HW3 - Group1

Mert Toprak (s336966) **Daghan Ufuk Leflef** (s338142) **Muhammed Emin Oral** (s343124)

I. System Implementation

A. Data Collection (Publisher)

We developed the `publisher.py` script for the Raspberry Pi. It uses the `adafruit_dht` library to read temperature and humidity from pin D4. We used the `paho-mqtt` library to connect to the broker `broker.emqx.io`. The script generates a payload containing the MAC address, a timestamp, and the sensor data. This JSON message is published every 5 seconds to the topic `s336966_s338142_s343124`.

B. Data Storage (Subscriber)

The `subscriber.ipynb` notebook manages data persistence. It connects to both the MQTT broker and the Redis Cloud database. Upon receiving a message, it parses the JSON and stores the values using **Redis Time-Series**. During our test, the subscriber successfully processed data from our device (MAC `26:99:64:91:45:14`). As shown in the notebook output, we stored values such as **Temperature: 20°C** and **Humidity: 58%** into keys like `26:99:64:91:45:14:temperature`.

C. REST Server

We extended the API in `rest_server.ipynb` using `CherryPy`. We added the `HistoricalData` resource with a `GET` endpoint at `/data/{mac_address}`. The logic accepts a `count` parameter (validated to ensure it is positive). It uses the Redis command `revrange` to retrieve the specific number of most recent samples. These are formatted into lists (timestamp, temperature, humidity) and returned as JSON.

D. REST Client & Visualization

In `rest_client.ipynb`, we verified the system. We first checked the status (`200 OK`) and registered the sensor. We then requested the last 10 data points. The server response was successfully converted into a Pandas DataFrame. The notebook output confirmed the retrieval of data:

```
timestamp temp humidity
2025-12-19 14:22:02.843 20 57
2025-12-19 14:22:08.098 20 57
```

Finally, we used `matplotlib` to generate line plots, visualizing the stability of the temperature (approx 20°C) and the variation in humidity over time.

II. Discussion

A. MQTT vs. REST for the Device

Why MQTT? MQTT is better than REST for the Raspberry Pi for these reasons:

1) **Lightweight:** MQTT messages have a very small header. Since we send data every 5 seconds, this saves bandwidth compared to heavy HTTP headers.
2) **Decoupling:** The Pi (Publisher) doesn't need to know who receives the data (Subscriber). If the database goes offline, the Pi can keep working without errors.
3) **Connection:** MQTT keeps one open TCP connection. REST would require a new ”handshake” for every single message, which is slower and uses more battery/cpu.

B. HTTP Method Selection

Method used: `GET`.

Motivation: We chose `GET` for the `/data` endpoint because its purpose is solely to **retrieve** information. In REST standards, `GET` implies reading data without changing the server's state (it is ”safe” and ”idempotent”). Methods like `POST` or `PUT` are for creating or updating data, which is not what we are doing here.

III. Conclusion

We were able to implement the complete IoT system. The code logic handles the flow from the sensor to the dashboard. The specific outputs in our notebooks confirm that data was correctly transmitted, stored in Redis, and visualized via the REST API.