ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

IoT system for real-time monitoring and analysis of domestic photovoltaic systems energy consumption

46062 David Monteiro Marques

Advisor Prof. Rui Duarte

Project report carried out under the Project and Seminar
Computer Science and Computer Engineering Bachelor’s degree
Summer Semester 2022/2023

July 2023

Instituto Superior de Engenharia de Lisboa

IoT system for real-time monitoring and analysis of domestic photovoltaic systems energy consumption

46062 David Monteiro Marques

Advisor Prof. Rui Duarte

Project report carried out under the Project and Seminar
Computer Science and Computer Engineering Bachelor’s degree
Summer Semester 2022/2023

July 2023

Abstract

The objective of this project is to design and prototype an IoT system that measures the electrical consumption of a home, supplied by photovoltaic panels. The system provides the user with simple access to energy consumption information and analytics to make more informed decisions about their energy usage, hence reducing energy waste and promoting a sustainable living, and possibly reducing electricity bills. The energy consumption is obtained from a custom wireless IoT sensor node installed on the electrical panel of a house. This node also provides electrical energy production from solar panels and environment data and supply. The sensor data is communicated over the Internet to an IoT server that hosts a database to record the measurements. An API was implemented for accessing the data storage and a simple Web application was created to provide a user-friendly interface to access the information collected, including analytics. This project concerned the selection of the hardware and software components and an implementation as proof of concept. Results demonstrate that the objectives were achieved as the proposed solution fulfills the system's requirements.

List of Abbreviations

API – Application Programming Interface.

TTL – Transistor-Transistor Logic

AC – Alternated Current

DHT – Digital Humidity and Temperature

TCP/IP – Transmission Control Protocol/Internet Protocol

HMI – Human Machine Interface

AMQP – Advanced Message Queuing Protocol

MQTT – Message Queuing Telemetry Protocol

IoT – Internet of Things

AWS – Amazon Web Services

SQL – Structured Query Language

RTU – Remote Terminal Unit

UART – Universal Asynchronous Receiver-Transmitter

NTP – Network Time Protocol

IP – Internet Protocol

HTTP – HyperText Transfer Protocol

HTML – HyperText Markup Language

HBS – Handlebars

CSS – Cascading Style Sheet

CSV – Comma Separated Values

Index

1. Introduction

Nowadays, there is more and more technology being involved in a domestic environment. Allied with growing care and conscience towards sustainability and reducing one's ecological footprint, and the rising popularity of solar energy collection, there is a trend of systems to monitor aspects of the domestic environment.

The project aims to develop a home system that utilizes sensors to monitor energy usage and provide the user with feedback, allowing them to make informed decisions about their energy usage, reducing energy waste, lowering electricity bills, and promoting sustainable living.

1.1 Motivation

The motivation behind this project is to provide the user with a system capable of presenting them with data relative to their domestic power consumption, giving them information and enabling them to make better decisions about electrical consumption and maximizing the uses of renewable energy. With increasing concerns about climate change and the environmental impact of energy consumption, there is a growing demand for more sustainable and responsible use of energy. An energy monitoring system provides homeowners with real-time data on their energy usage, allowing them to identify areas of high energy consumption and make informed decisions about reducing energy waste. Additionally, the project can help homeowners optimize their use of renewable energy sources, such as solar panels, by monitoring their performance and contribution to energy production.

1.2 Objectives

The objective of this work is to conceive and implement a project that reads the data from the electric panel through sensors and provides a system with a web application to show users their energy consumption. It will also monitor the energy production of renewable sources, such as solar panels. The project is meant to provide a low-cost, customizable, and scalable, to be applied to different scenarios in the future.

With these objectives in mind, some requirements were defined for the functioning of the project:

- Read electrical consumption from the different divisions of the house and the energy supply from the photovoltaic panels through an IoT sensor node, installed in the electrical panel of the house, sending measurements every second, equipped with a small battery in case of a power outage, and with the possibility of sending alarmistic messages (in case of outlier measurements)

- Implementation of an IoT server with the following functionalities:
    - Communication with the IoT node for the sending of data obtained from the sensors, through a broker and a defined protocol.
    - Database to store the data obtained.
    - A web API is responsible for accessing the stored data.
    - A web application to provide the user with a way to visualize all the information.

1.3 Structure of the report

This report is organized as follows. Chapter 2 provides background information on the existing solutions and the related technologies. Chapter 3 presents the Proposed System's Architecture and its implementation in Chapter 5. Chapter 6 is dedicated to discussing the results obtained and how the objectives of the project were met. This report ends with Chapter 7 which holds the conclusions of this work and discusses future work.

2 Background

In this chapter it's presented the state of the art related to the project (section 2.1) and the related technologies (section 2.2).

2.1 State of the Art on IoT Systems for Energy Consumption

In the market, there are some IoT systems to monitor domestic electricity consumption. One of them is Sense [1], which uses sensors in the electrical panel and machine learning to detect and inform the user of the consumption of the different home appliances, through a web app. However, the elevated cost, around 400€, makes it a less desirable solution. There is also Emporia [2], which also uses sensors in the electrical panel and presents the user with a web app, but without the use of machine learning. The problem with Emporia, although its lower cost compared to Sense, being around 200€, it's not very customizable, being sold in pre-arranged packs despite the user's needs.

In a more industrial context, there is VTScada [3], an HMI (Human-Machine Interface) software that allows the monitorization of industrial equipment and communication with these. This software is only used in networks of dozens of machines in multiple locations, needing a much bigger infrastructure to make it a viable option.

In the research of the many solutions to the problem, there are some evident disadvantages in the different IoT systems already present in the market:

- The elevated cost of many of them - almost all the systems studied presented costs in the high thousands of euros, making them a not so convincing options for users wanting to experiment with small-scale prototypes before committing to buying expensive equipment and software.
- Non-customizable nature - these systems are sold with predefined packs, most of the time meaning the customer is either overpaying due to not needing all of what is present in the package bought or having to search for other options due to missing some specific characteristics in the package offered by these systems.
- Many amateur projects that most of the times are not maintained or have no documentation available.

This project aims to resolve these issues by offering a low-cost solution, due to choosing cost-efficient components, and a personal system by offering a more modular solution, allowing the adding and removing of components easily.

2.2 Related Technologies

In this section, a more detailed analysis of the technologies available and relevant to the project is presented, along with some studies. It's broken down into different components, starting with the sensors (chapter 3.1), followed by the microcontroller (chapter 3.2), the message protocol and broker (chapter 3.3), the backend (chapter 3.4), sub-divided into the hosting (chapter 3.4.1), the database (3.4.2), and finally the frontend (chapter 3.5).

2.2.1 Sensors

In this project, the sensor we are focusing on is an electric current transformer and is responsible for measuring the current that passes through the cables of the electric panel.

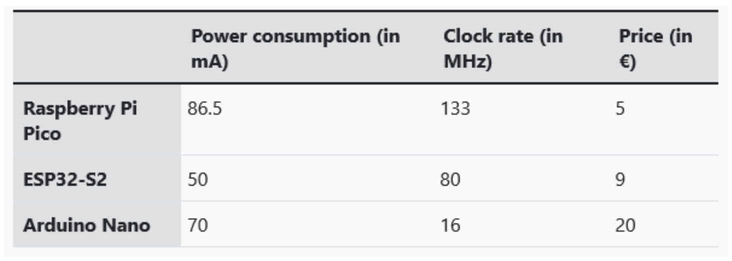
Current transformers [4] capture the magnetic field generated by the electrical current flowing through the cable in the center of it, through a magnetic circuit present inside the transformer. This way, the module transforms the current into a measurable level, which is then transformed into an output that represents the current value. In the market, there are many current transformers options, that vary in terms of cost, flexibility, and range of measurements. There are closed current transformers, which need to be installed before the installation of the cable, and split current transformers, which can be installed at any time, allowing it to be more flexible. Attached to the transformer there must be a module to transform the measurements into readable data, which is then sent through an interface. The most common module is the PZEM004T [5], which allows the data to be read through a TTL serial interface.

2.2.2 Microcontroller

Microcontrollers are used in numerous different industries and embedded systems, usually in the means of data collection and sensing. In the project, the microcontroller is used to receive the data from the sensor, treat it, and send it to the server. There are many options available, but the main ones explored were from EspressIf [6], Raspberry Pi [7], and Arduino [8].

From EspressIf, the option that most fitted the project was the ESP32-S2 chip, due to its low power consumption, numerous peripherals, wireless connectivity, and low cost. From Raspberry Pi, Raspberry Pi Pico is a microcontroller also with multiple peripherals, wireless connectivity, and low cost. From Arduino, there are multiple microcontrollers available. Despite that, most of the low-cost options do not have wireless capabilities and the ones with it are more costly. The most adequate option is Arduino Nano.

Figure 1 is a comparison between some of the microcontrollers present in the market more adequate for the project.



	Power consumption (in mA)	Clock rate (in MHz)	Price (in €)
Raspberry Pi Pico	86.5	133	5
ESP32-S2	50	80	9
Arduino Nano	70	16	20

Figure 1 - Comparisons of the different microcontrollers

To fulfill the requirements necessary for the project, the microcontroller must have Wi-Fi capabilities, to communicate with the server, low power consumption, be cheaper to run because it must be constantly connected to receive the readings, a high clock rate to be able to process the readings at an adequate rate, and low cost.

2.2.3 IoT Message Protocol and Broker

In this project the focus is on messaging protocols that work on top of TCP/IP, allowing messaging exchange between systems, and are more suitable for IoT projects. The two major examples of this type of messaging protocol are AMQP, standing for Advanced Message Queuing Protocol, and MQTT, which stands for Message Queuing Telemetry Transport.

AMQP is a general-purpose messaging protocol, mostly used in large-scale projects with a focus on security and reliability. It has features like reliable queuing, flexible routing, and various types of messaging, security, and transactions.

MQTT is a lightweight and resource-efficient messaging protocol, designed specifically for the communication between sensors and the main system. It has a fast response time, low bandwidth, and low resource usage.

MQTT only allows client/broker architecture while AMQP also allows client/server architecture. The difference is that a broker acts like a mediator between the two parties at each end of the messaging, while the server serves a single client. The messaging strategy supported by both protocols is the publish/subscribe (pub/sub). This strategy utilizes the concepts of "topics", allowing the consumer to choose what to receive, and having more flexibility in systems with many publishers and subscribers.

In this project, due to its characteristics, the most appropriate is the MQTT, due to its low resource and low latency aspects.

To interact with the MQTT protocol, there's a need for a broker. The broker acts as an intermediary between publishers and subscribers, managing message delivery and ensuring security, through security measures and access control policies, and the reliability of the protocol. Due to the inability of making large-scale tests, the decision was based on a couple of studies found on the internet. A study stress testing MQTT brokers [9] tested five popular MQTT brokers (Mosquitto, Bevywise, ActiveMQ, VerneMQ, EMQ X) in manners of latency, average computational resources used, and throughput in different environments and with different quality settings. Another study [10] benchmarked another five brokers in throughput, by simulating 25 concurrent clients each sending 10000 messages of 4Kbytes size each. Figure 2 is a graph taken from that study.

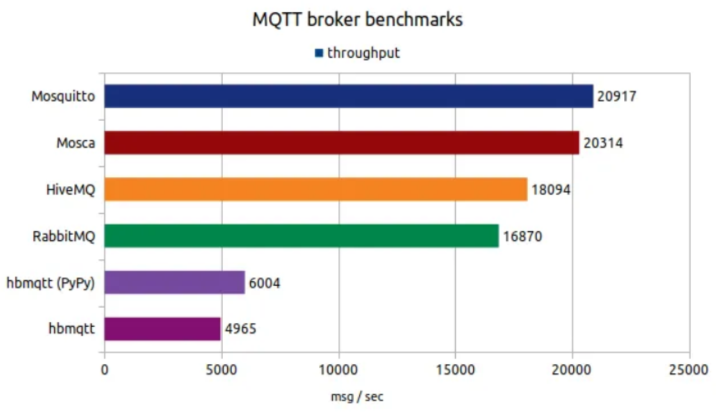


Figure 2 - Throughput evaluation of 5 MQTT brokers

2.2.4 Hosting Server

To host the backend there are two options: cloud hosting or local hosting. Cloud computing makes applications and websites accessible using cloud resources and is not deployed on a single server, while local hosting is a mechanism used to run the same but using a computer as the resource for hosting.

To cloud host there are many options like AWS [11], standing for Amazon Web Services, which is a cloud services platform that offers multiple functionalities and the ability to run applications and web servers through the cloud to host dynamic websites, Google Cloud [12], that offers the option to deploy and operate applications, allocating space to develop and run software and Azure [13], from Microsoft, which also offers a cloud platform to build, manage and deploy applications, along with a set of services including analytics, storage and much more.

Cloud hosting has its advantages, by being easier to get running, making your system readily available and having an infrastructure to resolve issues behind it but also its disadvantages, by not being very customizable, being reliable on a third party, meaning not being able to have agency if the service is down and there is usually a cost that comes with registering for these hosting services which can depend on the service.

To local host, there is a need for a dedicated machine to host the backend, which can be a computer or a Raspberry Pi. There is also a need for software and there are many options like XAMPP [14], Apache [15], nginx [16], and many others. It has the advantage of being very customizable, meaning you set up your service exactly like you want it, not being reliable on another party, and usually having little to no costs.

2.2.5 Backend

2.2.5.1 Database

There are many options to choose from when it comes to databases but are mainly divided into two options, SQL or NoSQL [17].

SQL databases, like MySQL and PostgreSQL, are more traditional databases, with a defined schema and well-structured relationships between data tables and consistency. However, due to the rigid schemas that need to be defined upfront, it's not the best option when it comes to flexibility or scalability.

NoSQL databases are databases with no defined schemas upfront, making them more flexible and easier to manipulate in relation to SQL databases. They are simpler, by avoiding SQL queries and statements, making it easier to deal with large amounts of data and some have real-time functionalities. Although all this, the limited query capabilities tend to pose some difficulty. Examples of these kinds of databases are MongoDB and RethinkDB.

Because of the nature of this project, a database that can deal with large amounts of data and being easily accessible is a necessity, making NoSQL databases much more appealing.

2.2.5.2 Web API

The web API is needed to provide the user with a way to interact with the information stored in the database. There are many frameworks to build APIs with but mainly two were taken into account due to the familiarity with these. These two were Spring [18] and Express [19].

Spring framework is a Java-based application framework, a very modular and extensible framework through numerous selectable components and extensions. It also provides Dependency Injection and Inversion of Control, making it easier to test and more maintainable. Despite these advantages, it has a somewhat steep learning curve and its layered architecture, and many features may introduce some performance overhead compared to other frameworks.

Express is a framework for the Node.js environment, designed to build robust and scalable web APIs, and known for its simplicity and extensive middleware support. Like Spring framework it's also very modular due to the easy integration with other Node.js modules. However, due to the lack of built-in features, it can have some potential security risks, needing additional middlewares to ensure security.

For the project, due to the simplicity and familiarity with the Node.js environment and Express framework, it was the chosen framework.

2.2.6 Frontend

To present all the information to the user there is the need for a web application, consisting of a frontend application. To create simple web applications only base technologies such as

HTML and CSS, that are responsible for creating the structure and content of the pages, and are used for styling and layout, respectively. To create more complex and structured web applications, frontend frameworks provide the tools necessary for it, such as component-based architecture, state management, routing, and more. There are many options of frontend frameworks suitable, but the most popular are React [20], Angular [21], and Vue.js [22].

React framework is an open-source framework that uses virtual Document Object Model, and it's a good option for a high-traffic and steady platform. Angular framework is a popular frontend framework. It has two-way data binding, which is exclusive to it, which means there is synchronization between the view and the model, making it great for web applications with developing models. Vue.js framework is a simple frontend framework that has a small size and two main benefits: visual Document Object Model and component-based. It also employs two-way data binding, which makes it a versatile framework and a good option for both dynamic and simple development.

3 Proposed IoT System Architecture

This chapter describes the proposed architecture for the IoT system. It starts by giving a high-level description of the architecture (section 4.1), followed by the description of the two main components: the Sensor Node (section 4.2) and the IoT Server (section 4.3). In the end, it is specified the Personalization and Scalability (section 4.4) capabilities of the project.

3.1 High-level Description

The system is composed of 2 major sub-systems: the sensor node and the IoT server. The sensor node is composed of the sensors and the microcontroller.

The system reads the value of the electrical current from the different parts of the house through the sensors in the cables in the electric panel and the humidity and temperature from an ambient sensor. Through the microcontroller, it sends the readings by Wi-Fi to the server, in which a broker forwards the message to the subscriber, who treats the information and saves it in the database. The Web API accesses the data present in the database and makes it available to the Web App, which presents it to the user. Figure 3 gives a visual representation of the project.

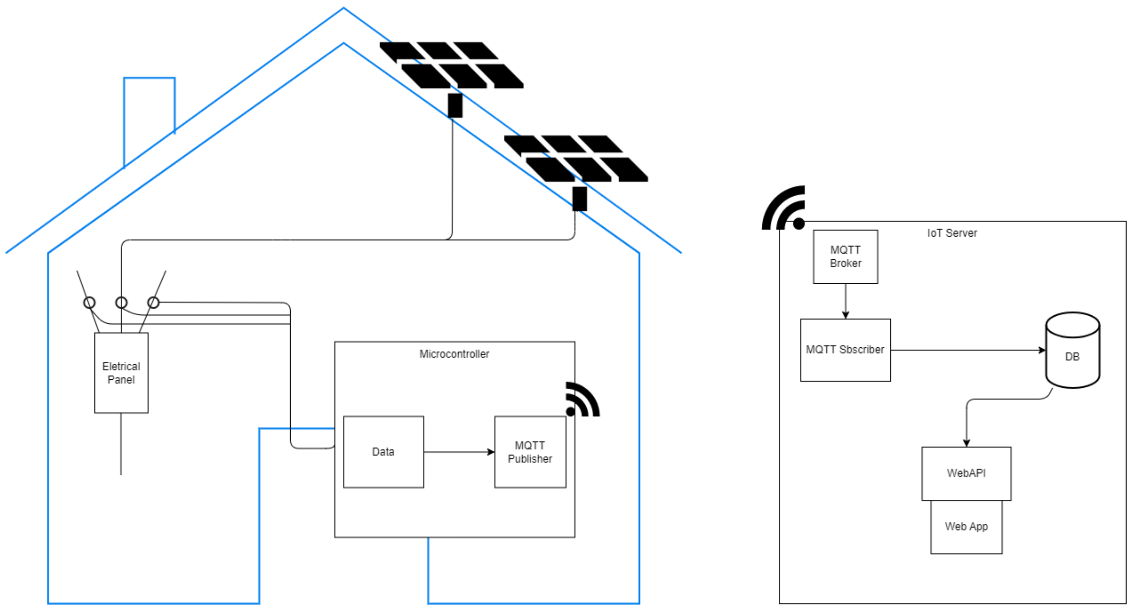


Figure 3 - Proposed architecture of the system

In Figure 3, it can be seen the electrical panel, with the sensors attached and connected to the microcontroller. The microcontroller receives the data and publishes it through the MQTT publisher, using the Wi-Fi network. The IoT Server receives this data in the MQTT broker, which is going to route it to the respective subscriber. This subscriber will store the data in the database (named DB in the figure), making it available to the Web API for access.

3.2 IoT Sensor Node

The IoT Sensor Node is responsible for acquiring the data from the sensors, via an electrical interface, and communicating their data to a remote server on the Internet. This sub-system is composed of two components: the sensors and the microcontroller. One of the sensors, a split transformer, connects to an energy calculator circuit (PZEM004T), which is responsible for providing readings of the voltage and current that passes through the cable it's attached to. It reads multiple data, such as voltage, current, power, energy, frequency, and power factor. It is expected to have one sensor for each division that the user wants to get information from. To get the readings from the module, the microcontroller sends a command to the sensor that asks for the specific metric. The other sensor is a DHT, an ambient sensor, which gets the readings of the humidity and temperature.

The sensors are connected to a microcontroller, an ESP32-S2, that gets the readings and compiles the information, adding a timestamp to the information sent, obtained through a request to an NTP server before sending it via MQTT protocol through Wi-Fi to the MQTT broker present in the server.

The sensor node will also have an alarmistic ability. The microcontroller, after receiving the readings from the current sensor, will be able to determine if the current received falls outside of the boundaries of what's the normal functioning of the domestic electrical current, being able to react, either by switching off the whole house power supply or alerting the user through some way.

3.3 IoT Server

In the server, to receive the messages sent from the microcontroller there is a broker, Mosquitto, that will register the capture of these messages and will forward them to the respective MQTT client (or subscriber). The client, after receiving the data, stores it in the NoSQL RethinkDB.

There is also a Web API, implemented in the Node.js environment with the use of the Express framework, responsible to provide an interface that allows access to the data stored in the database accessible through HTTP requests. The Web App, which gives the user an interface to request the different data, is implemented with the use of HTML, CSS and with HBS templating language.

Ideally, the server will be locally hosted in a RaspberryPi Model 3, a single-board computer.

3.4 Personalization and Scalability

The big difference between this project from the ones already on the market is that it's highly customizable and scalable to suit the needs of its user.

The ability to add and remove different types of sensors of the sensor node and being able to put multiple of these sensor nodes throughout the house makes the system very customizable, letting the user make decisions about what sensors they want.

The scalability comes both from the microcontroller and from the PZEM module itself. The microcontroller has available several ports to which different peripherals can be connected to, making the adding and removing of said sensors very easy. The PZEM module has a serial interface port. The microcontroller only has two of these ports, however the module has the ability to be attributed a fictional address, allowing multiple of these modules to be connected to the same port. The electrical interface of the serial interface is made with *open-drain* optocouplers which allow to have multiple devices sharing the bus without any conflicts. This way, the requests for readings made from the microcontroller are directed to a specific address, making the module with that address the only one that responds to that command. This can be defined in laboratory or in production, although it can also be defined after to accommodate changes in the needs of the user.

4 Implementation

In this chapter we will present the details of the implementation of the different components of the project, starting with the Sensor Node (section 5.1) and followed by the IoT server (section 5.2). The IoT server is subdivided into two sub-sections to talk about the Message Broker and Client (section 5.2.1), the Database (section 5.2.2), the Web API (section 5.2.3), and the Web App (section 5.2.4).

4.1 Sensor Node

There are two sensors that are part of the sensor node: the current sensor, composed of a current transformer and the PZEM module and the DHT sensor.

The 100 ampere external split transformer is implemented around the energy cable that goes to the division of our choice. That sensor is then connected to the module, the PZEM004T-V3 through cabling. This module can measure the voltage, current, power, energy, power factor, and frequency. It's connected to the microcontroller through an UART port, due to the physical layer communication protocol of the module, being UART to RS485 communication interface. The application layer communication protocol used is the Modbus-RTU. The DHT sensor is connected to the microcontroller through a GPIO port.

The microcontroller used is the ESP32-S2, more specifically an ESP32-S2 Saola 1 board [23], due to having Wi-Fi capabilities, low power consumption, and a lower power mode that can be activated to further reduce the energy consumption. It was programmed using the Arduino framework due to its simplicity and many resources available, mainly libraries. Figure 4 is a schematic of the sensor node.

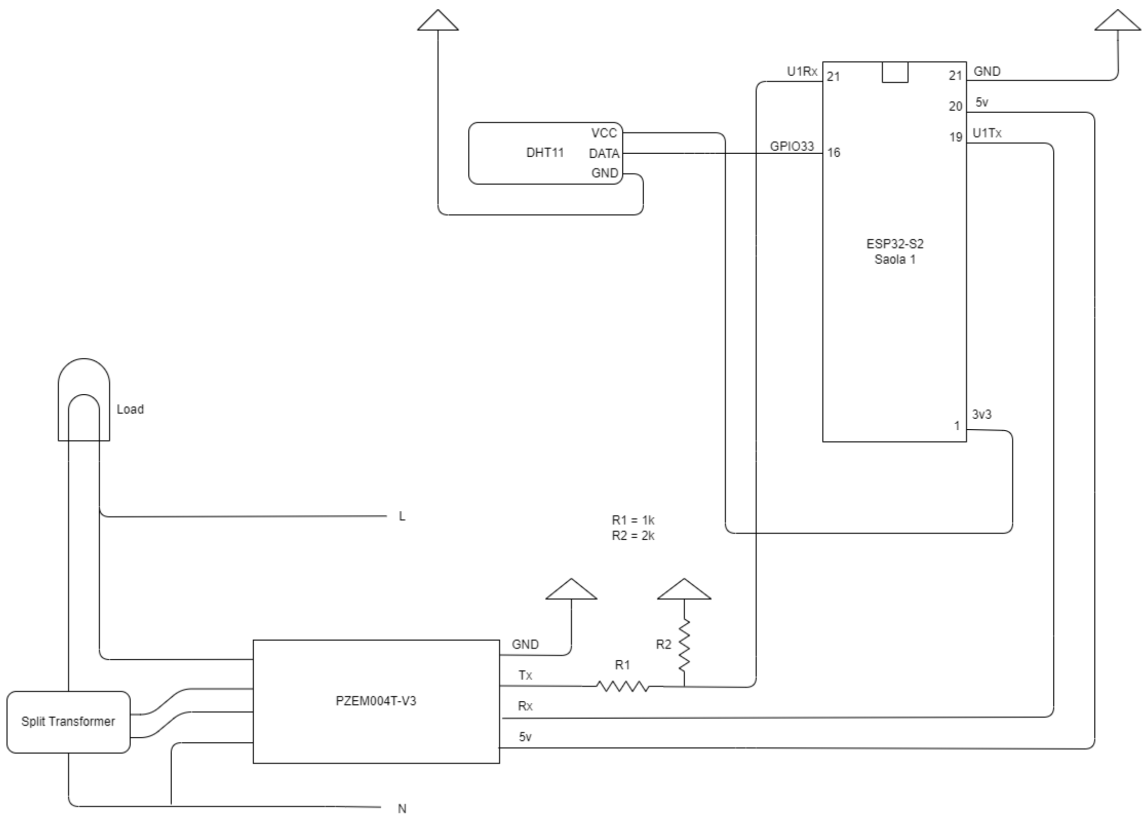


Figure 4 - Schematic of the node sensor

The PZEM004T-V3 module is connected to the UART port present in the microcontroller through cabling. However, the voltage sent from the module is 5 volts while the microcontroller is at 3.3 volts. To fix this problem, the signal coming from the module is run through two resistors, one with 1k ohm in series and one with 2k ohm in parallel, effectively reducing the voltage, In the figure it is shown as the R1 and the R2 components, being the resistors of 1k ohm and 2k ohm respectively. On the other way around, the module can read the signal sent from the microcontroller, at 3.3 volts. The DHT sensor does not have this problem, being simply connected to its port through a cable.

The microcontroller was programmed to send the commands needed to obtain the readings of the PZEM module with the help of a library designed to send data through Modbus via RTU, which is the case, called ModbusMaster. To add to the data read from this sensor, it is also fetched a timestamp through a request to an NTP server, specifically to a server based in Instituto Superior Técnico due to its proximity and to obtain the lowest latency possible, and a string indicating the type of sensor responsible for the data, to help with organization forward and with the possibility of adding more types of sensors. For the NTP request is used the NTPClient library. All this information is compiled into a JSON object with the help of ArduinoJson library and sent to the server. The same process is done to the readings of the DHT sensor, being received by sending commands to the sensor, without the need of the Modbus library despite its communication process also being serial, due to its single-wire serial interface.

To send the readings to the server it is used the MQTT protocol. It's used a library, PubSubClient, which its setup involves defining a connection to the IP and port of the broker present in the server to establish a connection. After the data is collected and treated, it's sent to the topic which is the division of the house being monitored. In Figure 5 is a flowchart of the functioning of the sensor node.

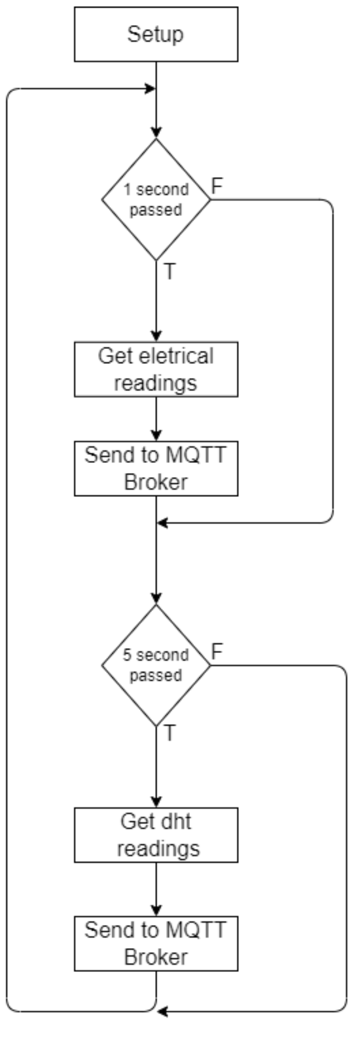


Figure 5 - Flowchart of the sensor node

4.2 IoT Server

The initial idea for the server is to be implemented in a single-board computer due to the low power consumption, necessary because of the continuous running of this server, compared to a normal computer, and the ease of deploying new software making it a great environment for testing. Figure 6 is a visual representation of the architecture of the IoT server.

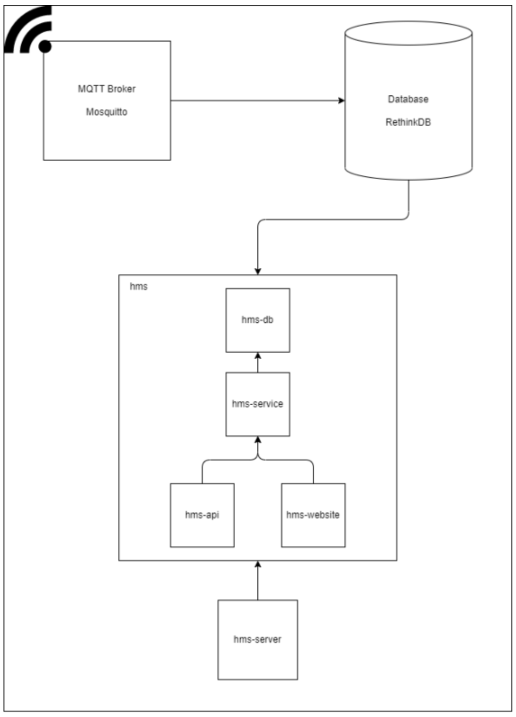


Figure 6 - Architecture of the IoT server

4.2.1 Message Broker and Client

The broker implemented was the Mosquitto Broker, created by Eclipse. It was chosen due to its simplicity in implementation and its low resource necessity.

The client was implemented in JavaScript, due to the familiarity with this language and the availability of many libraries to help with the development. To make the bridge with the broker it was used a library called mqtt. This client is connected to the broker through its IP and port and subscribed to the topic of the division being monitored, receiving the readings this way.

4.2.2 Database

The database chosen was RethinkDB [24], a NoSQL database. It was chosen a NoSQL database due to the time-sensitivity of the data in the project, receiving readings from the sensors every minute, and the amount of data also being handled. RethinkDB has functionalities such as automatic data sharding and real-time updates. In the database each monitored division will have its own RethinkDB-database, allowing the data to be more organized. All of these will also have a table for each type of sensor, allowing the addition of more types of sensors along the way at the user's discrepancy. In each of these tables there will be added all the readings done by each of the sensors, along with the timestamp of when the reading was done in the form of a string, in a JSON object, allowing easy access to a specific metric.

4.2.3 Web API

The Web API is implemented in the Javascript programming language on the Node.js environment, using the Express framework and the RethinkDB official driver for the Javascript language. It is organized through three distinct layers:

- Data Access Layer: This layer implements all the methods that interact directly with the databases, either it be to access the data or store new data, with the help of the driver provided by RethinkDB.
- Service Layer: The service layer helps with the business logic of the API, and although not currently doing so, it also helps with the validation of the request and the error handling, bridging the gap between the API itself and the data stored in the database and only accessed by the Data Access Layer, doing it by calling the methods provided by this layer.
- Web API Layer: Finally, this layer is the public one, and establishes the endpoints that provide the user with the respective data when HTTP requests are done to them. It provides endpoints that allow the user to look at all the sensor information regarding a specific division of the house or only a specific reading of it and allows the user to specify a specific hour to filter from, and hourly and daily averages from all the readings done by the sensors, through query parameters in the URLs and methods available in the service layer. It also provides the user with the ability to export all the data, filtered or not filtered, to a CSV file. This is useful as these analytics can be used to train algorithms and neural networks, a future work to be implemented in the project.

4.2.4 Web Application.

To create the web application, there had to be implemented two main cornerstones:

- The website layer: Parallel to the Web API layer, the website, also written in Javascript, was implemented to make the connections between what was asked of the IoT server to what the user is presented with, visually, mimicking the endpoints of the web API but having it be independent. The website layer uses the service layer to obtain the information requested by the user and present it, with the help of HTML, CSS, and Handlebars templating language, in a visually appealing and intuitive way.
- The views: The views are a group of Handlebars files that are responsible for transforming the information that is passed on to them to a more visually pleasant format. There is one for each different endpoint accessible The Handlebars language is a templating language that allows the generation of dynamic HTML pages, combining data passed to and templates. Also using CSS, a styling language for HTML pages, it was

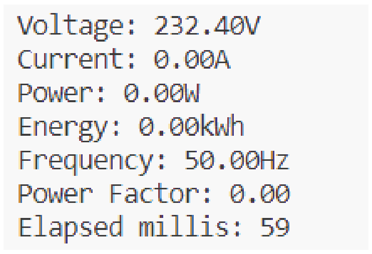
possible to design a simple visual experience for the user without losing out on any data presentation.

5 Evaluation

To validate the correct functioning of the multiple components of the project some tests were created and executed.

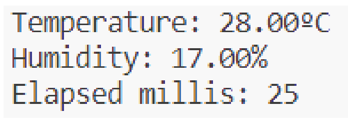
5.1 IoT Sensor Node

To test the sensor node was done small tests to the electric current sensor (the current transformer, and the PZEM004T module) and the ambient sensor (DHT), separately, registering the different readings that are obtained and the maximum sample rate at which it could obtain them, in each sensor, as seen in the Figures 7 and 8.



```
Voltage: 232.40V
Current: 0.00A
Power: 0.00W
Energy: 0.00kWh
Frequency: 50.00Hz
Power Factor: 0.00
Elapsed millis: 59
```

Figure 7 - Example of a message obtained from the electrical current sensor and the time needed to obtain them, in milliseconds.



```
Temperature: 28.00ºC
Humidity: 17.00%
Elapsed millis: 25
```

Figure 8 - Example of a message obtained from the ambient sensor and the time needed to obtain them, in milliseconds.

It was also tested the alarmistic ability of the system, which consists of the microcontroller turning on the built-in led for 5 seconds and returning to normal functioning after. After receiving the readings, the microcontroller checks to see if the voltage value read falls between the limits set by the user. To make the electrical current lower it was used 4 Zener diodes. This way, the current passing would be below 235 volts, so the limit was set there. Figure 9 shows the normal functioning of the system, with a multimeter to show the value of the electrical current passing through, and Figure 10 shows the alarm produced by the microcontroller.

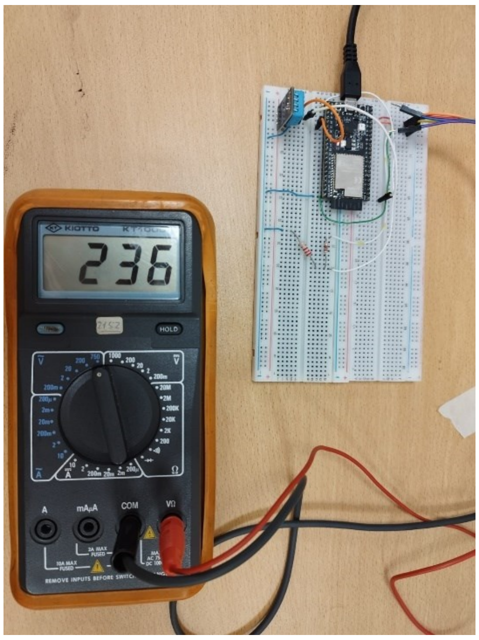


Figure 9 - Normal functioning of the system

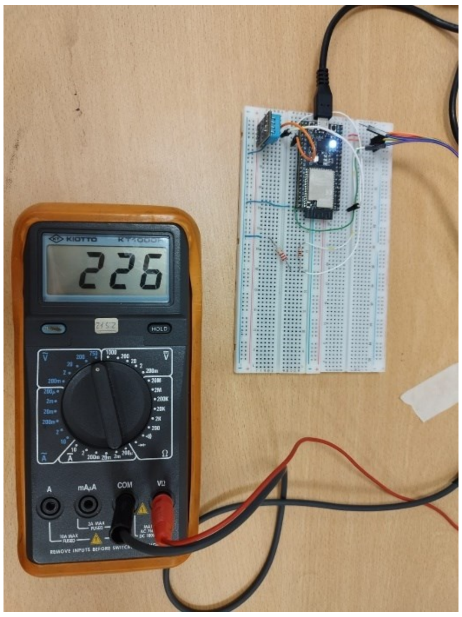
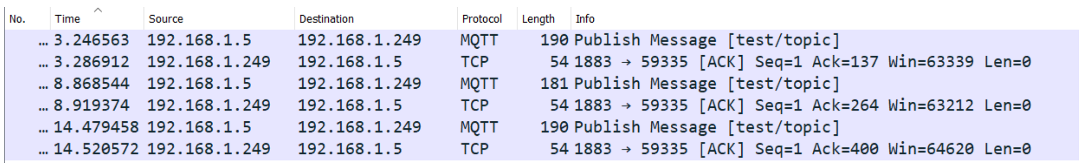


Figure 10 - Alarm notification of the system

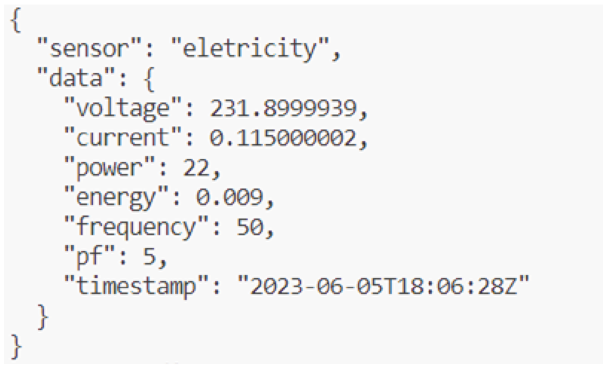
It was also the connectivity of the microcontroller. In this case, the Mosquitto broker was installed on a Windows machine. To visualize the result, it was used a packet-analyzer program called Wireshark. Figure 11 is an image of the packets sent in the communication between the broker and the microcontroller.



```
No.   Time      Source         Destination    Protocol  Length  Info
… 3.246563   192.168.1.5    192.168.1.249  MQTT       190 Publish Message [test/topic]
… 3.286912   192.168.1.249  192.168.1.5    TCP         54 1883 → 59335 [ACK] Seq=1 Ack=137 Win=63339 Len=0
… 8.868544   192.168.1.5    192.168.1.249  MQTT       181 Publish Message [test/topic]
… 8.919374   192.168.1.249  192.168.1.5    TCP         54 1883 → 59335 [ACK] Seq=1 Ack=264 Win=63212 Len=0
… 14.479458  192.168.1.5    192.168.1.249  MQTT       190 Publish Message [test/topic]
… 14.520572  192.168.1.249  192.168.1.5    TCP         54 1883 → 59335 [ACK] Seq=1 Ack=400 Win=64620 Len=0
```

Figure 11 - Wireshark capture of the test of connectivity to the MQTT broker

The whole sensor module was tested after setting it up and connecting it to a small fan, to simulate the electrical cable of the division in the electrical panel. In the test it was possible to see the sensor reading all the necessary information and how it is sent to the broker. Figure 12 is an example of a message sent from the sensor node.



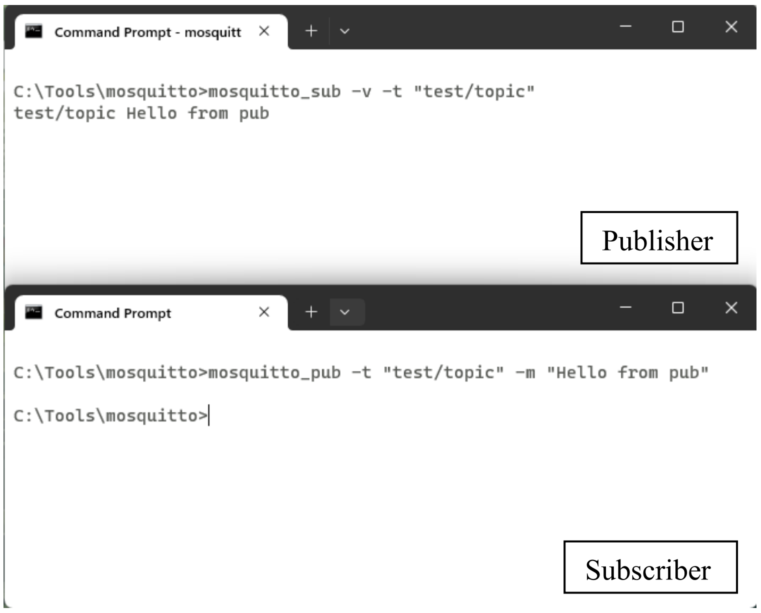
```
{
  "sensor": "eletricity",
  "data": {
    "voltage": 231.8999939,
    "current": 0.115000002,
    "power": 22,
    "energy": 0.009,
    "frequency": 50,
    "pf": 5,
    "timestamp": "2023-06-05T18:06:28Z"
  }
}
```

Figure 12 - Example message from the sensor node to the MQTT broker

This ensures the sensor node can communicate with the IoT server, with the sample rate decided.

5.2 IoT Server

For the IoT server it was tested on a personal computer running the Mosquitto Broker, the database, RethinkDB, in a docker container, and running the web API and web app locally. It was first installed and tested the Mosquitto broker, and the test was made using the command line to send simple messages from a test publisher to a test subscriber, verifying the correct functioning of the broker. Figure 13 is a demonstration of this small test.



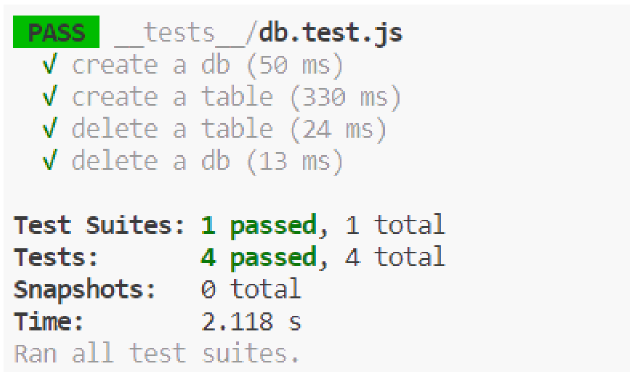
```
C:\Tools\mosquitto>mosquitto_sub -v -t "test/topic"
test/topic Hello from pub
```

```
C:\Tools\mosquitto>mosquitto_pub -t "test/topic" -m "Hello from pub"

C:\Tools\mosquitto>
```

Figure 13 - Test of the Mosquitto broker using the command line**.**

It was also tested the implementation and configuration of the database. Firstly, it was created the Data Access Layer (hms-db) with the use of the RethinkDB driver. Methods to create and delete databases, as well as to create and delete tables in said databases were implemented and tested with the help of Jest, a Javascript testing framework. Figure 13 is the result of said testing.

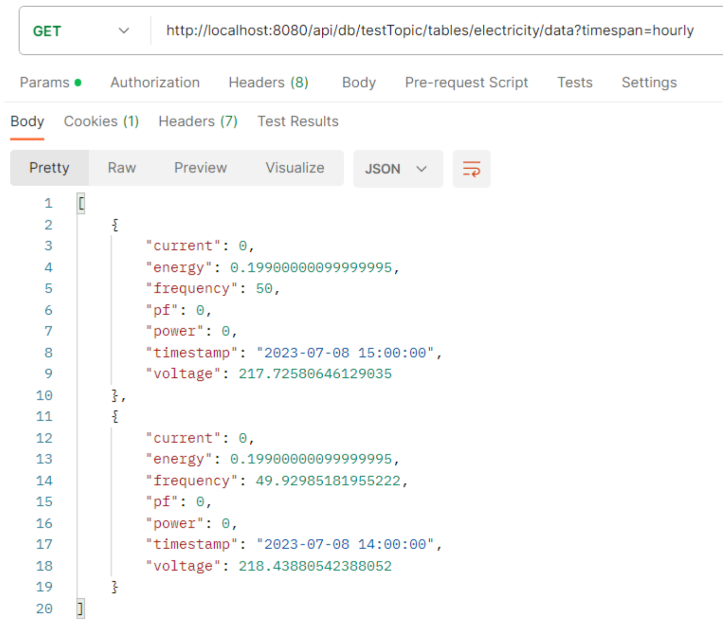


```
PASS  __tests__/db.test.js
  √ create a db (50 ms)
  √ create a table (330 ms)
  √ delete a table (24 ms)
  √ delete a db (13 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        2.118 s
Ran all test suites.
```

Figure 14 - Jest tests of the data access layer

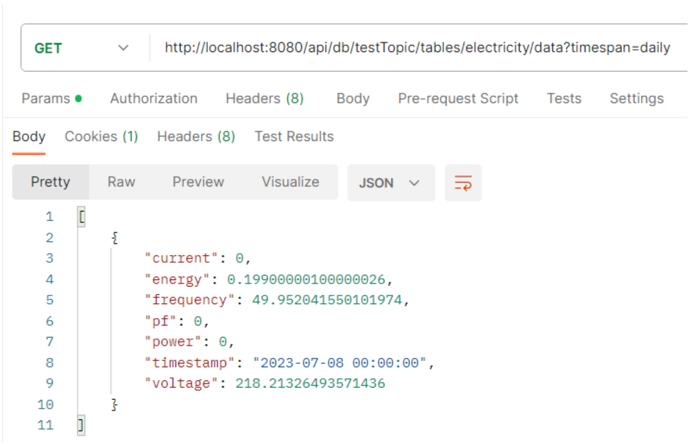
It was also tested the analytics. Both the web API and the web app provide the user with the ability to get the median of a specific timespan, daily or hourly. To test these filters, it was used a tool called Postman, a platform that allows testing APIs by performing HTTP requests. Readings of a small fan were collected in two different hours to have some data to apply these filters to, testing the methods responsible for making these calculations. Figure 15 shows the data returned with the hourly median and Figure 16 shows the daily median.



```
http://localhost:8080/api/db/testTopic/tables/electricity/data?timespan=hourly
```

```
[
    {
        "current": 0,
        "energy": 0.19900000099999995,
        "frequency": 50,
        "pf": 0,
        "power": 0,
        "timestamp": "2023-07-08 15:00:00",
        "voltage": 217.72580646129035
    },
    {
        "current": 0,
        "energy": 0.19900000099999995,
        "frequency": 49.92985181955222,
        "pf": 0,
        "power": 0,
        "timestamp": "2023-07-08 14:00:00",
        "voltage": 218.43880542388052
    }
]
```

Figure 15 – Return data with the hourly filter



```
http://localhost:8080/api/db/testTopic/tables/electricity/data?timespan=daily
```

```
[
    {
        "current": 0,
        "energy": 0.19900000100000026,
        "frequency": 49.952041550101974,
        "pf": 0,
        "power": 0,
        "timestamp": "2023-07-08 00:00:00",
        "voltage": 218.21326493571436
    }
]
```

Figure 16 – Returned data with the daily filter

6 Conclusions and Future Work

6.1 Conclusions

The project achieved most of its core functionalities, such as the ability to program the embedded system to read data from multiple sensors and present it to the user through a Web API and Web App. Experimental results demonstrate the correct operation of its components. Nevertheless, there needs to be studied an alternative to the PZEM module due to having manufacturing flaws discovered when in the development of this project. The presentation can also be improved.

6.2 Future Work

The future work outlined for this project is to recreate the user experience, making a more complete Web API, with more options for filtering data, and a more appealing Web Application, not only visually but also with more functionalities, such as graphical views and real-time data view. Also important would be the implementation of some authentication and encryption of the data being passed through Wi-Fi, to protect the data privacy of the user.

Another possible future work would be the implementation of neural networks, to provide the user with more analytics and the ability to preview the future electric consumption.

References

[1] Sense, “Sense.com - The Sense Home Energy Monitor”.

[2] Emporia, “Emporia: Revolutionizing Home Energy”.

[3] VTScada, “VTScada by Trihedral - VTScada Instantly intuitive SCADA HMI Software”.

[4] EletronicsTutorials, “Current Transformers Basics and the Current Transformer”.

[5] InnovatorsGuru, “PZEM-004T V3 Module | Arduino & NodeMCU Code , Circuit, Pinout and Library”.

[6] EspressIf, “Development Boards | Espressif Systems”.

[7] Raspberry Pi, “Buy a Raspberry Pi - Raspberry Pi”.

[8] Arduino, “Kits --- Arduino Official Store”.

[9] B. Mishra, B. Mishra and A. Kertesz, “Stress-Testing MQTT Brokers: A Comparative Analysis of Performance Measurements,” 2021.

[10] F. Mütsch, “Basic benchmarks of 5 different MQTT brokers,” 2019.

[11] Amazon, “Web Hosting - Amazon Web Services (AWS)”.

[12] Google Cloud, “Products and Services | Google Cloud”.

[13] Microsoft Azure, “Cloud Computing Services | Microsoft Azure”.

[14] Apache Friends, “XAMPP Hosting”.

[15] Apache Software Foundation, “About the Apache HTTP Server Project - The Apache HTTP Server Project”.

[16] nginx, “nginx”.

[17] B. Andreson and B. Nicholson, “SQL vs. NoSQL Databases: What's the Difference?,” 2022.

[18] Spring, “Spring | Home”.

[19] Express, “Express - Node.js web application”.

[20] React, “React”.

[21] Angular, “Angular”.

[22] Vue.js, “Vue.js - The Progressive JavaScript Framework”.

[23] EspressIf, “ESP32-S2-Saola-1 --- ESP-IDF Programming Guide latest documentation”.

[24] RethinkDB, “RethinkDB”.

[25] K. Matthews, “MQTT: A Conceptual Deep-Dive,” 2020.

[26] Eclipse, “Documentations | Eclipse Mosquitto”.

[27] Inbound Square, “AMQP vs. MQTT: A deep dive comparision”.

[28] Coursera, “SQL vs. NoSQL: The Differences Explained + When to Use Each,” 2023.

[29] A. Pattakos, “Angular vs React vs Vue 2023,” 2023.

[30] P. Peña and M. Legaspi, “Understanding the UART,” 2021.

[31] K. Sookocheff, “How Does NTP Work?,” 2021.

[32] ArduinoJson, “ArduinoJson: Efficient JSON serialization for M«embedded C++”.

[33] N. O'Leary, “Arduino Client for MQTT”.