Solar Energy Monitoring System

Final Report

Group: Dário Guimarães and Joseph Koyi

1 Project Goal

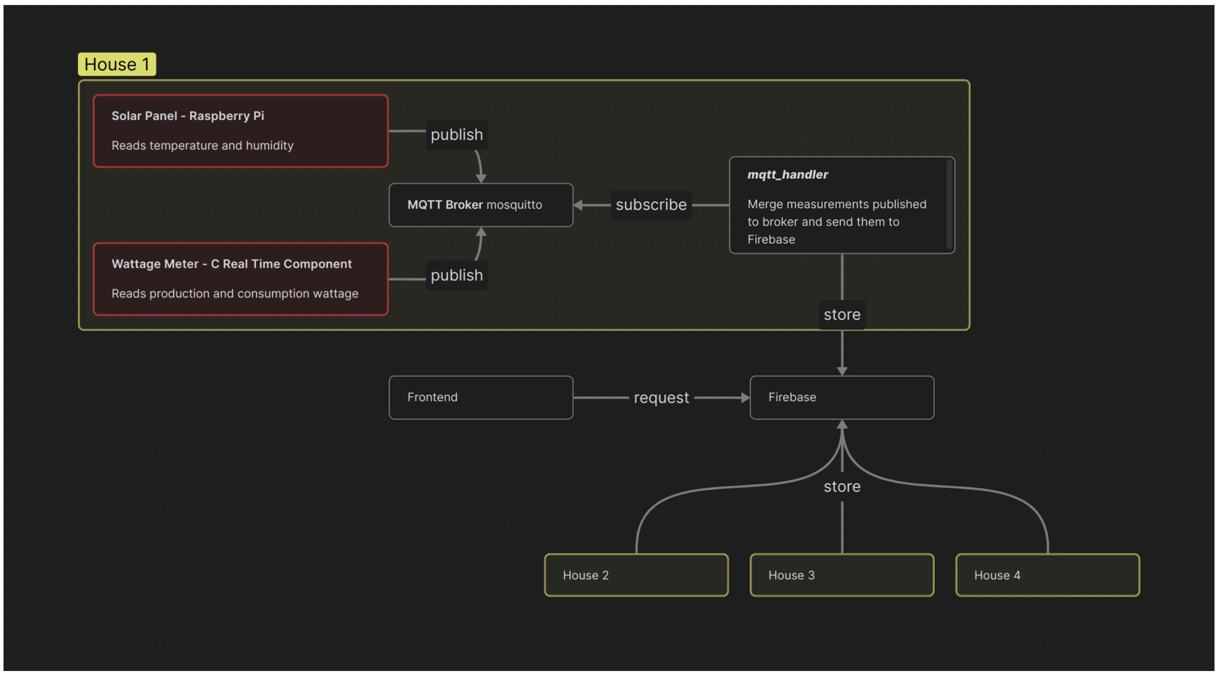
As outlined in the Project Idea assignment documentation, the goal of this project was to develop a system for real-time monitoring of environmental and electrical data related to solar energy production. A Raspberry Pi equipped with a Sense HAT was used to collect local weather data near the solar panel, while a separate real-time component written in C was used to simulate measurements from the household electrical panel, including power consumption and solar panel output.

2 Architecture

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 readings 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 appropriate topics. A dedicated component within the house, the `mqtt_handler`, subscribes to these MQTT topics, aggregates the incoming data, and pushes 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, and enables them to verify whether the production levels are consistent with the current weather conditions.


House 1
Solar Panel - Raspberry Pi
Reads temperature and humidity
publish
MQTT Broker mosquitto
subscribe
mqtt_handler
Merge measurements published to broker and send them to Firebase
Wattage Meter - C Real Time Component
Reads production and consumption wattage
publish
store
Frontend
request
Firebase
store
House 2
House 3
House 4

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. 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`.

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. The program calculates power values based on the hour of the 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`.

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, and a unique epoch timestamp that helps maintain message ordering in Firebase.

The complete message is then sent to the Firebase database under the respective topic `/houses/$HOUSE_ID`, ensuring that each house has its measurements properly grouped.

This is the JSON sent to Firebase:

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

    // Extra Data
    "timestamp"                // Epoch time to order entries on Firebase
}
```

3.4 Frontend

The frontend application is built using React. It provides a user-friendly interface for visualizing the data collected from each house.

For demonstration purposes, the `HOUSE_ID` can be specified via a query parameter in the URL using the format `?house_id=HOUSE_ID`.

Once a valid `HOUSE_ID` is provided, the UI fetches the corresponding data from Firebase and displays interactive charts showing daily, weekly, and monthly values.

4 Replication

For a quick setup, it's recommended to run all the components on the same machine, as the default broker IP address is set to `localhost`. However, all components accept the broker IP and port as parameters for more realistic deployments.

You can replicate the full Solar Panel Monitoring System by following these steps:

4.1 1. Create a virtual environment

Create a Python virtual 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, the Mosquitto MQTT Broker (including client and developer libraries), and a Node.js environment for the frontend.

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
```

Install `Node.js` and `npm` through the official documentation.

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 the Solar Panel - Raspberry Pi script:

```
$ python solar_panel_rpi.py

# or specify MQTT address:
$ python solar_panel_rpi.py [broker IP] [port]
```

Start the 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 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 a unique identifier per house as a parameter.

You can update the Firebase database URL by modifying the `FIREBASE_URL` variable.

```
$ python mqtt_handler.py <house_id>

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

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

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.

4.6 6. Frontend

To launch the frontend interface, navigate to the frontend-dis directory, install the necessary dependencies, and start the development server:

```
$ cd frontend-dis
$ npm i
$ npm run dev
```

Once the server is running, you can access the user interface by visiting: `http://localhost:8080/?house_id=<HOUSE_ID>`

Make sure that the HOUSE_ID matches the one used with `mqtt_handler.py`.