Software testing

Sensor Data Aggregation and Visualization System

Nischal Vooda (Z-1985022)
Niharika Prathi(Z-1981037)
Hemanth Chimakurthi(Z-1981900)

FUNCTIONAL REQUIREMENTS

BACKEND ARCHITECTURE

BME680 Sensors:
Multiple BME680 sensors are connected to Raspberry Pi Pico via wires.

Pico Network Connectivity:
Pico is connected to a network.
The LED indicator is on when Pico is successfully connected to the network.

Data structures used for Storage:
Utilizes I2C protocol to retrieve data from connected sensors.
Data is stored in Pico's memory.
The FIFO queue is used as a data structure for efficient data storage.

Data Processing:
Pico performs data processing internally.
Calculates MAX value and MEAN of the collected data.

API Functionality:
PICO acts as an API endpoint.
Responds with data in JSON format when queried.

FUNCTIONAL REQUIREMENTS

FRONT END ARCHITECTURE

Web Server Using Flask:
Flask employed as the web server

User Interaction:
User-triggered data update button on the front end.

Displayed Information:
Current Sensor Data, Max, and Mean Values of the past 30 seconds.
Total number of connected Picos.
Status of each Pico's availability.

Redundancy Handling:
Data is pulled from the backup Pico if the main Pico is unavailable.
If all Picos are down, values are set to zero.

User Preferences:
Toggle buttons for Max Values and Mean Values.
Allows users to turn off data retrieval if not interested.

NON-FUNCTIONAL REQUIREMENTS

Reliability:
Implemented robust exception handling to prevent errors and website crashes. In the event of all Picos being down, the system displays 0 instead of random errors for a more user-friendly experience.

Performance:
Achieved fast response times (less than 1 sec) for data retrieval upon clicking the "Update Data" button. Prioritized website speed and efficiency for a seamless user experience.

Maintainability:
Utilized Git for version control and code storage.
It is easy to maintain and update the changes whenever necessary in the project.

Efficiency:
Processed max and mean values of sensor data within the Pico for efficient resource utilization.we ensured the Pico's resources were utilized effectively.

Scalability:
Started with two BME680 sensors, but was easily scalable based on traffic and requirements. Flask web server seamlessly adjusts to an increasing number of connected Picos, ensuring real-time updates

NON-FUNCTIONAL REQUIREMENTS

Availability:
Implemented redundancy measures for 98% of website availability.
Even if one Pico fails, the server retrieves data from another, ensuring uninterrupted service.

Reusability:
Code is broken down into small, modular functions/methods for easy reuse.
Each function/method has a single responsibility, promoting reusability and maintainability.

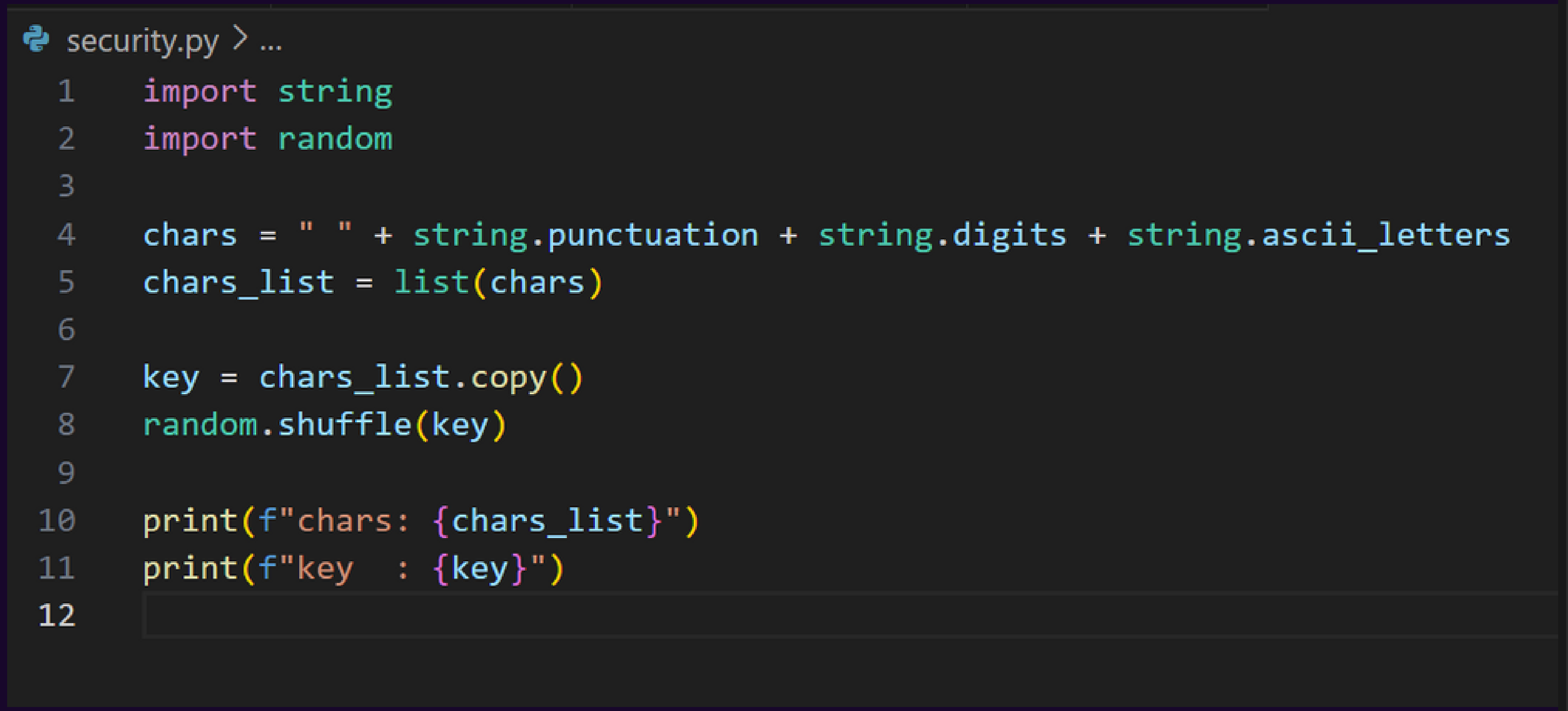
Security:
Implemented custom key-based encryption to secure data transmitted over HTTP.
Ensured client-side decryption for enhanced security measures against potential data breaches.

Correctness:
The website functions as expected based on project requirements.

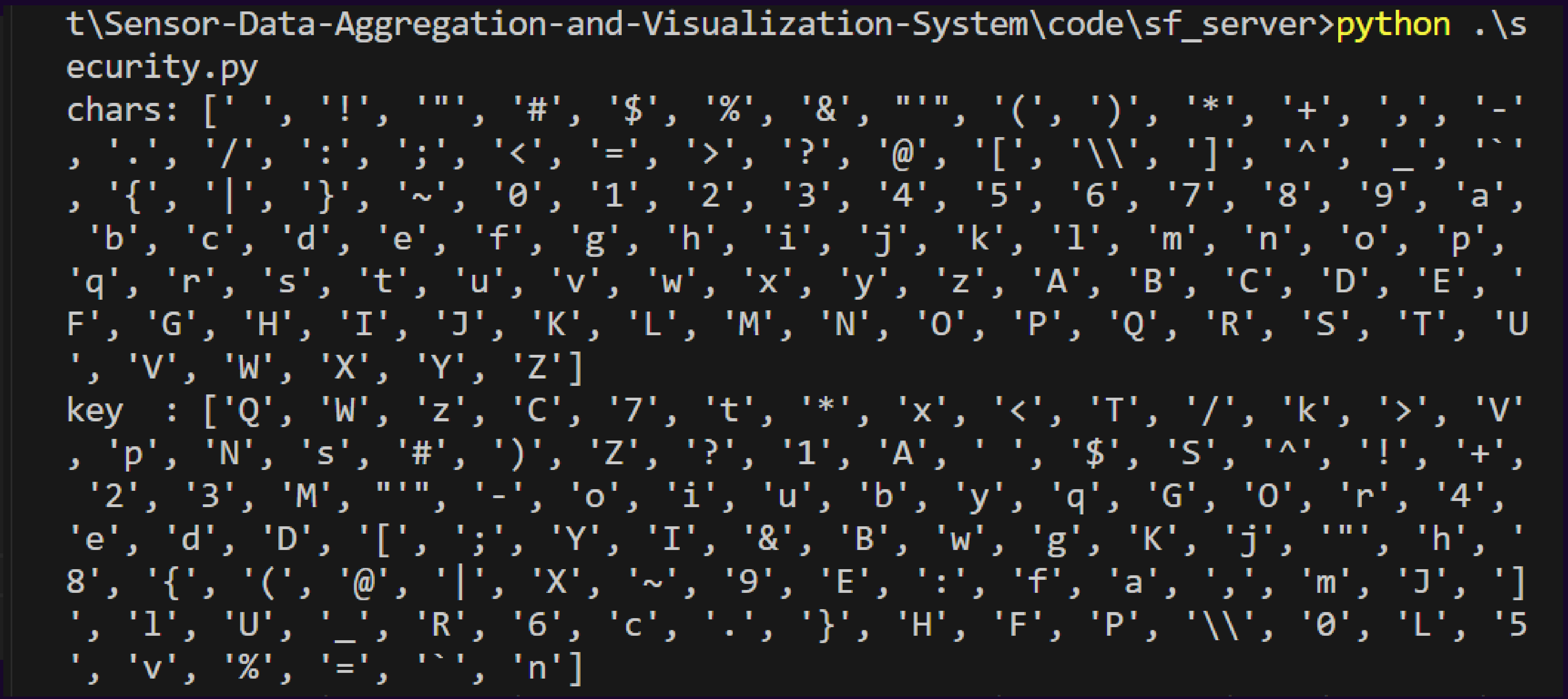
IMPLEMENTATION

Observer-Design pattern
When the user clicks on the "Update data" button server pulls sensor data from PICO to ensure efficient power management and battery conservation

Encryption and decryption
Using Python we created a custom key that is saved in PICO W and webserver. using this key the data is encrypted in PICO and decrypted in webserver before displaying the data. this helps to overcome man in the middle attack

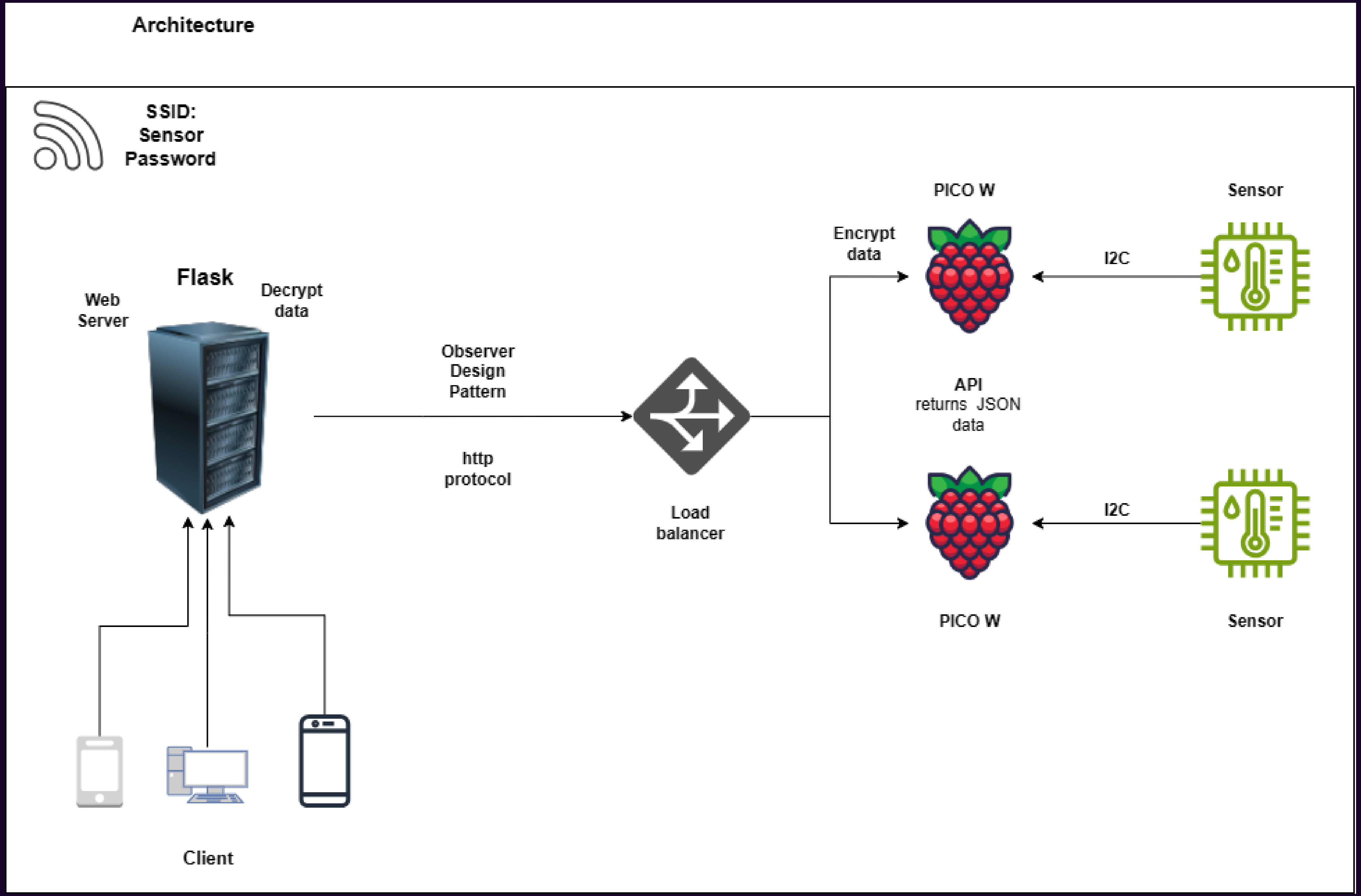

security.py > ...

```python
import string
import random

chars = " " + string.punctuation + string.digits + string.ascii_letters
chars_list = list(chars)

key = chars_list.copy()
random.shuffle(key)

print(f"chars: {chars_list}")
print(f"key  : {key}")
```

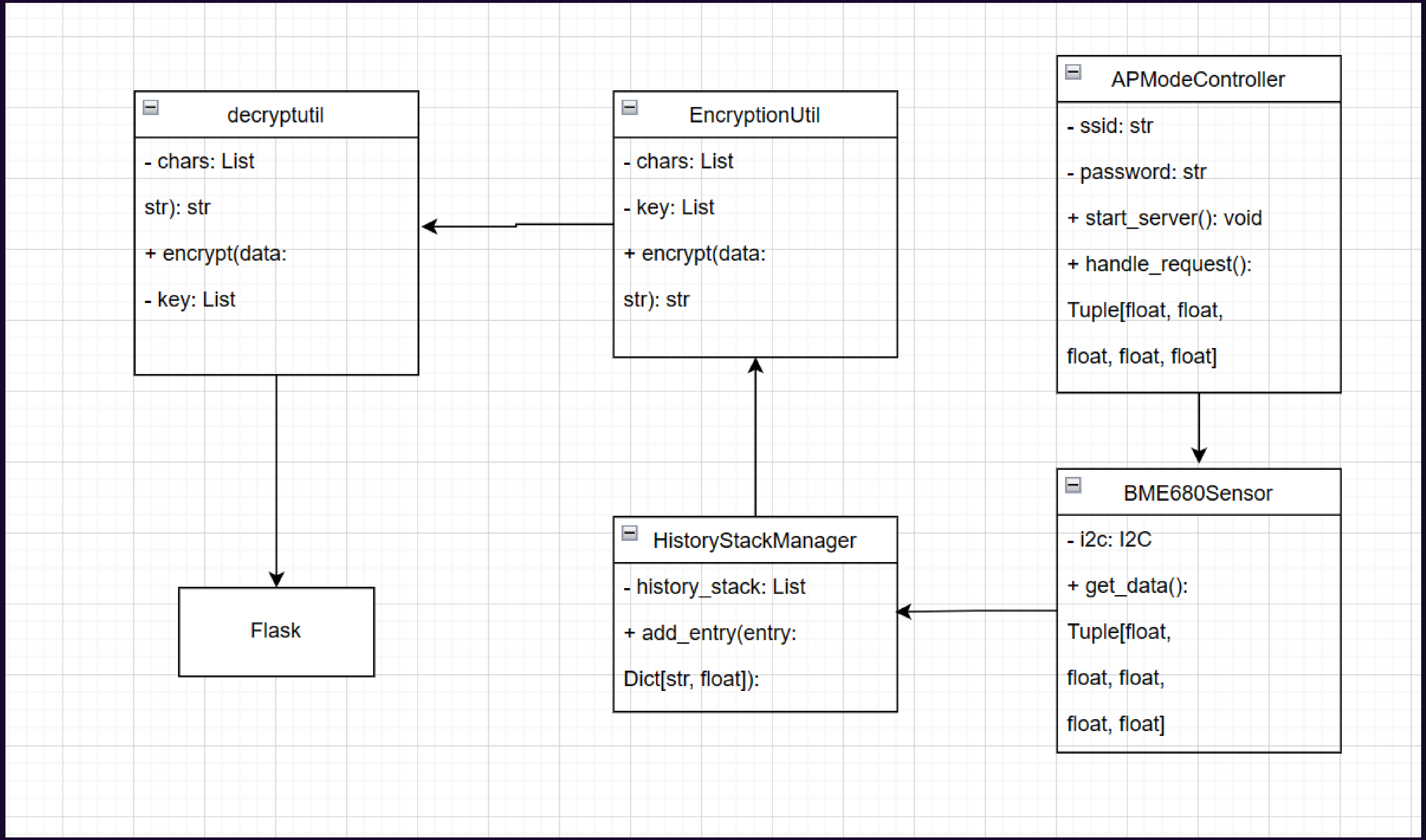


```
t\Sensor-Data-Aggregation-and-Visualization-System\code\sf_server>python .\s
ecurity.py
chars: [' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-'
, '.', '/', ':', ';', '<', '=', '>', '?', '@', '[', '\\', ']', '^', '_', '`'
, '{', '|', '}', '~', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a',
 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D', 'E', '
F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U
', 'V', 'W', 'X', 'Y', 'Z']
key  : ['Q', 'W', 'z', 'C', '7', 't', '*', 'x', '<', 'T', '/', 'k', '>', 'V'
, 'p', 'N', 's', '#', ')', 'Z', '?', '1', 'A', ' ', '$', 'S', '^', '!', '+',
 '2', '3', 'M', "'", '-', 'o', 'i', 'u', 'b', 'y', 'q', 'G', 'O', 'r', '4',
'e', 'd', 'D', '[', ';', 'Y', 'I', '&', 'B', 'w', 'g', 'K', 'j', '"', 'h', '
8', '{', '(', '@', '|', 'X', '~', '9', 'E', ':', 'f', 'a', ',', 'm', 'J', ']
', 'l', 'U', '_', 'R', '6', 'c', '.', '}', 'H', 'F', 'P', '\\', '0', 'L', '5
', 'v', '%', '=', '`', 'n']
```

HIGH-LEVEL DESIGN



Class diagram

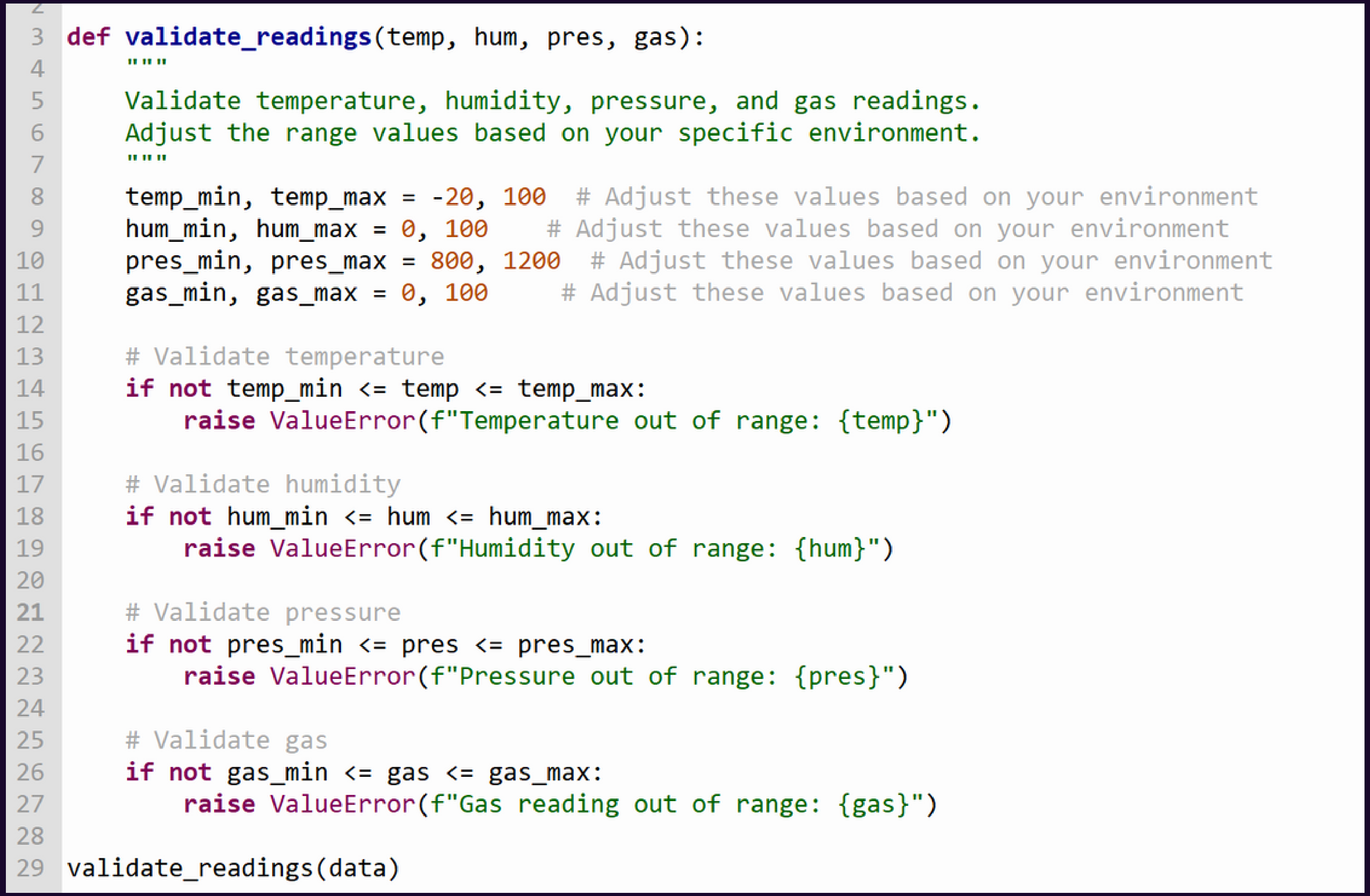


TESTING

Unit Testing

Testing Reading

To ensure that the temperature, humidity, pressure, and gas readings are within expected ranges. We add validation checks for all readings within the expected ranges.



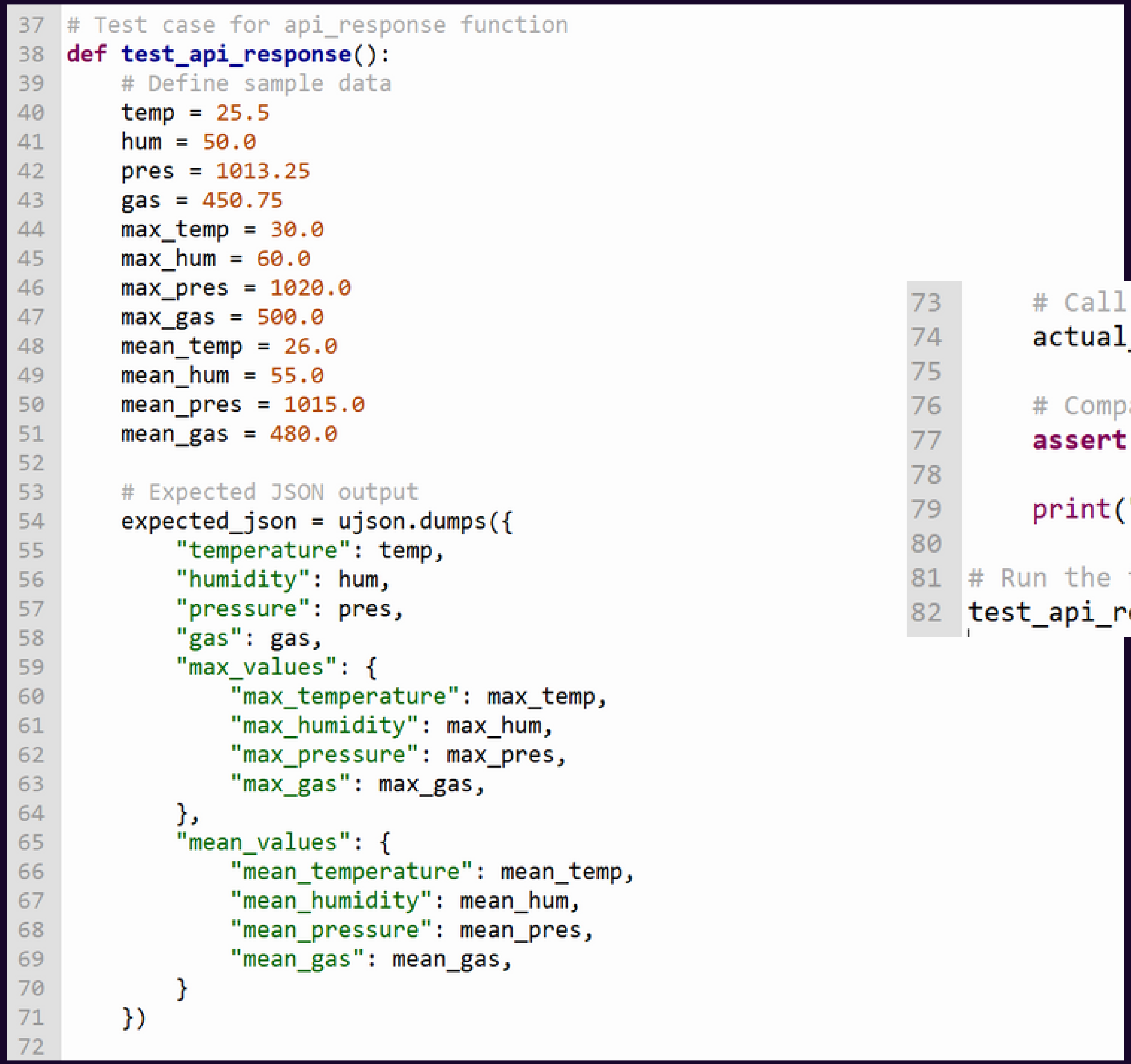
```python
def validate_readings(temp, hum, pres, gas):
    """
    Validate temperature, humidity, pressure, and gas readings.
    Adjust the range values based on your specific environment.
    """
    temp_min, temp_max = -20, 100  # Adjust these values based on your environment
    hum_min, hum_max = 0, 100    # Adjust these values based on your environment
    pres_min, pres_max = 800, 1200  # Adjust these values based on your environment
    gas_min, gas_max = 0, 100     # Adjust these values based on your environment

    # Validate temperature
    if not temp_min <= temp <= temp_max:
        raise ValueError(f"Temperature out of range: {temp}")

    # Validate humidity
    if not hum_min <= hum <= hum_max:
        raise ValueError(f"Humidity out of range: {hum}")

    # Validate pressure
    if not pres_min <= pres <= pres_max:
        raise ValueError(f"Pressure out of range: {pres}")

    # Validate gas
    if not gas_min <= gas <= gas_max:
        raise ValueError(f"Gas reading out of range: {gas}")

validate_readings(data)
```

TESTING

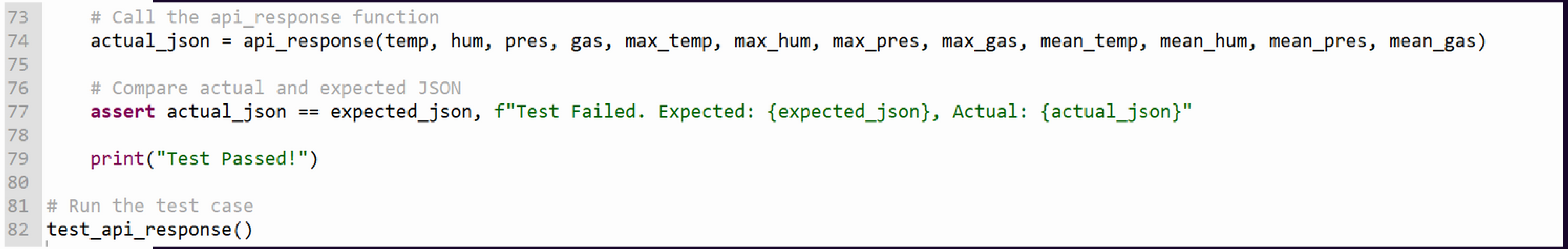
Unit Testing

API Response Testing

Test the api_response function to ensure it formats the data into the expected JSON format.
Verify that the JSON keys and structure are correct.



```python
# Test case for api_response function
def test_api_response():
    # Define sample data
    temp = 25.5
    hum = 50.0
    pres = 1013.25
    gas = 450.75
    max_temp = 30.0
    max_hum = 60.0
    max_pres = 1020.0
    max_gas = 500.0
    mean_temp = 26.0
    mean_hum = 55.0
    mean_pres = 1015.0
    mean_gas = 480.0

    # Expected JSON output
    expected_json = ujson.dumps({
        "temperature": temp,
        "humidity": hum,
        "pressure": pres,
        "gas": gas,
        "max_values": {
            "max_temperature": max_temp,
            "max_humidity": max_hum,
            "max_pressure": max_pres,
            "max_gas": max_gas,
        },
        "mean_values": {
            "mean_temperature": mean_temp,
            "mean_humidity": mean_hum,
            "mean_pressure": mean_pres,
            "mean_gas": mean_gas,
        }
    })

```



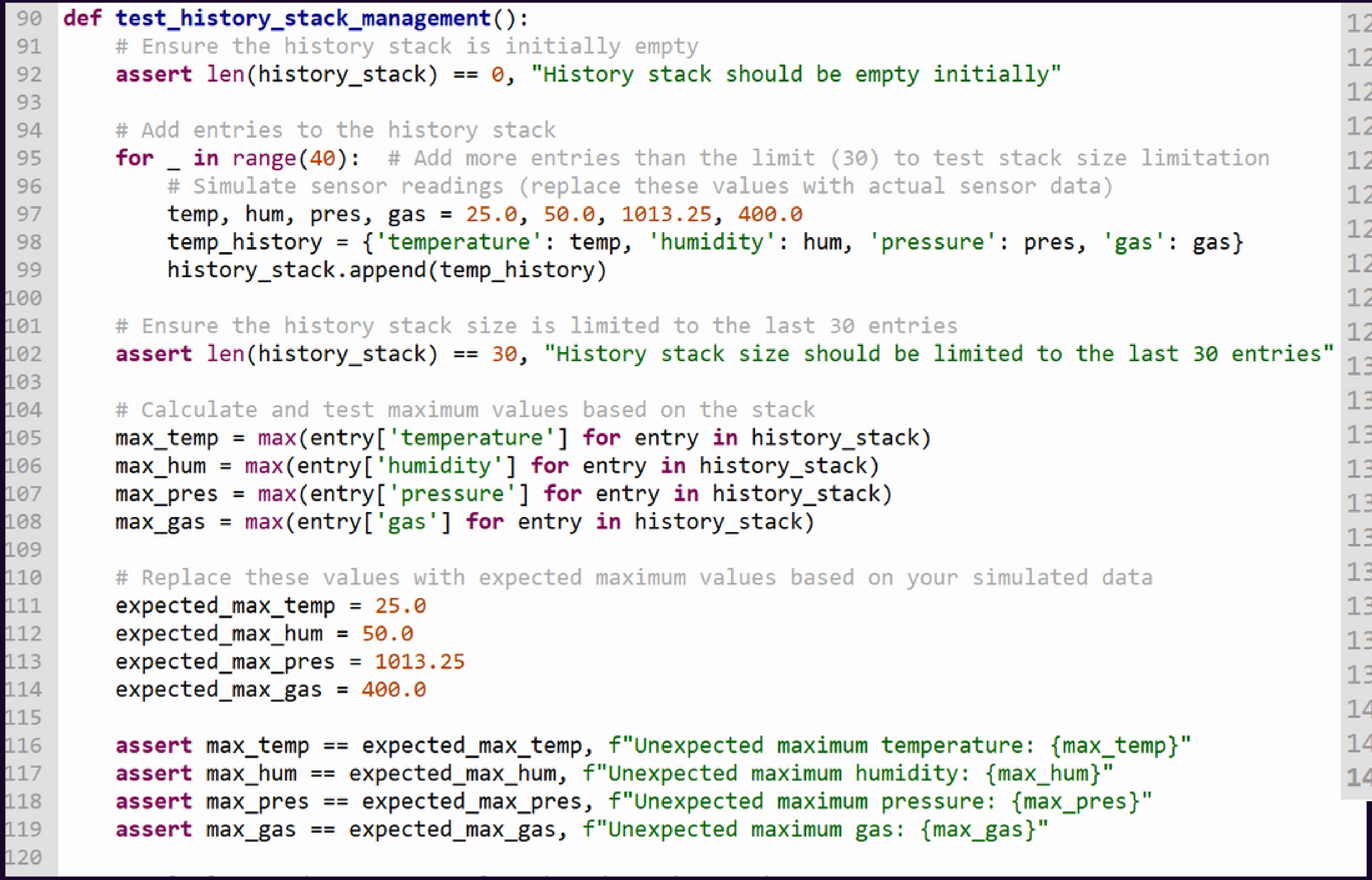
```python
    # Call the api_response function
    actual_json = api_response(temp, hum, pres, gas, max_temp, max_hum, max_pres, max_gas, mean_temp, mean_hum, mean_pres, mean_gas)

    # Compare actual and expected JSON
    assert actual_json == expected_json, f"Test Failed. Expected: {expected_json}, Actual: {actual_json}"

    print("Test Passed!")

# Run the test case
test_api_response()
```

TESTING

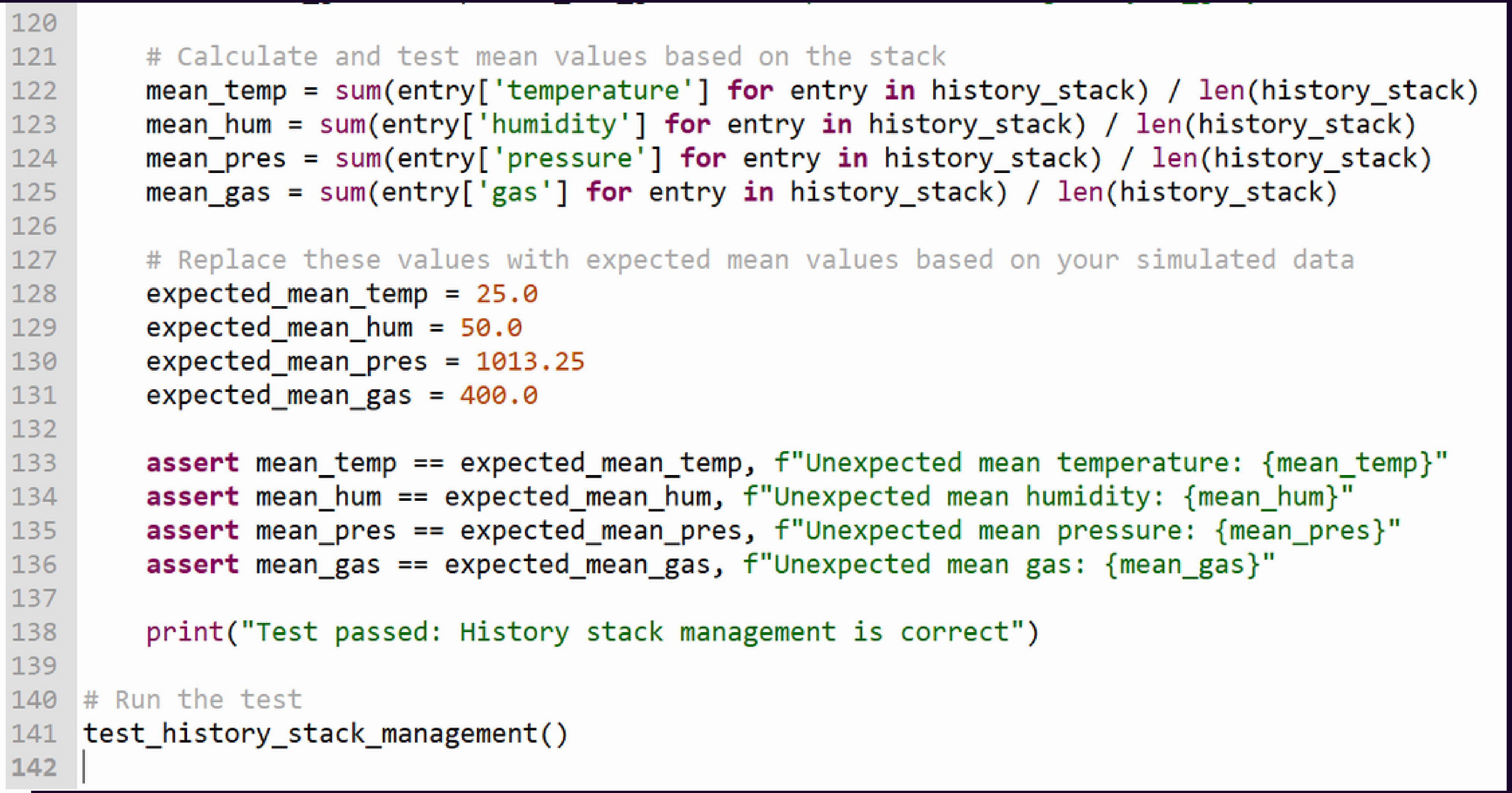
Unit Testing

History queue testing

Test that the history queue is correctly storing and limiting the number of entries to 30.
Verify that the maximum and mean values are calculated correctly based on the queue.



```python
def test_history_stack_management():
    # Ensure the history stack is initially empty
    assert len(history_stack) == 0, "History stack should be empty initially"

    # Add entries to the history stack
    for _ in range(40):  # Add more entries than the limit (30) to test stack size limitation
        # Simulate sensor readings (replace these values with actual sensor data)
        temp, hum, pres, gas = 25.0, 50.0, 1013.25, 400.0
        temp_history = {'temperature': temp, 'humidity': hum, 'pressure': pres, 'gas': gas}
        history_stack.append(temp_history)

    # Ensure the history stack size is limited to the last 30 entries
    assert len(history_stack) == 30, "History stack size should be limited to the last 30 entries"

    # Calculate and test maximum values based on the stack
    max_temp = max(entry['temperature'] for entry in history_stack)
    max_hum = max(entry['humidity'] for entry in history_stack)
    max_pres = max(entry['pressure'] for entry in history_stack)
    max_gas = max(entry['gas'] for entry in history_stack)

    # Replace these values with expected maximum values based on your simulated data
    expected_max_temp = 25.0
    expected_max_hum = 50.0
    expected_max_pres = 1013.25
    expected_max_gas = 400.0

    assert max_temp == expected_max_temp, f"Unexpected maximum temperature: {max_temp}"
    assert max_hum == expected_max_hum, f"Unexpected maximum humidity: {max_hum}"
    assert max_pres == expected_max_pres, f"Unexpected maximum pressure: {max_pres}"
    assert max_gas == expected_max_gas, f"Unexpected maximum gas: {max_gas}"

```



```python

    # Calculate and test mean values based on the stack
    mean_temp = sum(entry['temperature'] for entry in history_stack) / len(history_stack)
    mean_hum = sum(entry['humidity'] for entry in history_stack) / len(history_stack)
    mean_pres = sum(entry['pressure'] for entry in history_stack) / len(history_stack)
    mean_gas = sum(entry['gas'] for entry in history_stack) / len(history_stack)

    # Replace these values with expected mean values based on your simulated data
    expected_mean_temp = 25.0
    expected_mean_hum = 50.0
    expected_mean_pres = 1013.25
    expected_mean_gas = 400.0

    assert mean_temp == expected_mean_temp, f"Unexpected mean temperature: {mean_temp}"
    assert mean_hum == expected_mean_hum, f"Unexpected mean humidity: {mean_hum}"
    assert mean_pres == expected_mean_pres, f"Unexpected mean pressure: {mean_pres}"
    assert mean_gas == expected_mean_gas, f"Unexpected mean gas: {mean_gas}"

    print("Test passed: History stack management is correct")

# Run the test
test_history_stack_management()

```

TESTING

Acceptance Testing

Alpha testing

During alpha testing, the website was thoroughly evaluated to ensure proper functionality. Satisfied with the results, we confirmed that all features operated smoothly, meeting initial development expectations.

TESTING

Acceptance Testing

Beta Testing

Website shared with peers for feedback during beta testing.

The feedback from the user:
The website is considered user-friendly and smoothly functioning.
Users understand the website's purpose and functionality with minimal guidance.

Color Contrast Feedback:
Some users suggest that color contrast could be improved.
Recommendation to implement a lighter theme for better visual appeal.

Data Retrieval Feedback:
Users sometimes feel the need for only current data.
Suggestion to provide an option to avoid retrieving max and mean values when not needed.

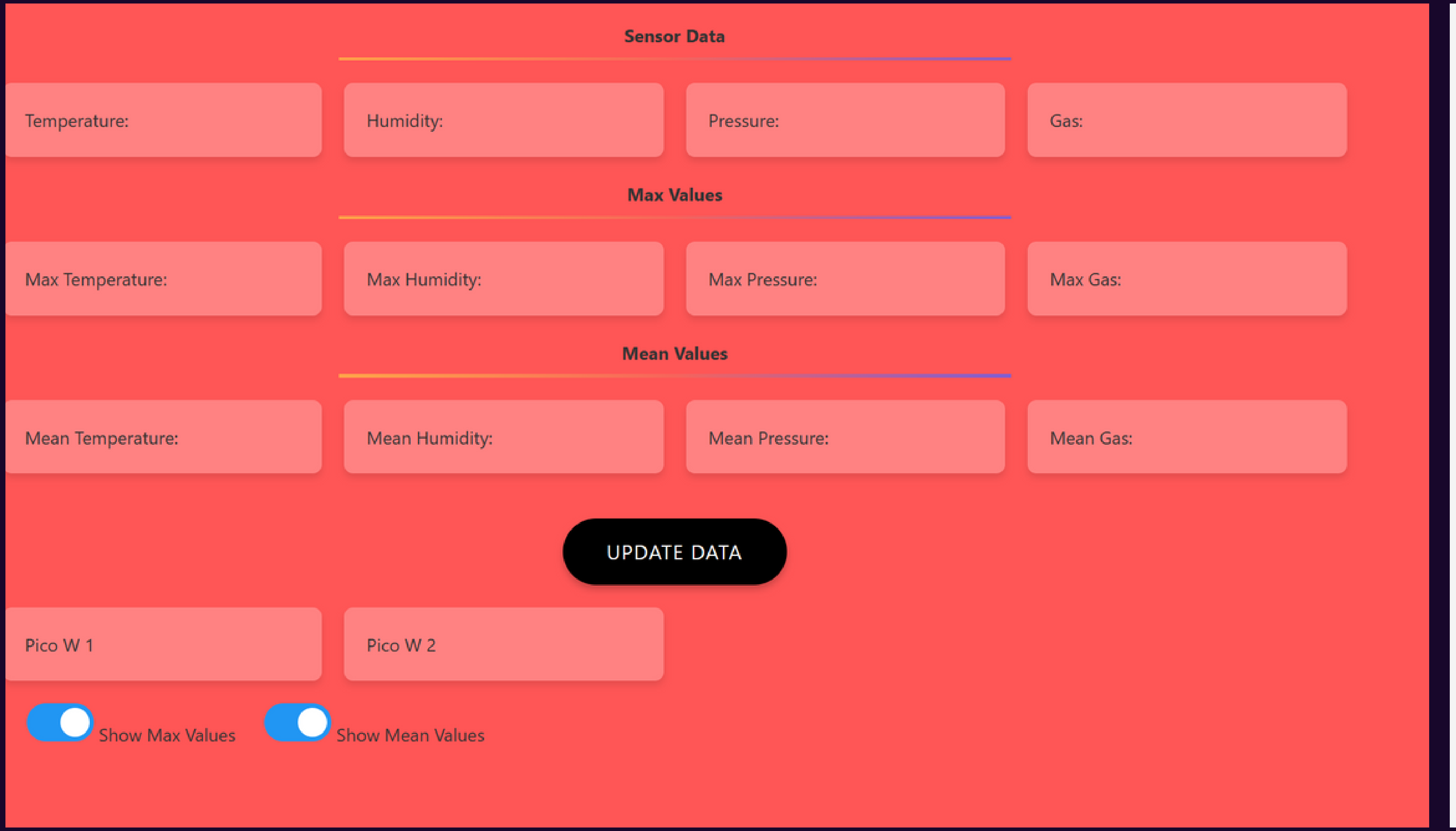
TESTING

Acceptance Testing

Beta Testing

As the user suggested we changed the odd theme to a good light theme

Before



Sensor Data

Temperature:	Humidity:	Pressure:	Gas:

Max Values

Max Temperature:	Max Humidity:	Max Pressure:	Max Gas:

Mean Values

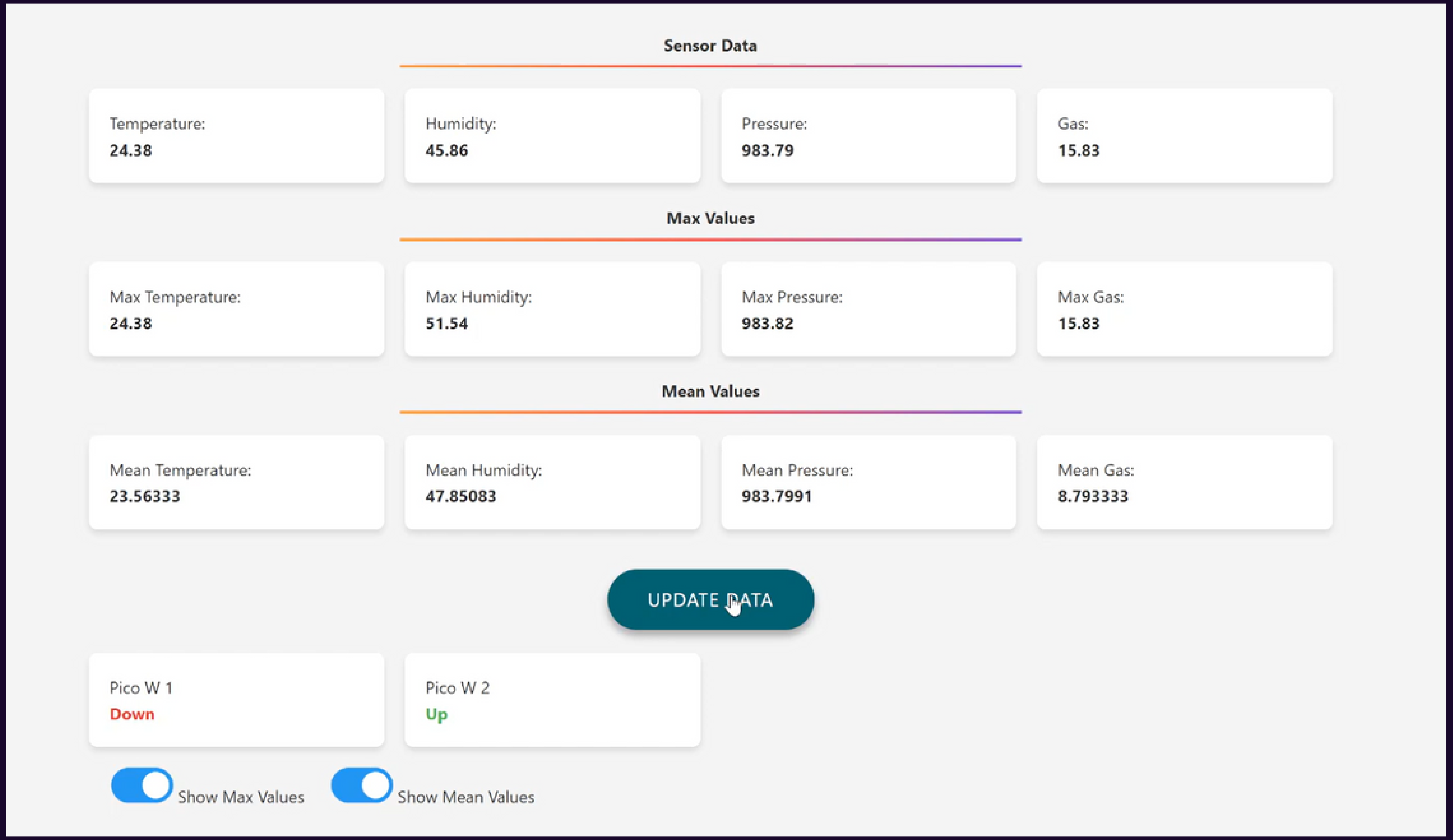
Mean Temperature:	Mean Humidity:	Mean Pressure:	Mean Gas:

UPDATE DATA

Pico W 1	Pico W 2

Show Max Values Show Mean Values

After



Sensor Data

Temperature: 24.38	Humidity: 45.86	Pressure: 983.79	Gas: 15.83

Max Values

Max Temperature: 24.38	Max Humidity: 51.54	Max Pressure: 983.82	Max Gas: 15.83

Mean Values

Mean Temperature: 23.56333	Mean Humidity: 47.85083	Mean Pressure: 983.7991	Mean Gas: 8.793333

UPDATE DATA

Pico W 1 Down	Pico W 2 Up

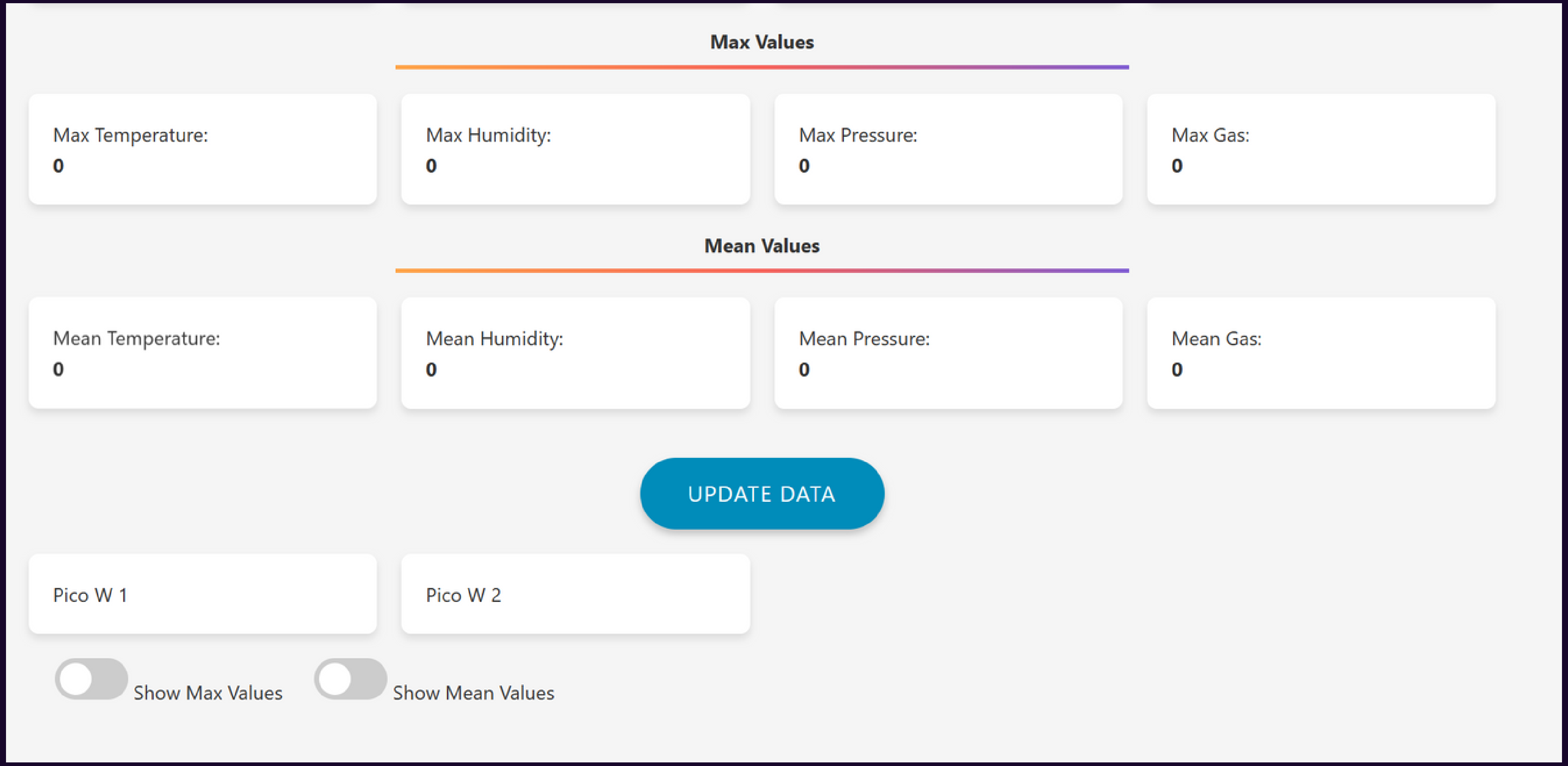
Show Max Values Show Mean Values

TESTING

Acceptance Testing

Beta Testing

As the user suggested we added two switches for Max Values and Mean Values so if the user feels like there is no need to retrieve this data he or she can turn it off also this will improve the website quality



Max Values

Max Temperature:
0

Max Humidity:
0

Max Pressure:
0

Max Gas:
0

Mean Values

Mean Temperature:
0

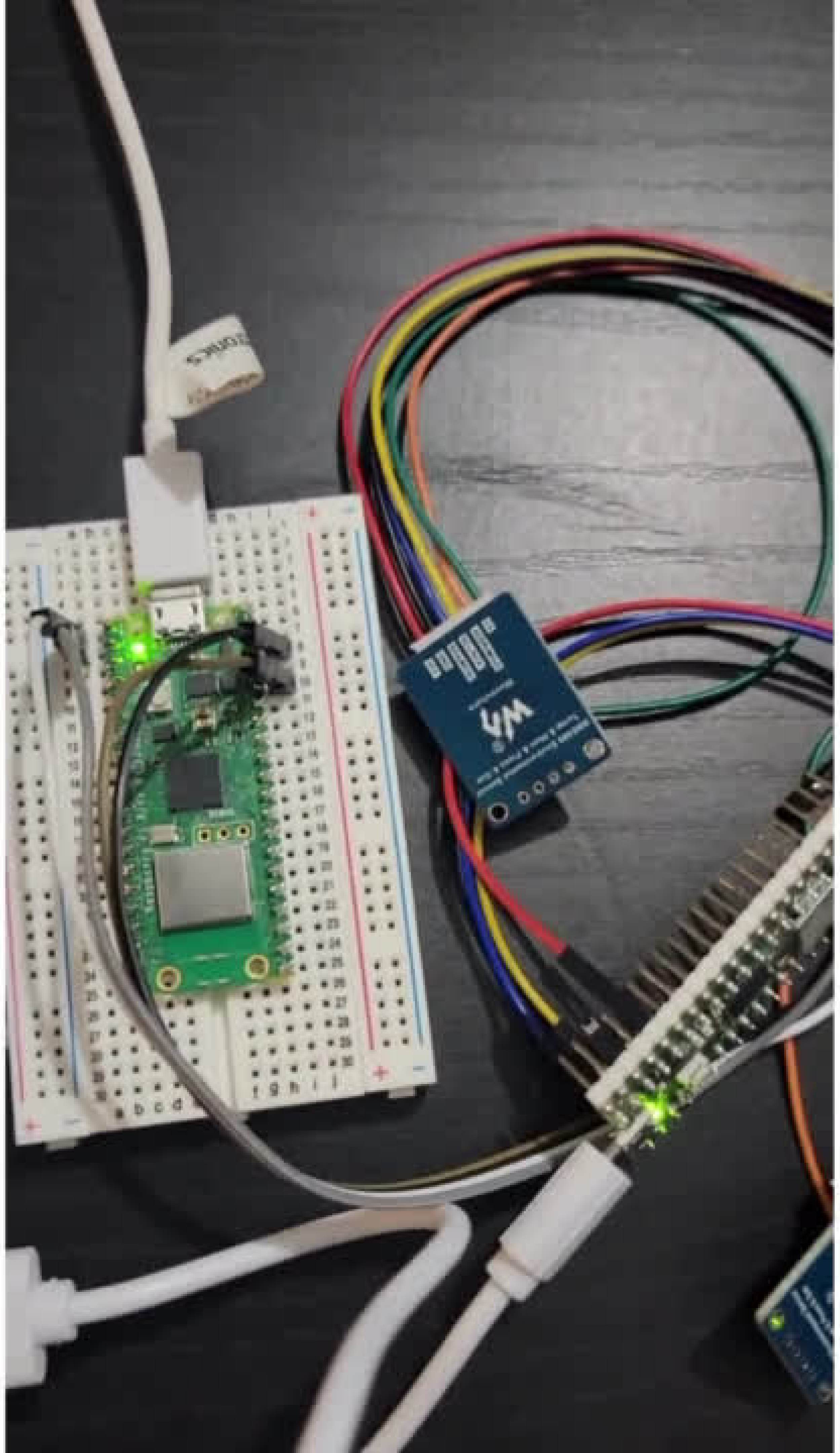
Mean Humidity:
0

Mean Pressure:
0

Mean Gas:
0

DEMO

DEMO



Q&A

Q & A

ThankYou!!