 38.72704925710238, 'field2': 15.327344138467424, 'field3': 543}
Published data: {'field1': -41.96842558836528, 'field2': 13.437520079025845, 'field3': 788}
Published data: {'field1': 38.1657216185116, 'field2': 2.2256848126933404, 'field3': 1849}
Published data: {'field1': -15.226640159576988, 'field2': 23.547703539209518, 'field3': 1982}
Published data: {'field1': 6.652845567115037, 'field2': 73.80536412327592, 'field3': 1076}
```

We see the inbuilt dashboards on ThingSpeak through MQTT Server



For latest sensor values we write a new code that displays value on web server



```python
from flask import Flask, jsonify
import requests

app = Flask(__name__)

# URL to ThingSpeak API endpoint
url = "https://api.thingspeak.com/channels/2894178/feeds.json"

@app.route('/data')
def get_data():
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        latest_data = data['feeds'][-1]  # Get the latest feed
        return jsonify({
            'temperature': latest_data['field1'],
            'humidity': latest_data['field2'],
            'co2': latest_data['field3']
        })
    else:
        return jsonify({'error': 'Failed to fetch data'}), 500


@app.route('/')
def index():
    return '''<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Latest Sensor Data</title>
    <script>
        // Function to fetch and update data every 10 seconds
        function fetchData() {
            fetch('/data')  // Fetch latest data from Flask API
                .then(response => response.json())
                .then(data => {
                    if (data.error) {
                        document.getElementById('error').textContent = data.error;
                    } else {
                        document.getElementById('temperature').textContent = 'Temperature: ' + data.temperature + '°C';
                        document.getElementById('humidity').textContent = 'Humidity: ' + data.humidity + '%';
                        document.getElementById('co2').textContent = 'CO2 Level: ' + data.co2 + ' ppm';
                    }
                })
                .catch(error => {
```

```
                .catch(error => {
                    document.getElementById('error').textContent = 'Error fetching data: ' + error;
                });
        }

        // Call fetchData every 10 seconds
        setInterval(fetchData, 10000);

        // Call fetchData once to load data immediately when the page loads
        window.onload = fetchData;
    </script>
</head>
<body>
    <h1>Latest Sensor Data</h1>
    <p id="temperature">Loading...</p>
    <p id="humidity">Loading...</p>
    <p id="co2">Loading...</p>
    <p id="error"></p>
</body>
</html>'''

if __name__ == '__main__':
    app.run(debug=True)

```

After running this code we see this in console

```
(venv) PS C:\Users\vinay> python iot.py
 * Serving Flask app 'iot'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployme
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 554-885-547
127.0.0.1 - - [26/Mar/2025 15:03:12] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Mar/2025 15:03:18] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Mar/2025 15:03:28] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Mar/2025 15:03:38] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Mar/2025 15:03:48] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Mar/2025 15:03:58] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Mar/2025 15:04:08] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Mar/2025 15:04:18] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Mar/2025 15:04:28] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Mar/2025 15:04:38] "GET /data HTTP/1.1" 200 -
```

Once we run in out python virtual environment and navigate to http://127.0.0.1/5000

Latest Sensor Data

Temperature: -13.228837242452393°C

Humidity: 72.28489639563968%

CO2 Level: 1132 ppm

We see latest sensor values on webpage.

For the values of last 5 hours we make changes in code

api.py 1 iot.py 2 task3.py 2 index.html

C: > Users > vinay > task3.py > ...

```python
from flask import Flask, jsonify
import requests
from datetime import datetime, timedelta

app = Flask(__name__)

# Replace these with your ThingSpeak channel details
CHANNEL_ID = "2894178"
API_KEY = "Z2P9QNAZYR5YZRKR"

# Calculate the start and end time for the last 5 hours
def get_time_range():
    end_time = datetime.utcnow()  # Current time (UTC)
    start_time = end_time - timedelta(hours=5)  # 5 hours ago

    # Convert times to ISO 8601 format
    end_time_str = end_time.strftime('%Y-%m-%dT%H:%M:%SZ')
    start_time_str = start_time.strftime('%Y-%m-%dT%H:%M:%SZ')

    return start_time_str, end_time_str

# URL for ThingSpeak API to fetch data from the last 5 hours
def get_thingspeak_data():
    start_time, end_time = get_time_range()
    url = f"https://api.thingspeak.com/channels/{CHANNEL_ID}/feeds.json"
    params = {
        "api_key": API_KEY,
        "start": start_time,
        "end": end_time
    }

    response = requests.get(url, params=params)
    if response.status_code == 200:
        return response.json()
    else:
        return None

@app.route('/historical_data')
def get_historical_data():
    data = get_thingspeak_data()

    if data:
        feeds = data.get('feeds', [])
        if not feeds:
            return jsonify({'error': 'No data available for the last 5 hours'}), 404
```

api.py 1 iot.py 2 task3.py 2 index.html

C: > Users > vinay > task3.py > index

```python
def get_historical_data():

        # Extract and format the data for display
        result = []
        for feed in feeds:
            result.append({
                'timestamp': feed['created_at'],
                'temperature': feed['field1'],
                'humidity': feed['field2'],
                'co2': feed['field3']
            })

        return jsonify(result)
    else:
        return jsonify({'error': 'Failed to fetch data'}), 500

@app.route('/')
def index():
    return '''<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sensor Data</title>
    <script>
        function fetchData() {
            fetch('/historical_data')  // Fetch historical data
                .then(response => response.json())
                .then(data => {
                    if (data.error) {
                        document.getElementById('error').textContent = data.error;
                    } else {
                        let dataHtml = '<h2>Sensor Data from the Last 5 Hours</h2><ul>';
                        data.forEach(item => {
                            dataHtml += `<li>Time: ${item.timestamp}, Temperature: ${item.temperature}°C, Humidity: ${item.humidity}%, CO2: ${item.co2}ppm</li>`;
                        });
                        dataHtml += '</ul>';
                        document.getElementById('sensorData').innerHTML = dataHtml;
                    }
                })
                .catch(error => {
                    document.getElementById('error').textContent = 'Error fetching data: ' + error;
                });
        }
        window.onload = fetchData;  // Call fetchData when page loads
    </script>
```

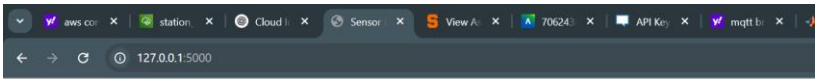
```
                });
            }
            window.onload = fetchData;  // Call fetchData when page loads
        </script>
    </head>
    <body>
        <h1>Latest Sensor Data</h1>
        <div id="sensorData">Loading...</div>
        <div id="error"></div>
    </body>
    </html>'''

    if __name__ == '__main__':
        app.run(debug=True)
```

After running we see this in console

```
(venv) PS C:\Users\vinay> python task3.py
 * Serving Flask app 'task3'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 554-885-547
127.0.0.1 - - [26/Mar/2025 15:17:30] "GET /data HTTP/1.1" 404 -
127.0.0.1 - - [26/Mar/2025 15:17:32] "GET / HTTP/1.1" 200 -
C:\Users\vinay\task3.py:13: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a
future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  end_time = datetime.utcnow()  # Current time (UTC)
127.0.0.1 - - [26/Mar/2025 15:17:32] "GET /historical_data HTTP/1.1" 200 -
```

The output of sensor value history is available on the hosted webpage



Latest Sensor Data

Sensor Data from the Last 5 Hours

- Time: 2025-03-26T18:28:25Z, Temperature: -41.85788637172554°C, Humidity: 87.67607080460184%, CO2: 1436ppm
- Time: 2025-03-26T18:28:45Z, Temperature: -13.56837154095387°C, Humidity: 44.551664352960 49%, CO2: 1513ppm
- Time: 2025-03-26T18:29:06Z, Temperature: 38.72704925710238°C, Humidity: 15.3273441384674 24%, CO2: 543ppm
- Time: 2025-03-26T18:29:26Z, Temperature: 38.1657216185116°C, Humidity: 2.22568481269334 04%, CO2: 1849ppm
- Time: 2025-03-26T18:29:46Z, Temperature: 6.652845567115037°C, Humidity: 73.805364123275 92%, CO2: 1076ppm
- Time: 2025-03-26T18:30:07Z, Temperature: -18.113191385038306°C, Humidity: 49.99135076026944%, CO2: 1855ppm
- Time: 2025-03-26T18:30:27Z, Temperature: 15.016291170642006°C, Humidity: 74.95970324306528%, CO2: 1313ppm
- Time: 2025-03-26T18:30:47Z, Temperature: -37.88007476707471°C, Humidity: 93.48765625592029%, CO2: 378ppm
- Time: 2025-03-26T18:31:08Z, Temperature: -26.366740215235986°C, Humidity: 32.034721019719 57%, CO2: 1358ppm
- Time: 2025-03-26T18:31:28Z, Temperature: 41.517929197372354°C, Humidity: 32.3705327486478 04%, CO2: 1319ppm
- Time: 2025-03-26T18:31:48Z, Temperature: -45.06946718667414°C, Humidity: 20.3004228064691 86%, CO2: 1186ppm
- Time: 2025-03-26T18:32:09Z, Temperature: 27.59393021374443°C, Humidity: 29.164337528364094%, CO2: 762ppm
- Time: 2025-03-26T18:32:29Z, Temperature: -11.169430279315243°C, Humidity: 95.83937776521 29%, CO2: 1973ppm
- Time: 2025-03-26T18:32:49Z, Temperature: -41.862575612153165°C, Humidity: 81.71137822341 81%, CO2: 715ppm
- Time: 2025-03-26T18:33:10Z, Temperature: -40.7129196448338 25°C, Humidity: 54.870851582126 59%, CO2: 1432ppm
- Time: 2025-03-26T18:33:30Z, Temperature: -7.965909217107161°C, Humidity: 19.679786476582063%, CO2: 590ppm
- Time: 2025-03-26T18:33:50Z, Temperature: -45.406666696752 97°C, Humidity: 93.609810239235 12%, CO2: 558ppm
- Time: 2025-03-26T18:34:11Z, Temperature: -28.000378416719197°C, Humidity: 18.336214210450418%, CO2: 1315ppm
- Time: 2025-03-26T18:34:31Z, Temperature: 37.5701446983218 55°C, Humidity: 76.163075334516 79%, CO2: 489ppm
- Time: 2025-03-26T18:34:51Z, Temperature: 14.376586111319 44°C, Humidity: 56.322047268702 12%, CO2: 1518ppm
- Time: 2025-03-26T18:35:12Z, Temperature: -9.533393571205885°C, Humidity: 74.131328615961 35%, CO2: 1648ppm
- Time: 2025-03-26T18:35:32Z, Temperature: 39.37687350922529°C, Humidity: 15.880909104642 95%, CO2: 430ppm
- Time: 2025-03-26T18:35:52Z, Temperature: -10.919864957096202°C, Humidity: 39.172000509787594%, CO2: 816ppm
- Time: 2025-03-26T18:36:13Z, Temperature: -20.4188865915887 13°C, Humidity: 83.945131234729 57%, CO2: 522ppm
- Time: 2025-03-26T18:36:33Z, Temperature: -3.5407791874938113°C, Humidity: 10.217267569032662%, CO2: 418ppm
- Time: 2025-03-26T18:36:53Z, Temperature: -44.37121716946145°C, Humidity: 59.512427037501 29%, CO2: 1940ppm
- Time: 2025-03-26T18:37:14Z, Temperature: 18.732357440441675°C, Humidity: 21.918433923968 65%, CO2: 1294ppm
- Time: 2025-03-26T18:37:34Z, Temperature: 20.354094853501465°C, Humidity: 19.576725397609 54%, CO2: 547ppm
- Time: 2025-03-26T18:37:54Z, Temperature: 38.739939875507 79°C, Humidity: 16.688725703421014%, CO2: 562ppm
- Time: 2025-03-26T18:38:15Z, Temperature: 23.369721300190264°C, Humidity: 7.2082633804365 52%, CO2: 1674ppm
- Time: 2025-03-26T18:38:35Z, Temperature: -17.867554245189666°C, Humidity: 64.989492702040 44%, CO2: 409ppm
- Time: 2025-03-26T18:38:55Z, Temperature: -16.498968909532 31°C, Humidity: 29.936416786706133%, CO2: 746ppm
- Time: 2025-03-26T18:39:16Z, Temperature: 43.458933437386 52°C, Humidity: 25.880823984138267%, CO2: 1377ppm
- Time: 2025-03-26T18:39:36Z, Temperature: 49.58439346990676°C, Humidity: 47.411344843963 97%, CO2: 1475ppm
- Time: 2025-03-26T18:39:56Z, Temperature: -28.424605095237688°C, Humidity: 0.32088351616630373%, CO2: 1977ppm
- Time: 2025-03-26T18:40:17Z, Temperature: 12.447740619613 2°C, Humidity: 82.728662309990 44%, CO2: 1614ppm

Brief steps to perform this experiment was:

Step 1: I created a ThingSpeak channel to collect data from the IoT sensors, such as temperature, humidity, and CO2 levels.

Step 2: Using Python, I wrote a script to send random data (simulating sensor readings) to ThingSpeak through its API. The data is sent at regular intervals, which updates the ThingSpeak channel.

Step 3: I used Flask to develop a web interface to display the data. The application retrieves the historical data from ThingSpeak using its REST API, filters it for the last 5 hours, and then displays the data on the webpage.

Step 4: I also used HTML, CSS, and JavaScript to make the page interactive and responsive.

Step 5: I tested the system by verifying that the sensor data appeared on the ThingSpeak channel and that it was correctly fetched and displayed on the web application. I ensured that the data was updated every 5 hours as per the requirements.

Reflection

While completing this assignment, I encountered several challenges, particularly when integrating the Flask web application with the ThingSpeak API. One of the major issues was correctly formatting the time range for fetching historical data. Initially, I struggled with adjusting the time parameters in the API request to retrieve data for the last 5 hours. After experimenting and learning more about Python's datetime module, I was able to compute the necessary time range. The most rewarding aspect was when the web application successfully displayed the sensor data in real-time. It was satisfying to see the combination of hardware and software working seamlessly, with the ability to update the data on the page.