UNIVERSITY OF L'AQUILA

Department of Information Engineering, Computer Science and Mathematics

PROJECT REPORT

Software Engineering for the Internet of Things

Project name: PeakFlip – smart boiler

Github link: https://github.com/vdavidaron/PeakFlip

Submitted to Professor Davide Di Ruscio
Department of Information Engineering,
Computer Science and Mathematics,
University of L'Aquila

STUDENT GROUP

David Aron Vigh
davidaron.vigh@student.univaq.it
Md Tasluf Morshed
mdtasluf.morshed@student.univaq.it
Tien Dung Nguyen
tiendung.nguyen@student.univaq.it

Erasmus Mundus M.Sc. Programme
Software Engineers for Green Deal (SE4GD)

L'Aquila, January 2025

Table of Contents

Project specification

As the electricity market in Europe becomes more and more liberal, spot pricing is now a norm. This change is largely due to the widespread renewable energy usage and supply-demand imbalance. With spot pricing, electricity prices fluctuate throughout the day, making energy costs unpredictable. For example, in the morning when the sun shines, solar energy boosts the electricity supply, dropping the electricity price. In contrast, in the evening, supply decreases and demand go up, prices can skyrocket. This leads to the need for a solution to utilize electricity bills, like purchasing electricity during an off-peak while still maintaining the necessary electricity usage.

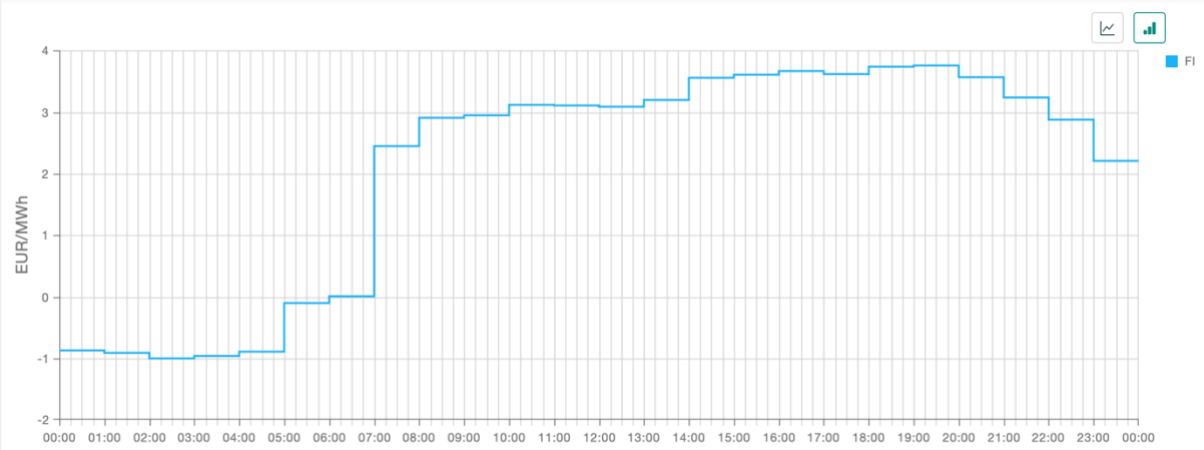


Figure 1. Electricity price by date (Source: NordPool)

We propose an IoT-based smart house system solution, PeakFlip, by taking advantage of fluctuating spot prices. This system automates electricity usage during low-price periods, helping consumers lower their energy costs. Additionally, PeakFlip can send alerts if predicted electricity consumption exceeds a pre-set threshold or if peak electricity price hours are coming. To demonstrate its functionality, we simulated a smart water boiler that regularly measures water level, as well as inside and outside temperature. The simulated device aims to reduce electricity consumption while maintaining a comfortable water temperature.

PEAKFLIP
ELECTRICITY

Figure 2. PealkFlip

Device description

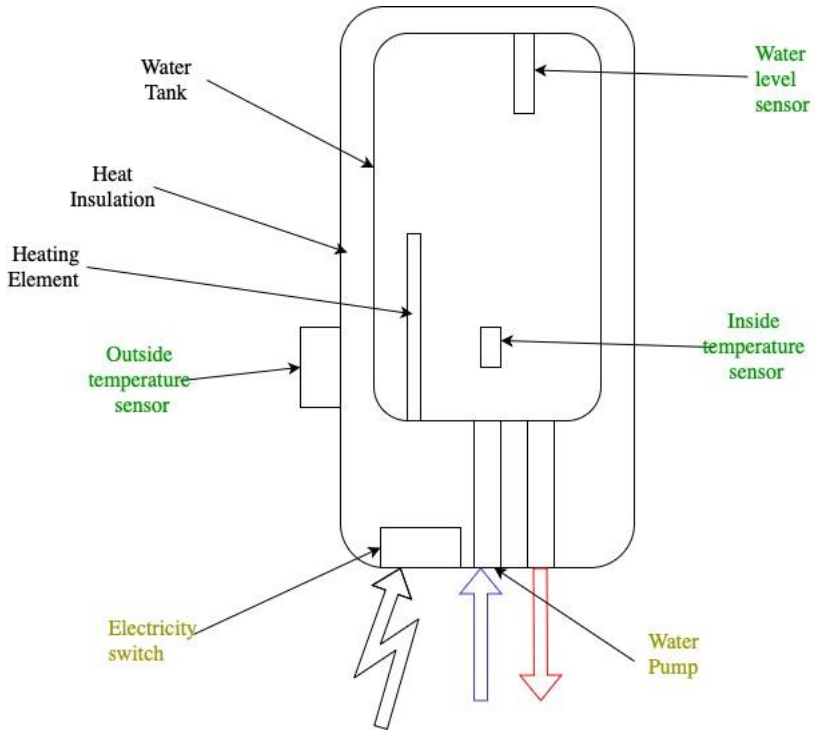


Figure 3. Boiler components

The proposed system features a smart boiler equipped with multiple sensors and actuators. Its primary purpose is to heat water for use in an average household. The water tank is refilled as needed, and the heater activates when electricity prices reach a favourable level. Hot water is dispensed based on household consumption. The boiler is insulated to retain heat and keep the water warm for extended periods.

The system includes sensors to monitor the internal water temperature, water level, and external air temperature. These measurements could support future algorithm improvements aimed at reducing electricity costs. The boiler is equipped with two actuators: a water pump for refilling the tank and an electric switch for activating the heater. The electric switch is also fitted with an electricity meter to track energy usage.

In this implementation, the sensors and actuators connect to an ESP32 microcontroller, which communicates data using the MQTT protocol.

Parameters required for PeakFlip

Water temperature

Refers to the temperature of the water inside the smart water boiler. This parameter is critical to ensure the water is maintained within a comfortable range, avoiding excessive heating during high electricity price periods. It helps optimize energy usage while ensuring water optimal temperature by adjusting heating cycles based on current electricity prices.

Outside temperature

This sensor measures the ambient temperature outside the heater in the house. This data helps the system adapt heating requirements based on weather conditions, such as during colder periods when higher water temperatures may be needed.

Water level

This parameter ensures sufficient hot water availability for user needs by pumping water in if water level is low. Furthermore, it can be used for efficient energy management by heating only the required amount of water, avoiding wasteful heating of an empty or overfilled tank.

Electricity consumption

This parameter monitors the amount of electricity consumed by the boiler. The electricity usage sent by the water boiler is crucial to provide real-time insight into energy usage, calculate electricity cost and identify overconsumption and inefficiencies.

Functional and non-functional requirements

Functional requirements

Monitor data from sensors

Collecting real-time data periodically from different sensors: water level, water temperature and outside temperature. By monitoring these data, it is possible to have insight into the current status and take action if needed.

Process and store data acquired

The collected data should be transformed into a format that can be stored or analyzed. By doing this, the system can decide which action to take to ensure the water level and temperature or to fulfill the necessary electricity usage to boil water based on the electricity price.

Data visualization

Visualizing both sensor data and data derived from it enables users to identify trends, patterns and electricity usage. The data should be presented in a user-friendly form that is informative. This helps users monitor real-time electricity consumption, control their electricity bills automatically and intervene in case of abnormality.

Send alerts about electricity overconsumption and notifications

This feature monitors electricity consumption patterns in real time and sends threshold-based alerts when usage exceeds predefined thresholds, indicating potential overconsumption. These alerts help users take corrective actions to avoid unexpected high energy costs.

The feature also notifies users about upcoming periods of low electricity spot prices for scheduled usage. Notifications include details about the expected duration of low prices and potential savings, keeping users informed about their energy usage.

Non-functional requirements

Performance

The system should be capable of handling and analyzing real-time data with low latency to produce visualize statistics.

Resilience

The system should handle errors that occur in case of communication failures.

Privacy

Only authorized users should have access to configure the system or view consumption data.

Usability

The user interface must be simple, allowing non-technical users to monitor and control devices easily.

Interoperability

The system should be compatible with various smart electricity devices and sensors for aggregating data across heterogeneous sources

Reliability

The system should provide reliable, complete and accurate data, alerts and notifications.

Scalability

The system should be able to integrate new devices without configuration overhead.

Technologies

Wokwi

Wokwi is a simulator for prototyping and simulating IoT devices. Wokwi can create virtual microcontroller, sensor, and peripheral prototypes without requiring physical hardware. This project utilizes Wokwi to simulate the behavior of IoT devices that send sensor data and electricity consumption data, allowing testing in a virtual environment.

Mosquitto MQTT broker

For projects require real-time data monitoring like PeakFlip, seamless communication between data from heterogeneous sources is crucial. Therefore, a lightweight protocol that can be flexibly integrated with other components of the system for scalability. Thus, MQTT (Message Queuing Telemetry Transport) protocol is a feasible option due to low bandwidth requirements.

Mosquitto is an open-source message broker that implements the MQTT protocol. In this project, the Mosquitto MQTT broker facilitates communication between the simulated IoT devices and other components by enabling the efficient publishing and subscribing of data related to electricity consumption.

Node-RED

Node-RED is a flow-based development tool for wiring together hardware devices, APIs, and online services. It provides an intuitive visual interface for designing workflows using nodes. It enables custom logic, transformations, and the integration of various services, making it a key component of the data pipeline.

In PeakFlip, Node-RED is used to process and route data received from the MQTT broker, as well as performing business tasks such as getting electricity price, calculate electricity cost, check for possible alert, perform logic to activate/deactivate device actuators.

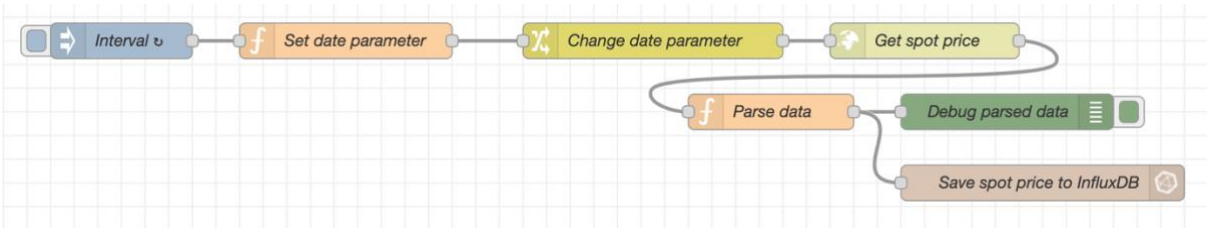


Figure 4. Node-RED flow for getting electricity price

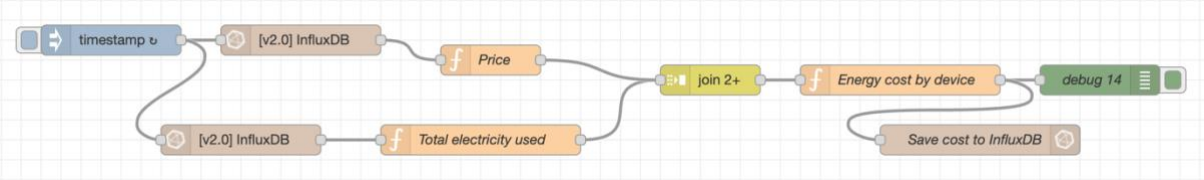


Figure 5. Node-RED flow for calculating electricity cost of devices

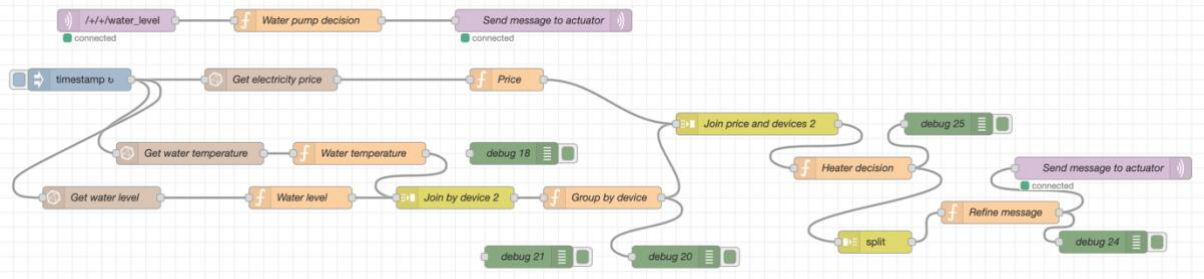


Figure 6. Node-RED flow for activate/deactivate actuators

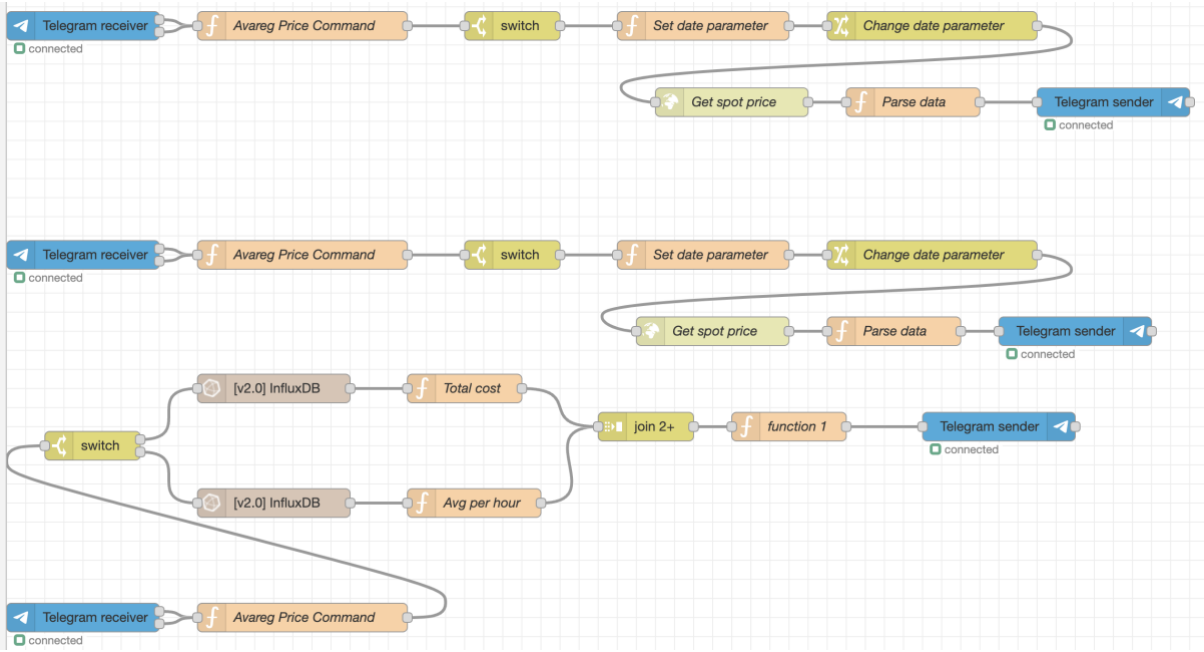


Figure 7. Node-RED flow for sending Telegram alert

InfluxDB

InfluxDB is a high-performance, open-source time-series database optimized for storing and querying time-stamped data. This project uses InfluxDB to store multiple data received from IoT devices or data calculated from Node-RED business logic flows. The ability to efficiently handle large volumes of time-series data from InfluxDB ensures quick query and making it real-time.

Telegraf

Telegraf is an agent used for collecting and processing metrics and event data. In this project, Telegraf serves as the data ingestion layer, integrating with the MQTT broker to

gather IoT data and forward it to InfluxDB. Its flexibility and lightweight architecture allow easy configuration.

Grafana

Grafana is an open-source visualization and monitoring tool used for creating interactive and customizable dashboards. PeakFlip employs Grafana to visualize electricity consumption and sensor data stored in InfluxDB. Grafana also provides dynamic dashboards based on parameters using query editor, allows users to monitor sensor data trends, track device-specific electricity usage and cumulative electricity costs in real time.

Docker

Docker is a platform that allows for the containerization of applications and dependencies, ensuring consistency across development and production environments. In this project, Docker is used to deploy Mosquitto MQTT broker, Node-RED, InfluxDB, Telegraf, and Grafana as containers. This approach simplifies the setup, improves scalability, allowing developers to reproduce and deploy system independently from different environments.

System architecture and configuration

Overall architecture

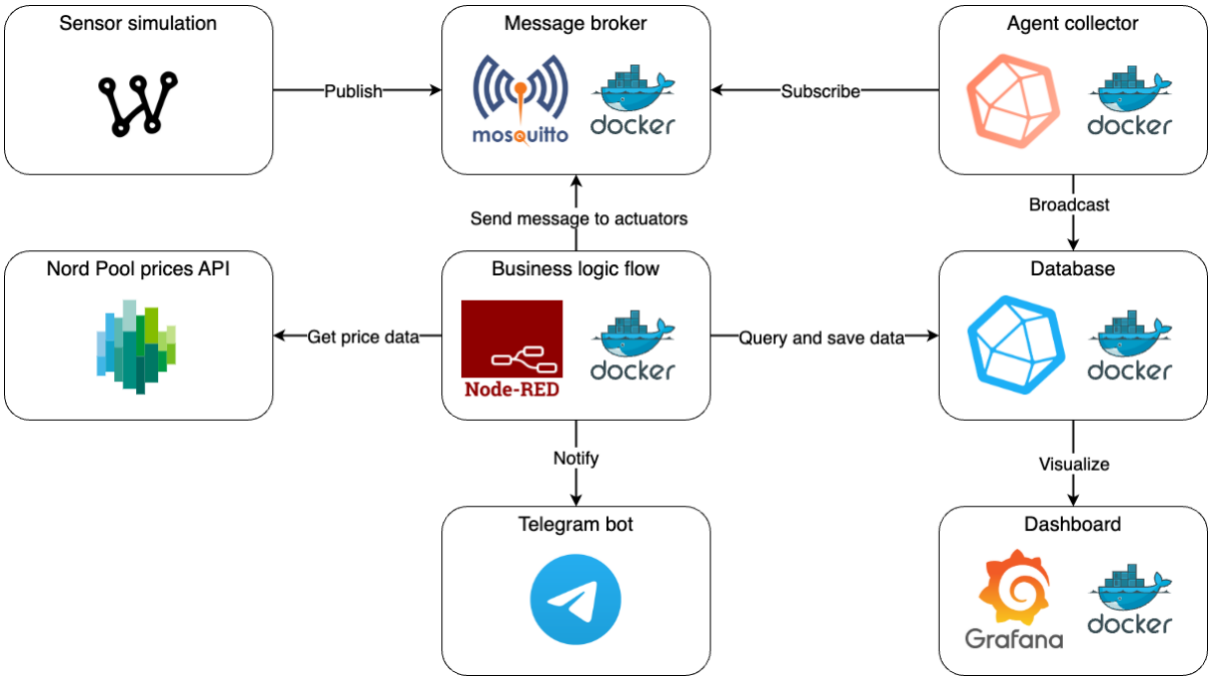


Figure 8. System architecture and technologies

Data flow in PeakFlip system:

- Sensors send data to the MQTT broker, which is subscribed by Telegraf agent. Data consumed is then saved to InfluxDB.
- Node-RED gets price data everyday from Nord Pool prices API and save to InfluxDB.
- Node-RED gets sensor data and do some logic to take appropriate actions (activate/deactivate actuators, send alerts via Telegram bot).
- Node-RED gets sensor data and electricity price data to calculate electricity cost and save to InfluxDB.
- Grafana dashboard gets data from InfluxDB and display data in real time.

The used technologies were Wokwi, Eclipse Mosquitto, Docker, Grafana, InfluxDB, Node-RED and Telegram. The data of the electricity price is received from Nord Pool, available at this URL.

System configuration

Topics for sensor data

Four types of sensor data is recorded by PeakFlip: water temperature, outside temperature, water level and electricity consumption. MQTT topics for exchanging these data are configured as follows:

- /{home_id}/{device_id}/water_temperature
- /{home_id}/{device_id}/outside_temperature
- /{home_id}/{device_id}/water_level
- /{home_id}/{device_id}/electricity_consumption

Topics for actuator action

Based on sensor data, Node-RED flows make logic to decide actions need to be taken by the actuator. Therefore, communication between Node-RED flows and devices' actuators is needed and the MQTT topics for exchanging action data are as follows:

- /{home_id}/{device_id}/pump
- /{home_id}/{device_id}/heater

Action data is exchanged in form of a binary value, 0 to deactivate the actuator and 1 to activate the actuator.

Addressing project requirements

Functional Requirements

1. **Sensor Integration:** The data from the sensors were simulated using the esp32 hardware with Wokwi. The simulation is set to produce realistic data points every second.

2. **Communication Protocols:** The MQTT protocol is implemented with different topics using Eclipse Mosquitto.

3. **Data Processing:** The system uses Node-RED as middleware for handling the logic using the incoming data and send the right message to the actuators and save the information to the database.

4. **Data Storage:** InfluxDB is used to store the necessary data with all the needed tags.

5. **Visualization:** A dashboard was created using Grafana that allows a quick view of important data including summarized statistics. Filtering is enabled by selecting the right home and device.

6. **Alerting Mechanisms:** Telegram was utilized as an easy-to-use alerting mechanism, that provides the user with all the necessary information.

Non-Functional Requirements

Portability: The algorithm is fully containerized using Docker, only the example implementation of Wokwi needs to be set up separately due to software limitations

Scalability: The system can easily support many devices in a real-world environment.

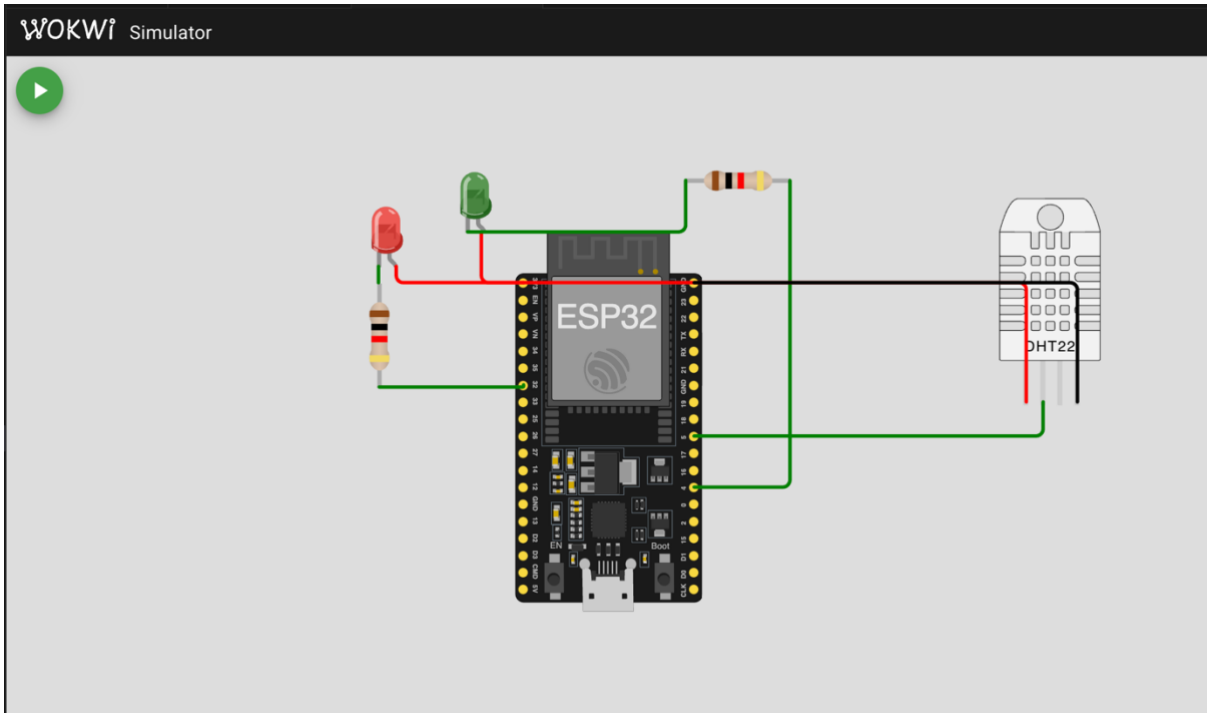
Resilience: During the development, the key principles of resilience was considered.

User-Friendly Design: A clear set-up guide is given and utilizing the graphical interface, the user has an enhanced experience.

Security: The data is password protected, which only allows access to authorized users.

Wokwi Implementation

The system is implemented using an ESP32 device, which handles receiving data from sensors and managing actuators. To visualize this implementation, the Wokwi service is utilized, simulating the physical world. This implementation runs outside of the Docker container and includes both randomized data readings and an optional physical sensor example. The system reads data from a DHT22 sensor, simulates other values such as temperature, water level, and electricity consumption, and controls devices like LEDs and water pumps based on MQTT commands. The actuators are represented by two LEDs—when an actuator (such as heating) is active, the corresponding LED turns on, and when it is inactive, the LED turns off.

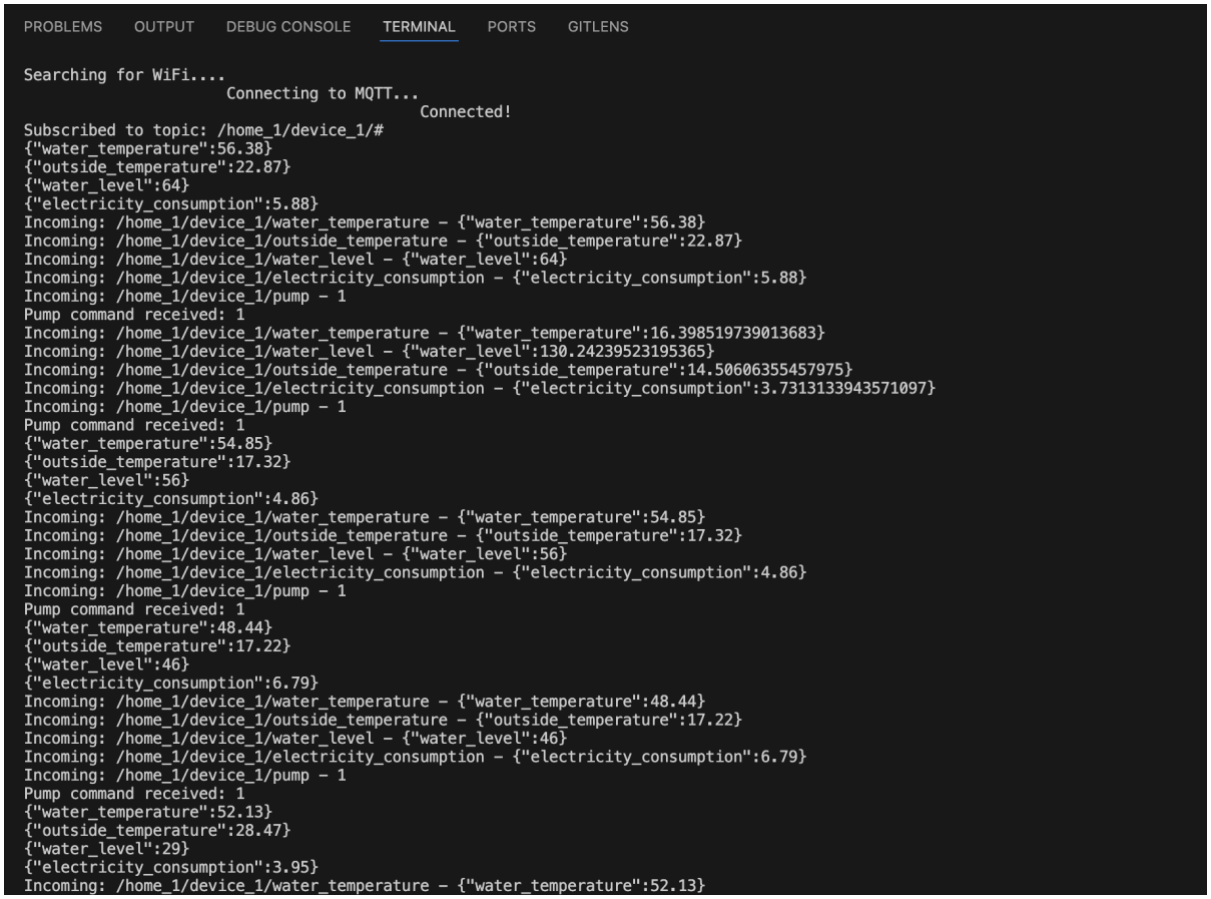


The system communicates through MQTT, publishing sensor data every second. The topics used include /home_1/device_1/water_temperature, /home_1/device_1/outside_temperature, /home_1/device_1/water_level, and /home_1/device_1/electricity_consumption. It also subscribes to dynamic topics like /home_1/device_1/#.

The hardware setup includes a red LED connected to pin 32, a green LED connected to pin 4, a DHT22 sensor connected to pin 5, and WiFi connectivity using Wokwi-GUEST. The system can handle multiple devices by renaming the Home ID (home_1) and Device ID (device_1).

To set up and run this system, first install the Wokwi Simulator in Visual Studio Code, log in to activate the license, and ensure the Docker container is running. Build the project by typing pio run in the terminal. To start the simulation, open Visual Studio Code

and choose Wokwi: Start Simulation. Clicking the start button will initiate the visualization. Connection logs, received MQTT messages, and published data can be monitored through the serial monitor.



Future work

There are numerous opportunities to enhance this system and make it more efficient and reliable. One major improvement lies in optimizing the algorithm controlling the actuators. By integrating data from all available sensors and applying more advanced logic, the system can respond more intelligently to real-time conditions. This could include the incorporation of machine learning techniques, which would allow the system to analyse historical data and identify patterns, enabling it to predict and adapt to future scenarios. Such advancements could lead to a more precise reduction in electricity consumption while maintaining system performance.

Furthermore, rigorous testing using real-world datasets and actual energy consumption trends is necessary to ensure the system achieves its intended objectives. Testing with randomized, rapidly changing values may provide initial insights but is not suitable for accurately evaluating the system's effectiveness in reducing energy consumption. Real-world data would not only validate the system's capabilities but also reveal areas requiring further refinement.

Finally, creating a physical prototype of the IoT system would be an invaluable step in testing and development. A tangible example would allow for in-depth experimentation in a controlled environment, simulating real-life usage scenarios. This approach would help identify practical challenges, verify the system's functionality, and pave the way for future improvements and potential scalability to larger systems. Combining algorithmic enhancements, robust testing, and real-world prototyping would ensure the system is both effective and practical for broader application.