Solar Energy Monitoring System

Final Report

Group: Dário Guimarães and Joseph Koyi

1 Project Goal

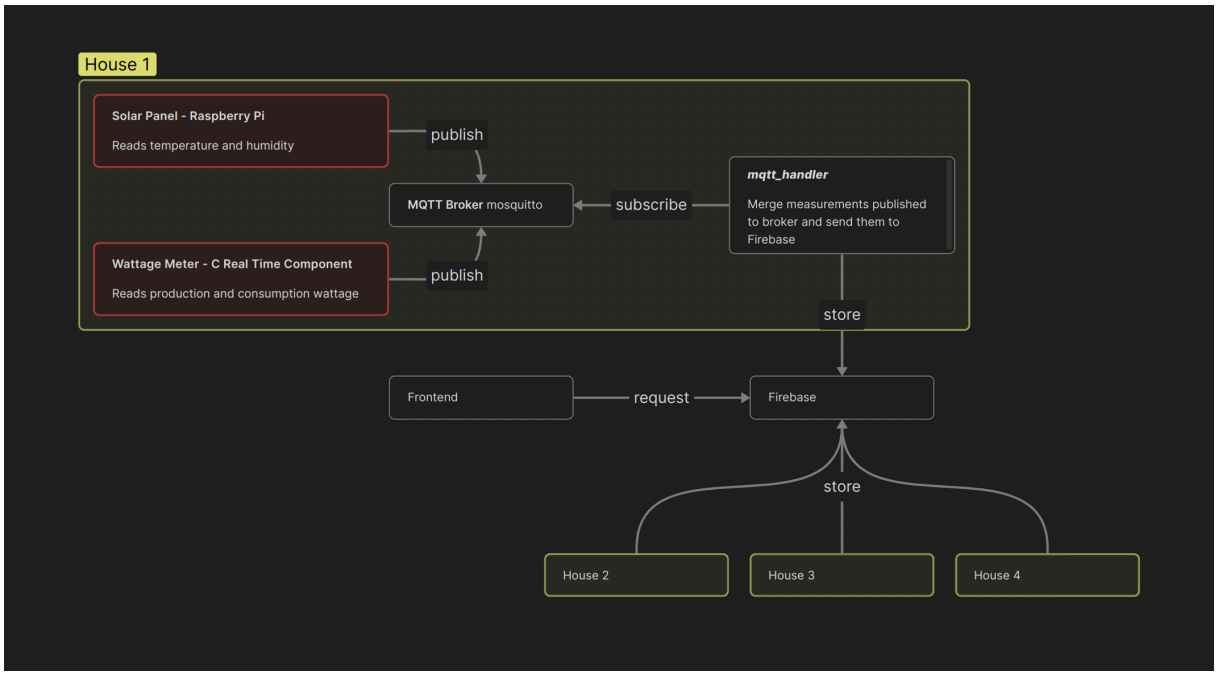
As described on the Project Ideia assignement, the objective of this project was to create a system for real-time monitoring of environmental and electrical data related to solar energy production. A Raspberry Pi equipped with a Sense HAT was used to collect local weather data on the solar panel and, in parallel, a real-time component written in C simulate measurements from the household electrical panel (power consumption and solar panel production).

2 Arquiteture

The architecture of the application is divided into two main layers: the **House layer** and the **Global layer**.

In the **House layer**, each household is equipped with a solar panel installation and a corresponding Raspberry Pi unit, referred to as the `Solar Panel - Raspberry Pi`. This device collects the temperature and humidity near the solar panel using the Sense HAT. Additionally, a Real Time C component, named `Wattage Meter - C Real Time Component`, simulates that reads data from the household's electrical panel, specifically the home's power consumption and solar energy production. Both devices publish their data to a local MQTT broker under the apropriated topics. A dedicated component within the house, the `mqtt_handler`, is responsible for subscribing to these MQTT topics, aggregating the incoming data, and pushing it to a centralized Firebase database.

The **Global layer** consists of a single Firebase database instance that receives and stores data from all homes using the system. This centralized data storage allows for consistent and scalable access to information across multiple installations. The data in Firebase is then accessed through a frontend interface, which provides users with real-time insights into their energy consumption and solar panel performance, as well as allowing them to verify whether the production levels are consistent with the current weather conditions.



3 Implementation

3.1 solar_panel_rpi.py

The `solar_panel_rpi.py` script is responsible for collecting temperature and humidity data near the solar panel using the Raspberry Pi's Sense HAT. If the Sense HAT is not available, it falls back to generating realistic simulated data based on time of day using sine functions. The script establishes a connection to a local MQTT broker and periodically publishes the collected environmental data in JSON format to the topic `sensor/solar_panel_rpi`. The simulation compresses a 24-hour day into a 240-second cycle for faster testing and development.

3.2 wattage_meter_rt_c_component.c

The `wattage_meter_rt_c_component.c` program simulates realistic data from a household electrical panel, including energy consumption and solar panel production. It leverages a real-time timer to execute operations at consistent intervals, ensuring that the simulation behaves deterministically over time. Like the weather data component, it compresses a full 24-hour day into a much shorter time span (e.g., 240 seconds), enabling rapid visualization of daily energy patterns. The program calculates power values based on the simulated time of day, reflecting typical usage and generation trends. These values are then published to a local MQTT broker under the topic `sensor/wattage_meter_rt_c_component`, making the data available for further aggregation and visualization in the system.

3.3 mqtt_handler.py

The `mqtt_handler.py` script acts as the bridge between the local MQTT system in each house and the centralized Firebase database. It connects to all the MQTT topics mentioned above and listens for the most recent messages from both components. Once it has received a fresh message from each source, it aggregates the data into a single payload. This payload includes environmental measurements, energy consumption and production values, a `house_id` to uniquely identify the source, a compressed-time hour to align with the system's accelerated 24-hour cycle, and a unique `order_id` epoch time timestamp to maintain message ordering on Firebase. The complete message is then sent to the Firebase database, ensuring temporal consistency and proper identification across the system.

This is the JSON sent to Firebase:

```
{
    // Measurements
    "humidity",             // from Solar Panel - Raspberry Pi
    "temperature",          // from Solar Panel - Raspberry Pi
    "consumption_wattage",  // from Wattage Mete - C Real Time Component
    "production_wattage",   // from Wattage Mete - C Real Time Component

    // Extra Data
    "house_id",             // Unique House Identifier
    "hour",                 // Fake spanned hour (value: int((time.time() / 10) % 24))
    "order_id"              // Epoch time timestamp to order entries on Firebase
}
```

4 Replication

To a quick setup is recommended run all the components on the same machine, since the default Broker IP address is set to `localhost`, but all the components accept the broker IP and port as parameter, to a more realistic setup.

Is possible replicate the full Solar Panel Monitoring System following these steps.

4.1 1. Create a virtual environment

Creates a Python environment.

```
$ python -m venv .venv
$ source .venv/bin/activate # run this in every new terminal session
```

4.2 2. Install dependencies

This project depends on some Python libraries and the Mosquitto MQTT Broker (including client and developer libraries).

Install Python dependencies:

```
$ pip install -r requirements.txt
```

Install Mosquitto client and developer library:

```
# Debian/Ubuntu
$ sudo apt install mosquitto libmosquitto-dev

# Arch Linux
$ sudo pacman -S mosquitto
```

4.3 3. Start the Mosquitto broker

The first component to be initialized is the Mosquitto MQTT broker.

```
$ mosquitto -c mosquitto.conf
```

4.4 4. Start the meters

Start Solar Panel - Raspberry Pi:

```
$ python solar_panel_rpi.py

# or if you want to specify MQTT address
$ python solar_panel_rpi.py [broker IP] [port]
```

Start Wattage Meter - C Real Time Component

```
$ gcc wattage_meter_rt_c_component.c -o wattage_meter_rt_c_component -lm -lmosquitto
$ ./wattage_meter_rt_c_component

# or if you want to specify MQTT address
$ ./wattage_meter_rt_c_component [broker IP] [port]
```

4.5 5. Start the MQTT handler

The final piece of the project is the MQTT Handler. This script requires as a parameter a unique identifier per house.

Is possible update the Firebase database updating the variable `FIREBASE_URL`.

```
$ python mqtt_handler.py <house_id>

# example usage
$ python mqtt_handler.py HOUSE_123
```

```
# or if you want to specify MQTT address
$ python mqtt_handler.py <house_id> [broker IP] [port] # optional: to specify MQTT address
```

After completing these steps, the system will be fully operational, with synchronized and time-consistent sensor data being pushed to Firebase for each registered house.