



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
Department of Electronic, Telecommunications and Computers
Engineering

Home Energy Management System

Paulo Miguel Monteiro Rodrigues | 47118

Carlos Miguel Freire Santos | 45938

Supervisor | Professor Rui António Policarpo Duarte

Examination Committee

Chairperson: Prof. Pedro Sampaio

Supervisor: Prof. Rui Duarte

Member of Committee: Prof. Pedro Miguens

Final Report for the Curricular Unit of Final Course Project of Bachelor's
Degree in Electronic, Telecommunications and Computers Engineering

September 2024

We declare that this document is an original work of our authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the *ISEL*

Acknowledgments

First and foremost, we would like to express our deepest gratitude to everyone who has supported us throughout this project. We immensely thank our supervisor, Professor Rui Duarte, for his invaluable guidance, support, and encouragement throughout the research process. His expertise and insight were instrumental in shaping this project.

We would also like to thank our closest friends and families for their comprehensive support and understanding during this journey. Additionally, we extend our appreciation to our colleagues and peers for their constructive feedback and encouragement, which significantly enhanced our work. Special thanks to our colleague João Correia for his invaluable contributions.

Finally, we would like to express our deep gratitude to our girlfriends, Mariana Fonseca and Inês Viola, for their significant help and support during the hard times that followed our involvement in this project. Their assistance was invaluable to the success of our work.

Thank you all for being an integral part of this journey.

*"Engineering is achieving function while avoiding failure."
- Henry Petroski*

Abstract

In pursuit for sustainable energy solutions, European energy cooperatives such as REScoop advocate for efficient systems that optimize home energy consumption. This project contributes to advance open-source residential systems integrating sensors to monitor energy usage. The system provides users with insights to facilitate informed decisions, minimizing energy waste and reducing its consumption. Motivated by existing closed-source solutions and products from companies like Shelly, and Tuya, this project aims to design and implement an open-source, expandable household solution to control appliances and monitor energy consumption. Key components of the proposed solution include low-power and small footprint microcontrollers like ESP32-S2 or ESP32-C3, environment (temperature, humidity and luminosity) and energy consumption sensors, and the Home Assistant application on a Raspberry Pi 5 ("Cofy-Box"). The prototype of the proposed system architecture shows real-time energy analysis and remote home appliance control, contributing to efficient energy management in domestic environments.

Keywords

- Renewable energy
- Energy efficiency
- IoT (Internet of Things)
- Smart home
- Raspberry Pi
- ESP32 microcontroller

Resumo

Em busca de soluções de energia sustentável, cooperativas de energia europeias como a REScoop defendem sistemas eficientes que otimizem o consumo de energia doméstica. Este projeto contribui para o avanço de sistemas residenciais de código aberto, integrando sensores para monitorizar o uso de energia. O sistema fornece aos utilizadores dados para facilitar decisões informadas, minimizando o desperdício de energia e reduzindo o seu consumo. Motivado por soluções e produtos existentes de código fechado de empresas como Shelly e Tuya, este projeto visa projetar e implementar uma solução doméstica expansível e de código aberto para controlar aparelhos e monitorizar o consumo de energia. Componentes chave da solução proposta incluem microcontroladores de baixo consumo e compactos, como o ESP32-S2 ou o ESP32-C3, sensores de ambiente (temperatura, humidade e luminosidade) e de consumo de energia, e a aplicação Home Assistant em um Raspberry Pi 5 ("Cofy-Box"). O protótipo da arquitetura do sistema proposto mostra a análise de energia em tempo real e controlo remoto de eletrodomésticos, contribuindo para uma gestão eficiente da energia em ambientes residenciais.

Palavras-chave

- Energia Renovável
- Eficiência Energética
- Internet das Coisas
- Casa Inteligente
- Raspberry Pi
- ESP32 microcontrolador

Table of Contents

ACKNOWLEDGMENTS.....	V
ABSTRACT	VII
RESUMO.....	VIII
TABLE OF CONTENTS	IX
LIST OF FIGURES.....	XI
LIST OF TABLES	XIII
1. INTRODUCTION.....	1
1.1. MOTIVATION	1
1.2. OBJECTIVES	1
1.3. SYSTEM REQUIREMENTS.....	1
1.4. REPORT OUTLINE.....	2
2. BACKGROUND ON HOME AUTOMATION.....	3
2.1. RELATED WORKS.....	3
2.1.1. REScoopVPP	3
2.1.2. Shelly.....	4
2.1.3. Tuya.....	5
2.1.4. Other Solutions.....	5
2.1.5. Comparison of Available Solutions.....	5
2.2. RELEVANT TECHNOLOGIES	5
2.2.1. Single Board Computer.....	6
2.2.2. Microcontroller.....	7
2.2.3. Temperature and Humidity Sensor.....	8
2.2.4. Power Consumption Sensor	9
2.2.5. Light Sensor	10
2.2.6. Electronic Switching Device.....	11
2.2.7. Domotic Application	11
2.2.8. IOT Communication Protocol.....	11
3. PROPOSED HOME AUTOMATION SYSTEM.....	13
3.1. SYSTEM OVERVIEW	13
3.2. COFY-BOX.....	14
3.3. COFY-COOKIE.....	14
3.3.1. ESP32-S2	14
3.3.2. ESP32-C3	15
3.3.3. LDR.....	15
3.3.4. DHT11.....	17
3.3.5. PZEM-004T V3.....	18
3.3.6. Eletromechanical Relay.....	20
3.3.7. LED Light	22

4. COFY-COOKIE IMPLEMENTATION	23
4.1. HARDWARE IMPLEMENTATION	23
4.2. FIRMWARE	25
4.3. WI-FI COMMUNICATION	25
4.4. MQTT PROTOCOL	25
5. COFY-BOX IMPLEMENTATION	27
5.1. HARDWARE IMPLEMENTATION	27
5.2. RASPBERRY SETUP.....	27
5.3. HOME ASSISTANT SETUP	27
5.3.1. Configuration YAML	28
5.3.2. Home Assistant Overview	28
5.3.3. Security.....	28
5.3.4. Different Devices Usage	28
6. RESULTS & DISCUSSION	29
6.1. SENSOR DATA COLLECTION	29
6.2. WI-FI AND MQTT COMMUNICATION	30
6.3. SYSTEM INTEGRATION	31
6.4. ANALYSIS OF RESULTS	31
7. CONCLUSIONS & FUTURE WORK.....	33
7.1. CONCLUSIONS.....	33
7.2. FUTURE WORK	34
ANNEXES	35
1. FLOWCHART OF THE IMPLEMENTED CODE FOR THE COFY-COOKIE.....	35
2. RASPBERRY PI & HOME ASSISTANT INSTALLATION TUTORIAL	36
3. GITHUB & CLOUD REPOSITORY FOR THE PROJECT.	43
REFERENCES	44

List of Figures

Figure 1. Initial Project Scheme.	2
Figure 2. REScoop's Architecture.	3
Figure 3. REScoop's Model.	4
Figure 4. Shelly Plus 1.	4
Figure 5. Shelly Plus ADD-ON.	4
Figure 6. Tuya Switch.	5
Figure 7. Raspberry Pi4 used in the project.	6
Figure 8. Esp32-C3 used in the project.	7
Figure 9. Esp32-S2 used in the project.	7
Figure 10. DHT22.	8
Figure 11. DHT11.	8
Figure 12. PZEM-004T V3.	9
Figure 13. JSY-MK-109.	9
Figure 14. LDR used in the project.	10
Figure 15. Eletromechanical Relay.	11
Figure 16. System Overview.	13
Figure 17. ESP32-S2 Overview.	14
Figure 18. ESP32-S2 Pinout.	14
Figure 19. ESP32-C3 Pinout.	15
Figure 20. Resistance as a function illumination.	15
Figure 21. Correlation between resistance and lumens.	16
Figure 22. LDR eletrical scheme.	16
Figure 23. DHT11 connection to MCU.	17
Figure 24. Communication between DHT11 and MCU.	18
Figure 25. PZEM-004T V3 Pinout.	18
Figure 26. Eletromechanical Relay.	20
Figure 27. Opto-Coupler's eletrical scheme.	21
Figure 28. Relay's eletrical scheme.	22
Figure 29. Electronic Circuit using the ESP32-S2.	23
Figure 30. Electronic Circuit using the ESP32-C3.	24
Figure 31. Cofy-Cookie Mounted on Breadboard.	24
Figure 32. Raspberry Pi 5 used as server.	27
Figure 33. Home Assistant Home Page.	28
Figure 34. Temperature and Humidity Measurements.	29
Figure 35. Luminosity Values.	29
Figure 36. Electricity Measurements.	30
Figure 37. MQTT Wireshark Capture.	30
Figure 38. Home Assistant's Home Page.	31
Figure 39. Home Assistant's Bedroom Page.	32
Figure 40. Flowchart of the Cofy-Cookie.	35
Figure 41. Choosing Raspberry Pi Device.	36
Figure 42. Choosing the Operating System.	36
Figure 43. Choosing the Operating System (64-bit).	37
Figure 44. Selecting edit settings.	37

Figure 45. Wireless LAN Configuration. 37

Figure 46. Ping Command..... 38

Figure 47. ssh Command..... 38

Figure 48. ssh Configuration. 38

Figure 49. ssh Configuration. 39

Figure 50. Update Command. 39

Figure 51. Upgrade Command. 40

Figure 52. CasaOS installation..... 40

Figure 53. CasaOS status..... 40

Figure 54. CasaOS Creating Account..... 41

Figure 55. Overview Page..... 41

Figure 56. Home Assistant Page..... 42

List of Tables

Table 1. Single Board Computers considered in the Initial Study.	6
Table 2. Microcontroller Comparison.....	7
Table 3. Humidity and Temperature Sensor Comparison.	8
Table 4. Measurements Results.	19

1. Introduction

In seeking sustainable and efficient solutions in the energy sector, European energy cooperatives promote efficiency by implementing innovative systems. The first chapter discusses the project's motivation, objectives, system requirements, and report outline.

1.1. Motivation

The project aims to create an open-source residential system incorporating sensors to track energy consumption, offering users feedback to make informed decisions, thereby minimizing waste and reducing electricity costs. The primary motivation is to provide a system that delivers insights into household electricity consumption, enabling automated decisions to maximize renewable energy integration.

Optimizing energy consumption at home is crucial, particularly with systems capable of controlling appliances to operate during periods of higher renewable energy availability, such as solar. Given the use of personal data and the vulnerability of closed systems to hacking, adopting such systems raises serious concerns.

This project builds on a previous study by REScoop, which proposed a controlled home environment that can monitor and manage the energy costs of each appliance. However, the study did not provide a final product or specify optimal components. Our goal is to develop and publish an open-source system that can be easily adopted by the IoT community and expanded for more complex use cases.

1.2. Objectives

The project aims to implement an embedded system, designed to analyze each section's energy consumption within a household. This initiative seeks to effectively reduce electricity usage.

The sensor data collected by the “Cofy-Cookie” in each section of the house is transmitted to an application called "Home Assistant", deployed on a Raspberry Pi, which will be referred to as “Cofy-Box” throughout the report. This application facilitates the analysis of each monitored section, providing insights into the consumption of individual components. The "Cofy-Cookie" is able to receive commands from the “Cofy-Box” to control home appliances such as washing machines and heaters & AC.

1.3. System Requirements

The system requirements for the project are as follows:

- Analyse the current and voltage flowing in an electrical line using the *PZEM-004T V3* sensor.
- Use an ESP32 MCU to transmit data obtained from the sensor via Wi-Fi to the "Home Assistant" application.
- In the application, installed on a Raspberry Pi, it is possible to analyze the data sent from different areas.
- From the application, send a message back to the desired microcontroller to control the respective device on or off.

- Connect the microcontroller to a Solid State Relay (SSR), or electromechanical relay, that implements the instruction given to the ESP32 (MCU) by "Home Assistant."
- Set up a Wi-Fi router, which will serve as the base for the system to work standalone but also within any given Wi-Fi network available.

In **Figure 1**, it is possible to see the initial project scheme.

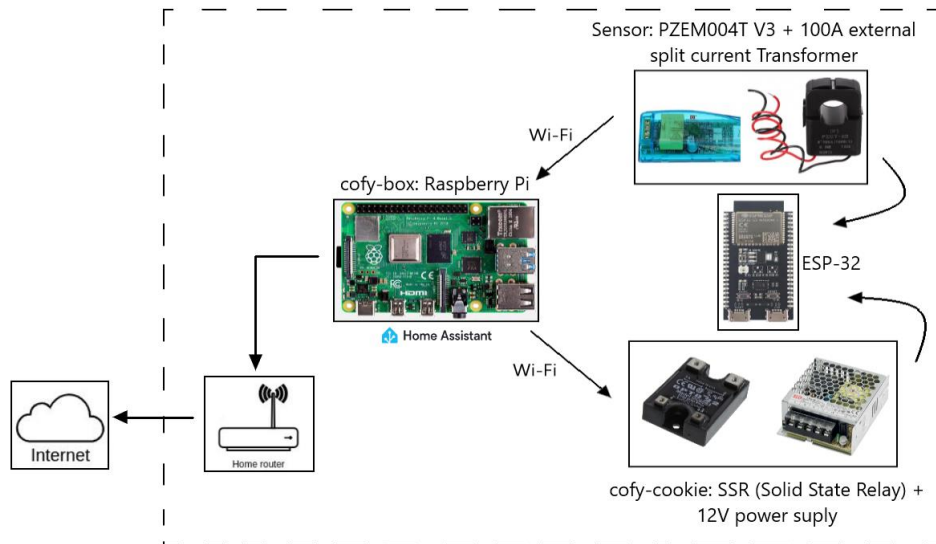


Figure 1. *Initial Project Scheme.*

Besides using the Solid State Relay shown in **Figure 1**, it is also possible to use a relay. Subsequently, when implementing the “Home Assistant” application, the usage of a *Raspberry Pi* or an *Orange Pi* will be a task undertaken during the selection process to identify the most optimal solution.

1.4. Report outline

This report is organized as follows. Chapter 2 presents the background, reviewing literature and existing technologies pertinent to the development of the home automation system. Chapter 3 describes the proposed home automation system, detailing the planned architecture, key components, and their functions. Chapter 4 covers the implementation of the Cofy-cookie, while Chapter 5 details the implementation of the Cofy-box. Chapter 6 discusses the results, providing data and performance analysis of the implemented system, as well as discussing these results. Chapter 7 offers conclusions and future work, summarizing the findings and suggesting possible improvements and extensions for future research. Finally, Chapter 8 lists the references, including all bibliographic sources and resources used throughout the report.

2. Background on Home Automation

This section is dedicated to explaining the foundation of this project, as it took some inspiration from different articles and reports available online and from some companies that already provide closed-source products that have almost the same purpose as this work. There are some comparisons for different materials that could be used in this project and justifications of why each component was chosen.

2.1. Related Works

After research and analysis of scientific articles and reports available online, several sources were identified that cover the technologies utilized in this project. While many articles and reports were reviewed, the one referred to below provides a comprehensive explanation of the theoretical basis for the current project. There is also mention of some companies that are dedicated to domotic technologies that already sell products that come close to the project at hand, with closed-source software and hardware.

2.1.1. REScoopVPP

The project report “Smart Building Ecosystem for Energy Communities” by REScoop [1], is focused on the description of an open-source energy monitoring and optimization system. It uses a range of existing open-source technologies to integrate and control legacy and new appliances into a smart building ecosystem. Helps in understanding the Hardware behind the final product as well as real-life options to use the device.

This work delivered an idea of how to get consumption data from different elements of a house and how to deliver them to a user, even though they don’t specify which components can make part of the Cofy-Cookie. For the Cofy-Box, there are some examples of software that make part of their project, one of them the “Home Assistant” as can be seen in **Figure 2**.

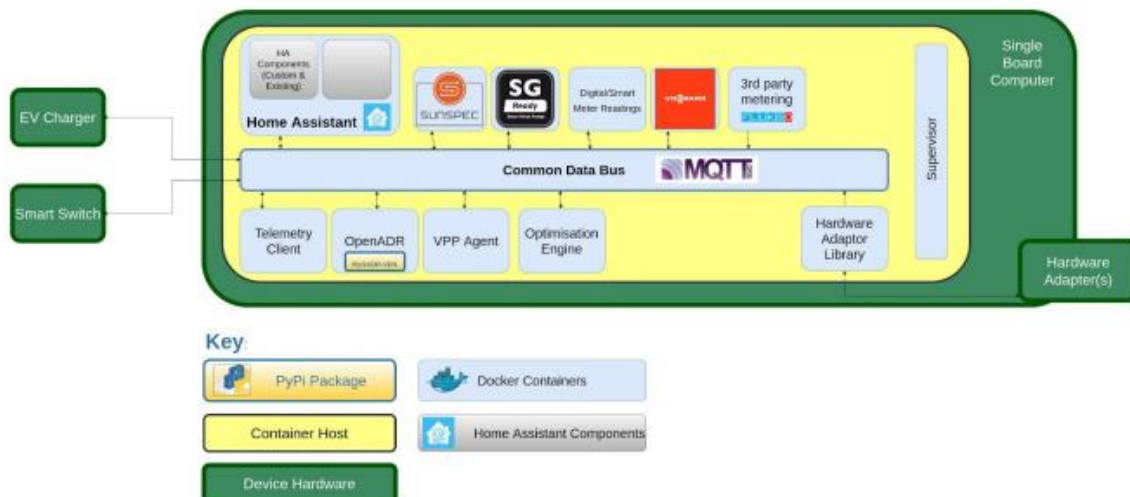


Figure 2. *REScoop's Architecture.*

REScoop made a very simple scheme that allows an easy understanding of how their project is going to work in **Figure 3**, however, they did not specify all the components for its implementation. The challenge for the current project is to find the best hardware components and applications that make this idea the safest and cheapest way to control energy consumption and the working time of household appliances.

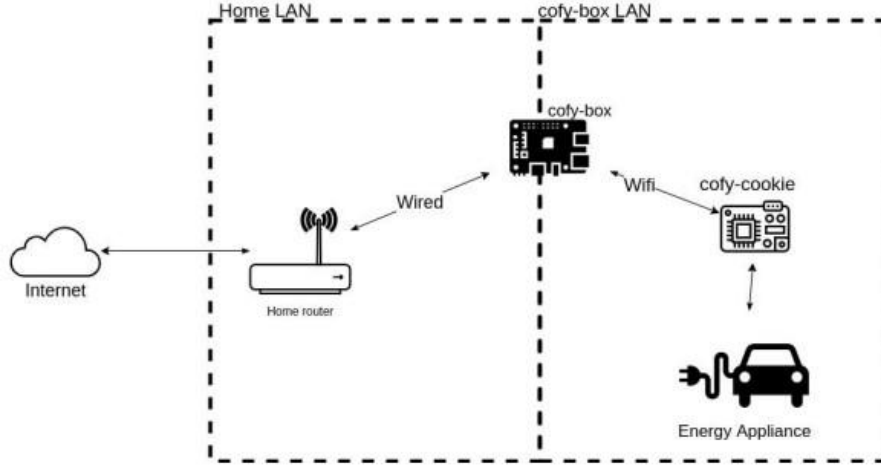


Figure 3. *REScoop's Model.*

2.1.2. Shelly

The advisor of this project provided equipment from Shelly, a company that dedicates their work to many areas, one of them being domotic devices. The devices provided were Shelly Plus 1 and Shelly Plus ADD-ON, shown in **Figure 4** and **Figure 5** respectively. The first device is an interrupter with a relay controlled by Wi-Fi and it can be installed in almost any appliance at home. The other device is used to make measurements of temperature and humidity, light intensity, and movement, for example.



Figure 4. *Shelly Plus 1.*



Figure 5. *Shelly Plus ADD-ON.*

These devices are closed-source, which can present adversity. This project as an open-source aims to become a solution to this issue.

2.1.3. Tuya

Tuya is another company dedicated to providing IoT solutions for many areas, one being house domotic. Different from Shelly, Tuya doesn't provide measurement devices to install in home appliances, but they provide software and equipment to have in a smart and efficient home. As a closed-source product, it doesn't provide the reliability and flexibility that is provided by the proposed project. **Figure 6** shows one of the most used smart switches by this brand.

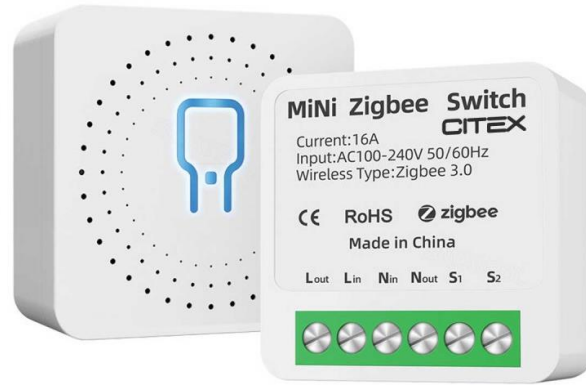


Figure 6. *Tuya Switch.*

2.1.4. Other Solutions

As for other solutions, there is the *Hue Bridge* from *Philips*. This device is used to control smart lamps, it uses a different protocol for communication with the lamps called *Zigbee*. It also uses ethernet to be connected to the lamps. It is used through their application, where you can also add automation for your smart lamps.

NOUS E1 is another solution that was found, which is a device that can control up to 128 smart devices that work with Zigbee connectivity or Bluetooth

2.1.5. Comparison of Available Solutions

Several solutions were reviewed for their applicability to the project's goals. REScoopVPP provides insights into open-source energy monitoring but lacks detailed hardware specifications. Shelly and Tuya offer closed-source solutions with limited customization, while Philips Hue and NOUS E1 focus on proprietary ecosystems like Zigbee for specific smart home functionalities. Each solution's balance of functionality, openness, and integration capabilities makes it interesting to explore a solution to take the best from each.

2.2. Relevant Technologies

In this section, there is a brief explanation of each component and technologies involved in the project and a comparison between components to show which ones are the best possible for this project.

2.2.1. Single Board Computer

The communication between a household appliance and the user is facilitated through an application installed on a Raspberry Pi. Specifically, the application utilized is Home Assistant, with the scientific report [13] detailing the ease of its implementation on *Raspberry Pi* and the advantages it offers over alternative applications. This report shares many similarities with the project being discussed, providing ample evidence that the Home Assistant is a good method of exploring IoT within a domestic setting. **Table 1** shows the comparison.

Table 1. *Single Board Computers considered in the Initial Study.*

Single Board Computer Comparison		
Model	Raspberry Pi 4	Orange Pi zero
RAM Memory	4GB	0.5GB
CPU Speed	4 x 1.8 GHz	4 x 1.2 GHz
Wi-Fi Version	Wi-Fi 4 (802.11n); Wi-Fi 5 (802.11ac)	Wi-Fi 4 (802.11n)
Bluetooth	yes	no
HDMI Output	yes	no
USB Type-C	yes	no
Operating Voltage	5V	5V

Raspberry Pi 4 was the chosen computer to use in this project, there was the option to use an *orange Pi Zero*, but after studying the hardware, it was decided that it wasn't recommended due to the lack of computer power. *Raspberry PI 4* also has better and newer features, for example, support for newer Wi-Fi protocols, Bluetooth, HDMI output, and USB Type-C. In **Figure 7**, you can see the comparison made between the 2 computers and better understand the reasoning for this choice. This comparison was based on [7].

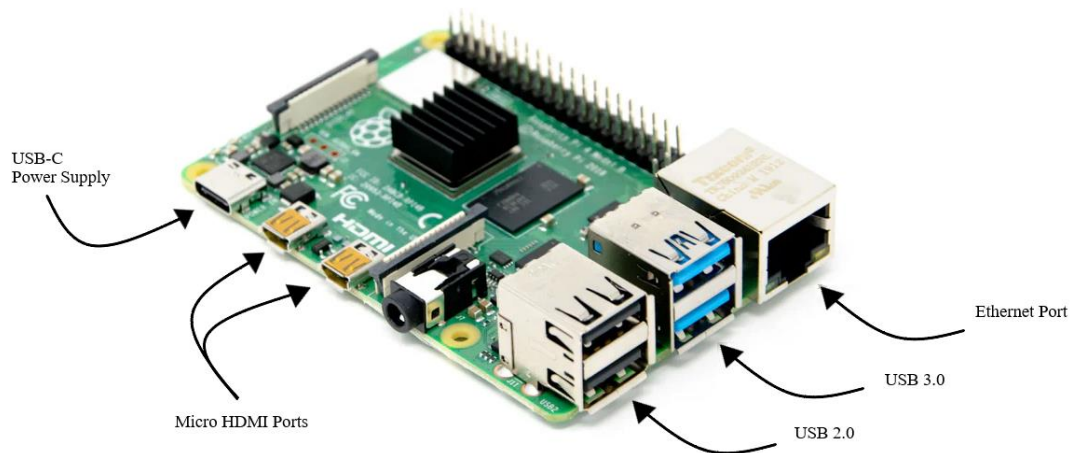


Figure 7. *Raspberry Pi4 used in the project.*

2.2.2. Microcontroller.

The project needs a microcontroller to implement the Cofy-Cookie. It will gather information from the sensors and communicate them with the Home Assistant serving on the SBC. The microcontroller needs to be able to talk to the server, so it needs Wi-Fi.

The article specified in [4] explains in detail the ESP-32 and its use, which will be an important device in this project. This microcontroller already has a Wi-Fi module and is connected to BLE (Bluetooth Low Energy) via a chip, so it is very powerful and can be a good choice for creating an IoT application system. In [5], there is also the comparison between different versions of the ESP32, so it was also used for this project.

This last one along with [8], helped extract a comparison table between the *ESP32-S2*, *ESP32-S3*, and *ESP32-C3*, which were the models frequently used for this kind of project.

Table 2. *Microcontroller Comparison.*

Microcontroller Comparison			
Model	ESP32-S2	ESP32-S3	ESP-C3
Frequency (MHz)	240	240	160
SRAM (KB)	320	512	400
ROM (KB)	128	384	384
Flash (MB)	Up to 4	Up to 8	Up to 4
SPI (MB)	4	4	3
USB	Yes	Yes	Yes
Wi-Fi	802.11 b/g/n, 2.4 GHz	802.11 b/g/n, 2.4 GHz	802.11 b/g/n, 2.4 GHz
Bluetooth	No	Yes	Yes
GPIO	43	45	22
CPU	single-core	dual-core	single-core
Price (€)	7.52	14.1	8.46

According to Erro! A origem da referência não foi encontrada., the most efficient choice for the project will be the *ESP32-C3*. The C3 contains sufficient ROM memory and SRAM capacity, offering the necessary resources for interfacing with various peripherals. The C3 also has Bluetooth and the S2 does not have this feature, even though this project is very unlikely to use a Bluetooth connection since the use of Wi-Fi is optimal for further distances. Moreover, the *ESP32-C3*, as seen in **Figure 8**, provides an efficient performance with its single-core CPU, when compared to a dual-core CPU. Therefore, considering these factors, the *ESP32-C3* emerges as the most suitable option for our project.

The project also incorporated a solution for using the ESP32-S2 microcontroller, as seen in **Figure 9**, which was solicited to be also considered by this project's advisor.

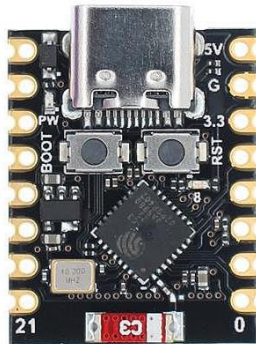


Figure 8. *Esp32-C3 used in the project.*

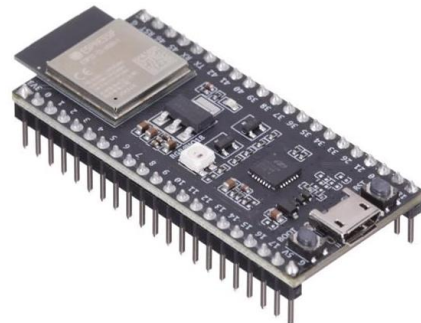


Figure 9. *Esp32-S2 used in the project.*

2.2.3. Temperature and Humidity Sensor

A component integrated into the "Cofy-Cookie" is the temperature and humidity sensor. This sensor interfaces with an MCU ESP32, with specific parameters outlined in this paper [2]. Although this work primarily evaluates the connection between the sensor and an Arduino, understanding the necessary response times for accurate data is important to understand the use of this component.

The research made, and based on [9], showed two sensors with the potential to be used. **Table 3** shows the different specifications of each.

Table 3. *Humidity and Temperature Sensor Comparison.*

Humidity and Temperature Sensors		
Model	DHT11	DHT22
Measures	Temperature and Humidity	Temperature and Humidity
Interface	One-wire	One-wire
Supply Voltage	3 to 5.5V	3 to 5.5V
Current Supply	0.5 – 2.5 mA	1 - 1.5 mA
Temperature Range	0° to 50°C	-40 to 80°C
Resolution	Humidity: 1% ; Temperature: 1°C	Humidity: 0.1%; Temperature: 0.1°C
Price (€)	6,99	10,99

The following figures, **Figure 11** and **Figure 10** represent the temperature and humidity sensors compared in the previous table.

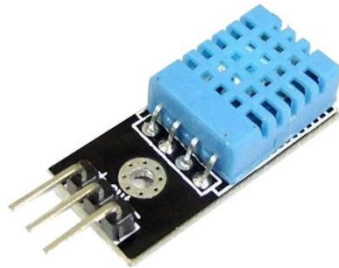


Figure 11. *DHT11.*



Figure 10. *DHT22.*

For communication, both DHTs use one wire which makes it easier to use, than SPI. Even though, in DHT11, the temperature range does not include negative temperatures and the resolution is less accurate due to having a bigger interval, this sensor is sufficient for the case study as the ambient in which the work will be performed does not require more specification.

The MCU utilized in the Cofy-Cookie is an ESP32, operating at a voltage of 3.3V. Similarly, the temperature and humidity sensor DHT11 employed in this project operates at the same voltage level as the MCU.

2.2.4. Power Consumption Sensor

To keep track of the power consumption on an electric wire, or line, it is necessary to use a sensor, which was decided would be the *PZEM-004T V3*.

The *PZEM-004T V3*, represented in **Figure 12**, is a multifunctional sensor module that functions to measure power, voltage, current, and energy contained in an electric current. The paper specified in [3], explains a lot about the use of this device and will be important to understand how to implement it in the project at hand.

PZEM-004T V3 has a serial UART (Universal Asynchronous Receiver/Transmitter) that works with 5V, to allow the communication to work at 3.3V with a board such as *Raspberry Pi* or *ESP32* a resistance of 1k Ω must be used, so with this modification, UART can work at 5V or 3.3V [6].

Another option could be the power consumption sensor represented in **Figure 13**. The downsides of this sensor are that the wire has to pass through the printed circuit board (PCB), making its usability more complex, and the fact that it does not have a box that isolates the 220V, which implies a more careful experiment.

Therefore, the *PZEM-004T V3* will be the chosen sensor for the current project. It was provided by the advisor of this project and after exhaustive research it was concluded that this module is the most utilized in this type of work, for its accuracy and efficiency.

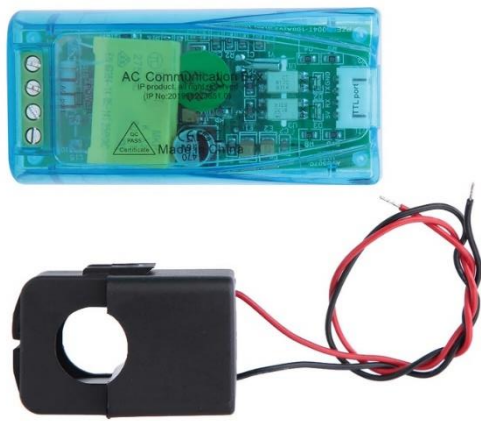


Figure 12. *PZEM-004T V3*.



Figure 13. *JSY-MK-109*.

2.2.5. Light Sensor

The decision to include a light sensor in the energy optimization system was driven by the need to monitor ambient light levels. Effective management of lighting can significantly reduce energy consumption, especially during daytime when natural light is available. For this purpose, a Light Dependent Resistor (LDR) was chosen due to its simplicity, reliability, and sensitivity to changes in light intensity. The LDR is a widely used sensor for detecting light levels because it offers several advantages:

1. **Cost-Effectiveness:** LDRs are inexpensive, making them a cost-effective choice for the implementation.
2. **Ease of Integration:** With three simple connections—VCC, Ground, and Analog Out—LDRs are easy to integrate into existing systems.
3. **High Sensitivity:** The change in resistance in response to light intensity makes the LDR highly sensitive, providing accurate measurements over a wide range of lighting conditions. This change is better explained in chapter 3.3.3.
4. **Low Power Consumption:** LDRs consume minimal power, which is ideal for applications focused on energy efficiency.

These factors collectively make the LDR a good choice for monitoring ambient light and optimizing energy usage in the household energy management system. **Figure 14** shows the ldr used in this project, a module with the resistor included.

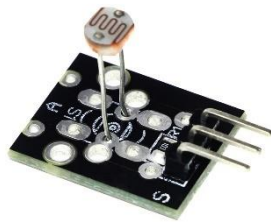


Figure 14. *LDR used in the project.*

2.2.6. Electronic Switching Device

To make the user's instructions happen, there is a need for a switching device to be able to control the appliances through the application. By researching on the Internet, there were a few options to implement. The most common were the Solid State Relay and the Electromechanical Relay.

Solid State Relays use semiconductor devices for switching, which means that they consume very little power. They can operate on as little as a few milliwatts. Electromechanical Relays, on the other hand, use electromechanical contacts to switch the current. This process requires more power [10]. However, the project will include the Relay, since it was the available solution for the project, the relay is represented in **Figure 15**.



Figure 15. *Electromechanical Relay.*

2.2.7. Domotic Application

There was the need for an application to give the user the ability to control household devices and with an interface to see each sensor's measurements. Among these, two applications stand out: Home Assistant and OpenHAB.

While there are minimal differences between them, Home Assistant holds a notable advantage over OpenHAB due to its extensive array of open-source programming options. Additionally, Home Assistant is recognized for its robust automation engine, user-friendly interface, and powerful integrations, yet not by a significant margin. Therefore, Home Assistant will be employed for this project to maintain alignment with the original concept.

2.2.8. IOT Communication Protocol

After evaluation, the MQTT (Message Queuing Telemetry Transport) protocol was selected for facilitating communication between the device and the server. MQTT is widely adopted in the IoT industry for its lightweight and efficient operation.

MQTT operates on a publish-subscribe model, where devices (publishers) send messages to a broker, and other devices (subscribers) receive messages from the broker based on their subscriptions.

Small Packet Size: MQTT messages are compact, minimizing the data transmitted over networks. This feature is particularly advantageous for IoT devices operating on constrained networks with limited bandwidth.

Low Overhead: The protocol utilizes an efficient binary messaging format, resulting in minimal network overhead. This efficiency reduces data transmission times and conserves power, which is critical for battery-operated devices.

Retained Messages: MQTT brokers can store the last message sent to a topic, allowing new subscribers to receive the most recent status immediately upon connecting. This feature is useful for applications requiring real-time data updates.

Session Persistence: MQTT supports persistent sessions, enabling clients to reconnect and resume communication with the broker seamlessly. This ensures continuous operation even in the event of network disruptions.

Due to its lightweight design and efficient use of network resources, MQTT enhances energy efficiency in IoT applications. Devices can frequently enter low-power states for extended periods.

Compared to protocols such as HTTP, which are request-response based and typically involve higher overhead, MQTT offers advantages in responsiveness and resource utilization. These characteristics make MQTT a preferred choice for IoT deployments requiring efficient, scalable, and reliable communication.

In conclusion, MQTT's design principles and feature set make it a robust choice for IoT communication, balancing efficiency, reliability, and scalability across diverse deployment scenarios.

3. Proposed Home Automation System

This chapter provides the proposed system architecture. It begins with an introduction to the system overview, which includes a description of the main components and the interaction between them. The next sections focus on each component that was chosen and explain why were they chosen in the previous chapter.

3.1. System Overview

After researching all the alternatives for the elements to use in this project, **Figure 16** represents, the block diagram to be implemented for this work. It also represents the working state of this project.

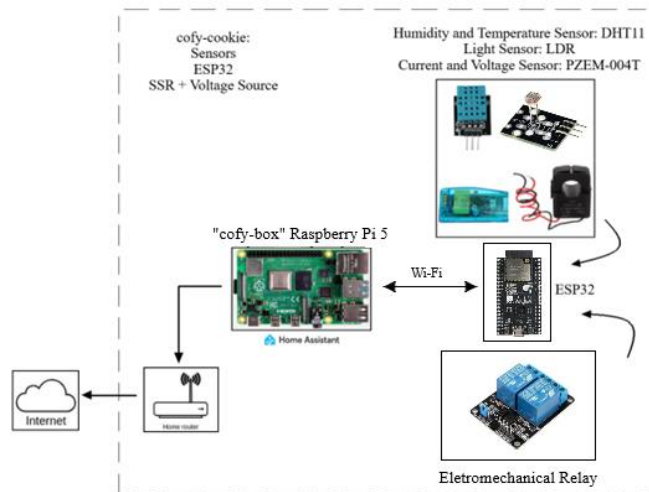


Figure 16. *System Overview.*

The previous figure gives an image of the system itself. The top right of the figure shows the Cofy-Cookie that contains the sensors. These communicate with the ESP32, which is located right under them in the figure. The ESP32 then sends the information through Wi-Fi to the Raspberry Pi 4 that has the home assistant application running, allowing the user to see the measurements taken by the sensors, and to send back information for the Cofy-Cookie with the SSR (bottom right of the figure). These allow the user to take action based on the information. The Raspberry Pi 4 is also connected to a home router, configured by the authors, to connect it to the internet as seen on the left side of the figure.

3.2. Cofy-Box

The Cofy-Box serves as the intermediary between the user and the Cofy-Cookie, facilitating communication. It consists of a *Raspberry Pi 5*, a single-board computer, with a Home Assistant image to oversee all Cofy-Cookie measurements, installed in an SD Card that must have at least 16GB of space, so it can have the image used on this project, this will be explained later on this report. The decision to utilize a single-board computer for the Cofy-Box is driven by the goal of reducing expenses, conserving energy, and optimizing memory usage in handling incoming data from the Cofy-Cookie and user interactions.

3.3. Cofy-Cookie

To have a working project that can be compared with the schematics previously presented, and as this work is an open-source project, this section explained how each component connects to others to have a functional Cofy-Cookie. Arduino IDE will be used to program both ESP32.

3.3.1. ESP32-S2

The microcontroller used for the system will be the ESP32-S2, as said before. **Figure 17** shows the overview of each block of the MCU.

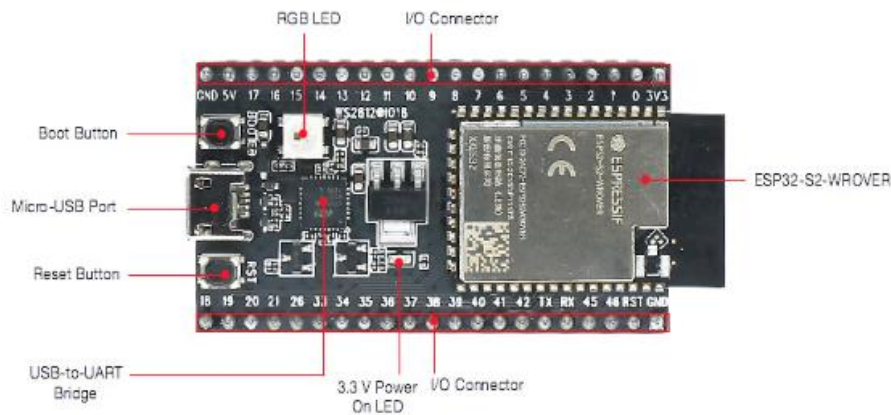


Figure 17. *ESP32-S2 Overview.*

Figure 18 shows the pinout of the chip, where it is possible to see where each pin is located, and the availability of so many GPIO ports, as well as a 3.3V pin and a 5V pin, 1 reset pin, and 2 grounds. It is not possible to see in the figure, but there is an Rx in pin 44 and a Tx in pin 43.

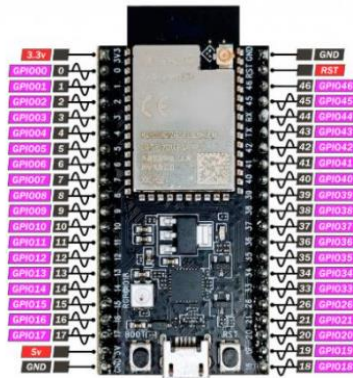


Figure 18. *ESP32-S2 Pinout.*

3.3.2. ESP32-C3

To showcase the project's adaptability with various components, an alternative microcontroller, the ESP32-C3, was also tested. This decision was influenced by the project's minimal GPIO pin needs, which arise from the use of a small number of sensors.

As it is possible to see in **Figure 19**, there are much fewer GPIO pins in this microcontroller than in the ESP32-S2 but there are enough for the project that is being designed.

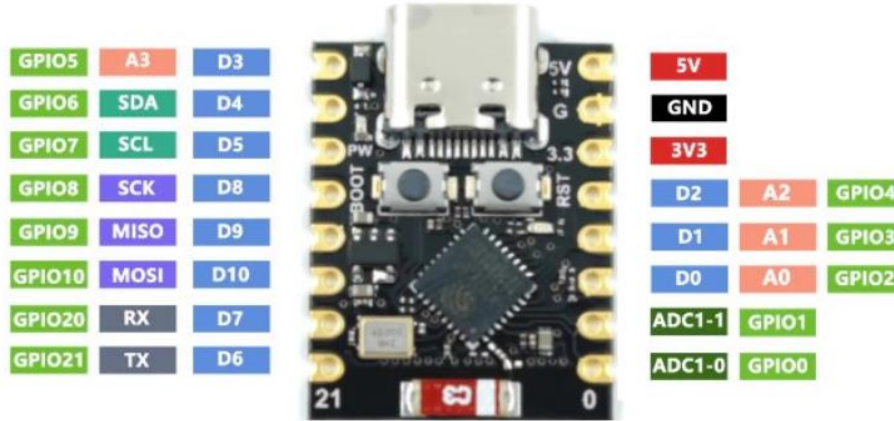


Figure 19. *ESP32-C3 Pinout.*

3.3.3. LDR

To optimize energy savings during daytime, an ambient light sensor was integrated into the system. This sensor utilizes a LDR, which exhibits variable resistance depending on the intensity of ambient light. The LDR operates on the principle that as light intensity increases, its resistance decreases accordingly.

As it is shown in the **Figure 20** retrieved from the datasheet of this component [14] it is possible to see that there is a linear relationship between resistance and lumens. From this graph it is possible to make different study cases taking measurements of the resistance of the LDR and correlate those with lumens values.

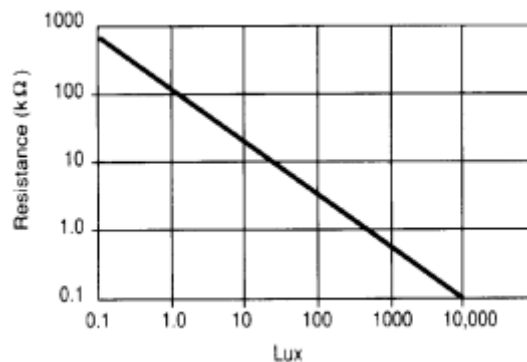


Figure 20. Resistance as a function illumination.

The analog value obtained from the Arduino using the LDR ranged between 0 and 4095, corresponding to a 3.3V voltage source from the ESP32-S2. This means that a 3.3V output directly translates to an analog reading of 4095. Using this relationship, the voltage for each lighting condition can be calculated using a simple proportionality (rule of three).

Considering this graph, seven different resistance levels were tested under varying lighting conditions to determine the corresponding lumens values for our specific component. This can be seen in **Figure 21**.

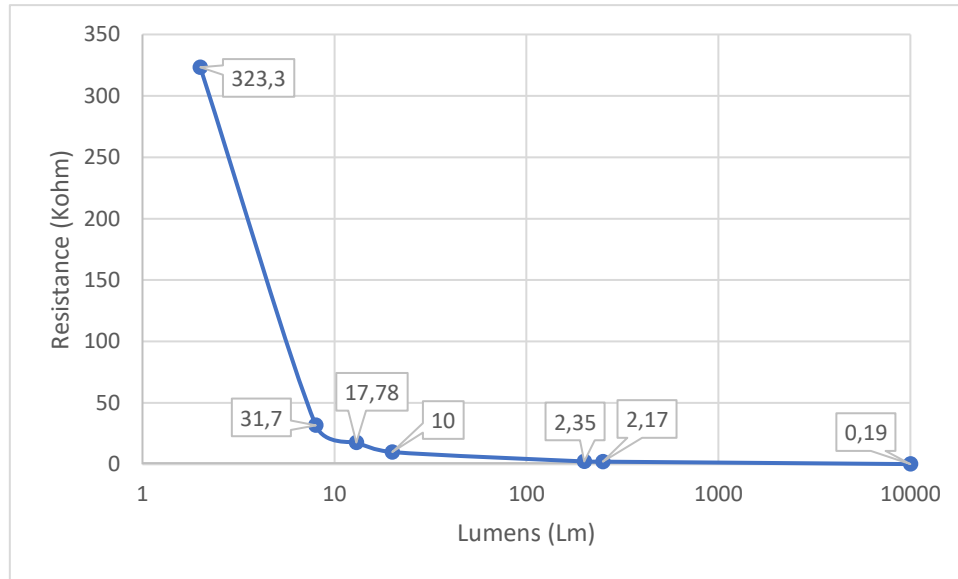


Figure 21. Correlation between resistance and lumens.

The LDR circuit consisted of a series connection between a pull-up resistor and the LDR itself, as shown in **Figure 22**. The pull-up resistor that should be used must be around the resistance value of the LDR when exposed to low-intensity light. As shown in **Figure 21**, at the middle illumination level, we have a resistance of 10k ohms, which is why this value is used for this specific LDR. For each measurement, the voltage value was first determined, as explained earlier. Using the voltage divider equation, the LDR's resistance was then calculated. These resistance values were subsequently correlated with lumens values using the function presented in **Figure 20**.

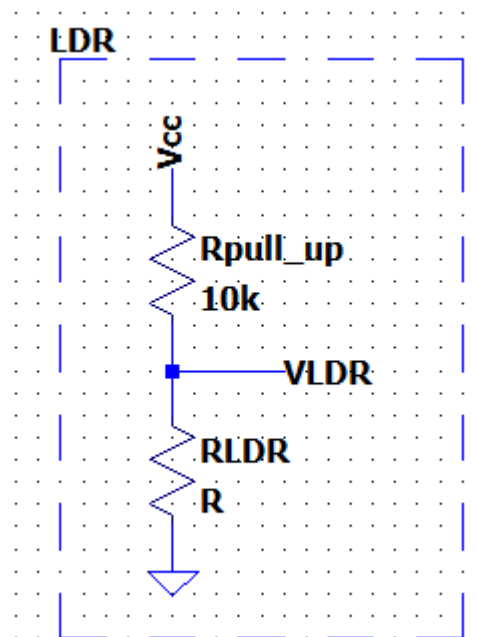


Figure 22. LDR eletrical scheme.

3.3.4. DHT11

The **DHT11** is a sensor that can measure both temperature and humidity. It has a built-in microcontroller that processes the data and sends it to a connected device, such as an Arduino, via a single data wire. It's small, low-cost, and easy to use, making it a good choice for projects where there is the need to monitor environmental conditions.

The DHT11 includes two key sensing elements:

1. **Humidity Sensing Element:** This component is a resistive humidity sensor that measures the moisture content in the air. As the humidity changes, the resistance between the sensor's electrodes also changes, allowing the sensor to detect and measure relative humidity.
2. **Temperature Sensing Element:** The sensor uses an NTC (Negative Temperature Coefficient) thermistor to measure temperature. This thermistor decreases its resistance as the temperature rises, enabling the sensor to accurately gauge the temperature of the surrounding environment.

These elements work together with the built-in microcontroller to provide precise readings of temperature and humidity, making the DHT11 a reliable and versatile sensor for various applications.

The DHT11 operates on a voltage between 3 to 5.5V. It has 3 pins, the VCC to connect to the power supply, the GND to connect it to the ground of the circuit and the DATA pin which is a single wire used for communication. This can be seen in **Figure 23**.

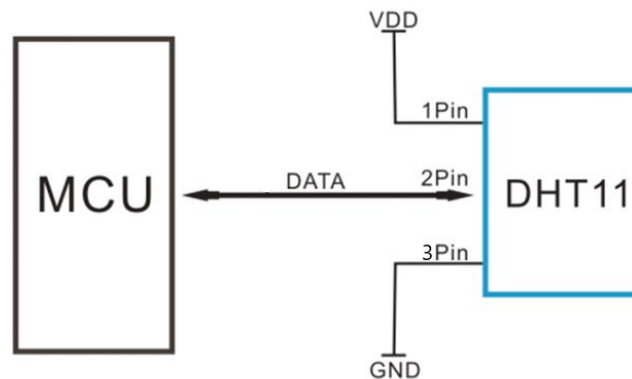


Figure 23. DHT11 connection to MCU.

The DHT11 communicates with the microcontroller, in this case the ESP32, using a **single-wire serial interface**. This means data is transferred between the DHT11 and the microcontroller through just one wire.

When the microcontroller starts communication, it sends a signal to the DHT11. The sensor then sends back 40 bits of data. This data includes:

- 16 bits for humidity (8 bits for the integer part, 8 bits for the decimal part).
- 16 bits for temperature (8 bits for the integer part, 8 bits for the decimal part).
- 8 bits for a checksum to ensure the data is correct.

The communication process has several steps included. It starts with the microcontroller sending a start signal which will set Data Single-bus voltage level from high to low and this process must take at least 18ms to ensure DHT's detection of MCU's signal.

Once DHT detects the start signal, it will send out a low-voltage-level response signal, which lasts 80us. Then the programme of DHT sets Data Single-bus voltage level from low to high and keeps it for 80us for DHT's preparation for sending data. When DATA Single-Bus is at the low voltage level, this means that DHT is sending the response signal. Once DHT sent out the response signal, it pulls up voltage and keeps it for 80us and prepares for data transmission. When DHT is sending data to MCU, every bit of data begins with the 50us low-voltage-level and the length of the following high-voltage-level signal determines whether data bit is "0" or "1". If the high voltage-length takes 70us it means the bit is 1, if it takes 26-28us it means it's a 0.

All of these steps can be seen in **Figure 24**.

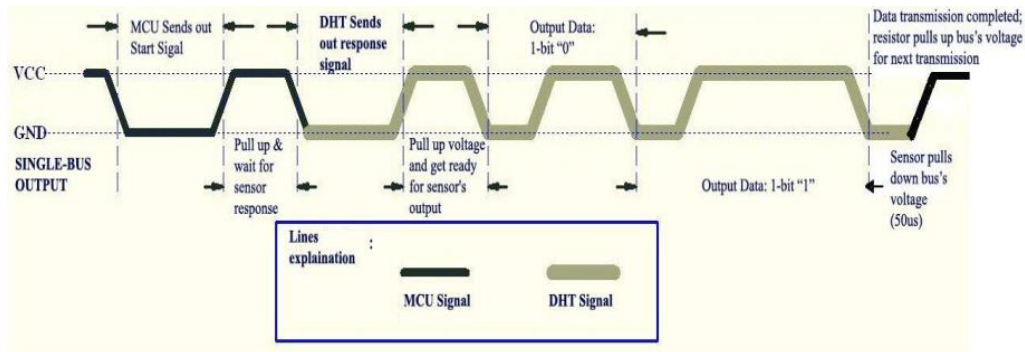


Figure 24. Communication between DHT11 and MCU.

Finally, to transmit the data obtained from the DHT11 to the Home Assistant application, a pre-existing and functional library, in Arduino code [14] was tested. However, to make the project more unique, it was decided to create a new library, based on the scheme shown in **Figure 24**.

3.3.5. PZEM-004T V3

The *PZEM-004T V3* will be used to measure current. In **Figure 25** from bottom to top, the left pins are: Pin1 is the ground, which connects the sensor to the ground of the circuit. Pin2 is the Tx pin that will connect to the Rx pin of the microcontroller, for communication. Pin3 is the Rx pin that will connect to the Tx pin of the microcontroller, and Pin4 is the power supply of 3,3 - 5V. The pins on the right side of the component from top to bottom are: Pin1 and Pin2 connected to the load terminals, being Pin1 connected to the load part that goes through the current transformer. Pin 3 and Pin 4 are the cables that make the connection between the *PZEM-004T V3* and the current transformer.



Figure 25. *PZEM-004T V3* Pinout.

How the PZEM-004T V3 Works:

1. **Current Measurement:** The current transformer (CT) attached to the load measures the current flowing through the monitored wire.
2. **Voltage Measurement:** The PZEM-004T also measures the AC voltage across the load. It uses both the current and voltage readings to calculate power consumption ($P = IV$), where I is current and V is voltage.
3. **Power and Energy Calculation:** With both voltage and current values, the module calculates the power (in watts) being used by the load. The module also tracks energy consumption over time, reporting the total energy consumed in watt-hours (Wh).
4. **Frequency Measurement:** In addition to current and voltage, the PZEM-004T measures the frequency of the AC signal, typically 50Hz or 60Hz, depending on the region.
5. **Microcontroller Communication:** Data collected by the PZEM-004T is sent to a microcontroller via the UART serial interface (Tx and Rx pins). The microcontroller can process this data, display it, or use it for decision-making in energy management systems.

The PZEM-004T V3 module uses **opto-couplers** (also known as **optocouplers** or **optoisolators**) as part of its design to ensure electrical isolation between different parts of the circuit. This isolation is essential for safe and reliable operation when measuring AC power, especially when interfacing with sensitive microcontroller systems.

It is a versatile and reliable module for monitoring electrical parameters in AC circuits. The left-side pins manage communication and power supply, allowing it to interface with a microcontroller, while the right-side pins handle current and voltage measurements through the load and current transformer. This setup enables real-time monitoring of critical parameters such as current, voltage, power, energy, and frequency, making it an essential tool for energy management systems, smart homes, and industrial applications.

In the PZEM's manual, there is a table that shows the results of the measurements made by the device. **Table 4** shows these results.

Table 4. *Measurements Results.*

Register address	Description	Resolution
0x0000	Voltage value	1LSB correspond to 0.1V
0x0001	Current value low 16 bits	1LSB correspond to 0.001A
0x0002	Current value high 16 bits	
0x0003	Power value low 16 bits	1LSB correspond to 0.1W
0x0004	Power value high 16 bits	
0x0005	Energy value low 16 bits	1LSB correspond to 1Wh
0x0006	Energy value high 16 bits	
0x0007	Frequency value	1LSB correspond to 0.1Hz
0x0008	Power factor value	1LSB correspond to 0.01
0x0009	Alarm status	0xFFFF is alarm, 0x0000 is not alarm

3.3.6. Eletromechanical Relay

The 2-Channel Electromechanical Relay used in this project, show in **Figure 26** on the down side of the relay, Pin 1 is used for ground connection, while Pin 2 connects to a digital port of the microcontroller for control. The relay is powered through Pin 4 (VCC), which operates between 3.3V and 5V. On the top side, the terminals function as a traditional switch for controlling the load connected to the relay.



Figure 26. *Eletromechanical Relay.*

The 2-Channel Relay Module is a device that allows to control high-voltage appliances using low-voltage signals from a microcontroller, such as an Arduino or Raspberry Pi. It is often used in projects where there is the need to switch on or off larger electrical devices like lamps, fans, or other household appliances.

1. Each of the two relays on the module is an electromechanical switch that can control the flow of electricity to a device. It has three main contacts:
 - **COM (Common):** The point of connection for the incoming electrical circuit.
 - **NO (Normally Open):** The relay switch is open when the relay is inactive, meaning the circuit is off. When the relay is activated, this switch closes and allows current to flow, turning the connected device on.
 - **NC (Normally Closed):** The relay switch is closed when the relay is inactive, meaning the circuit is on. When the relay is activated, this switch opens and cuts off the current, turning the connected device off.

2. **Optocoupler:** The module includes an optocoupler, which isolates the low-voltage control side (connected to the microcontroller) from the high-voltage side (connected to the appliance). This ensures that any potential high-voltage spikes do not damage the microcontroller. **Figure 27** shows the electric scheme.

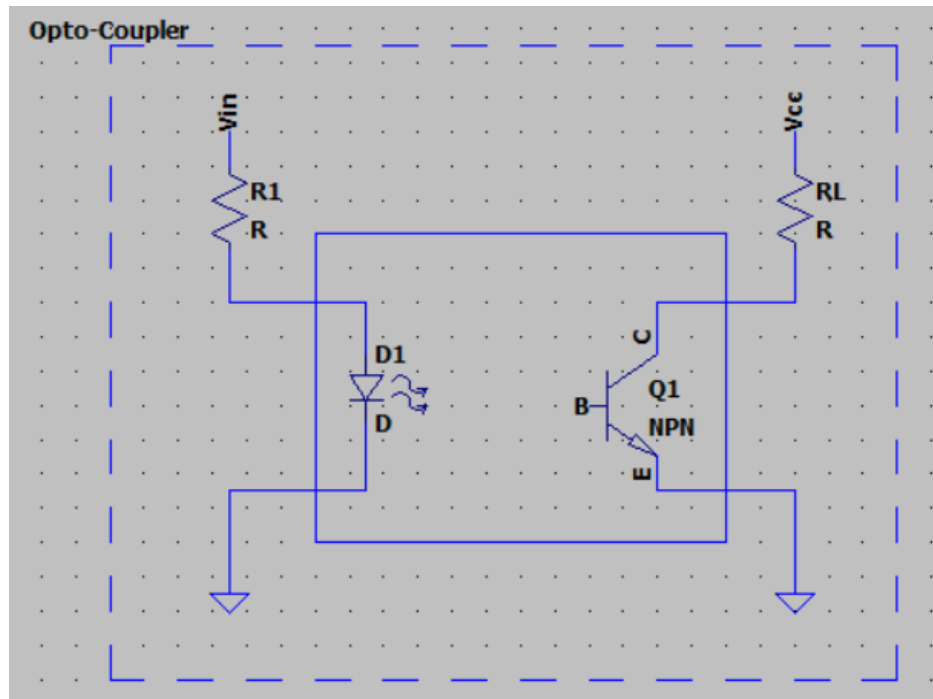


Figure 27. Opto-Coupler's eletrical scheme.

3. **Triggering Mechanism:** The relay module is designed to trigger on a LOW signal from the microcontroller. When the microcontroller sends a LOW signal to the relay's input pin, it activates the relay, switching the connected device on or off depending on the wiring.

Operating Principle

When the relay is powered and the input signal is set to LOW by the microcontroller:

- The **electromagnet** inside the relay is energized.
- This energization pulls the armature (a movable contact) toward the electromagnet.
- Depending on how the device is connected (to NO or NC), this movement either closes or opens the circuit, turning the connected device on or off.

When the signal is set to HIGH or when the relay is not powered:

- The electromagnet is de-energized.
- The armature is pushed back to its original position by a spring, restoring the circuit to its default state (either connected or disconnected).

Figure 28 represents the electrical scheme of the relay that was used in the project.

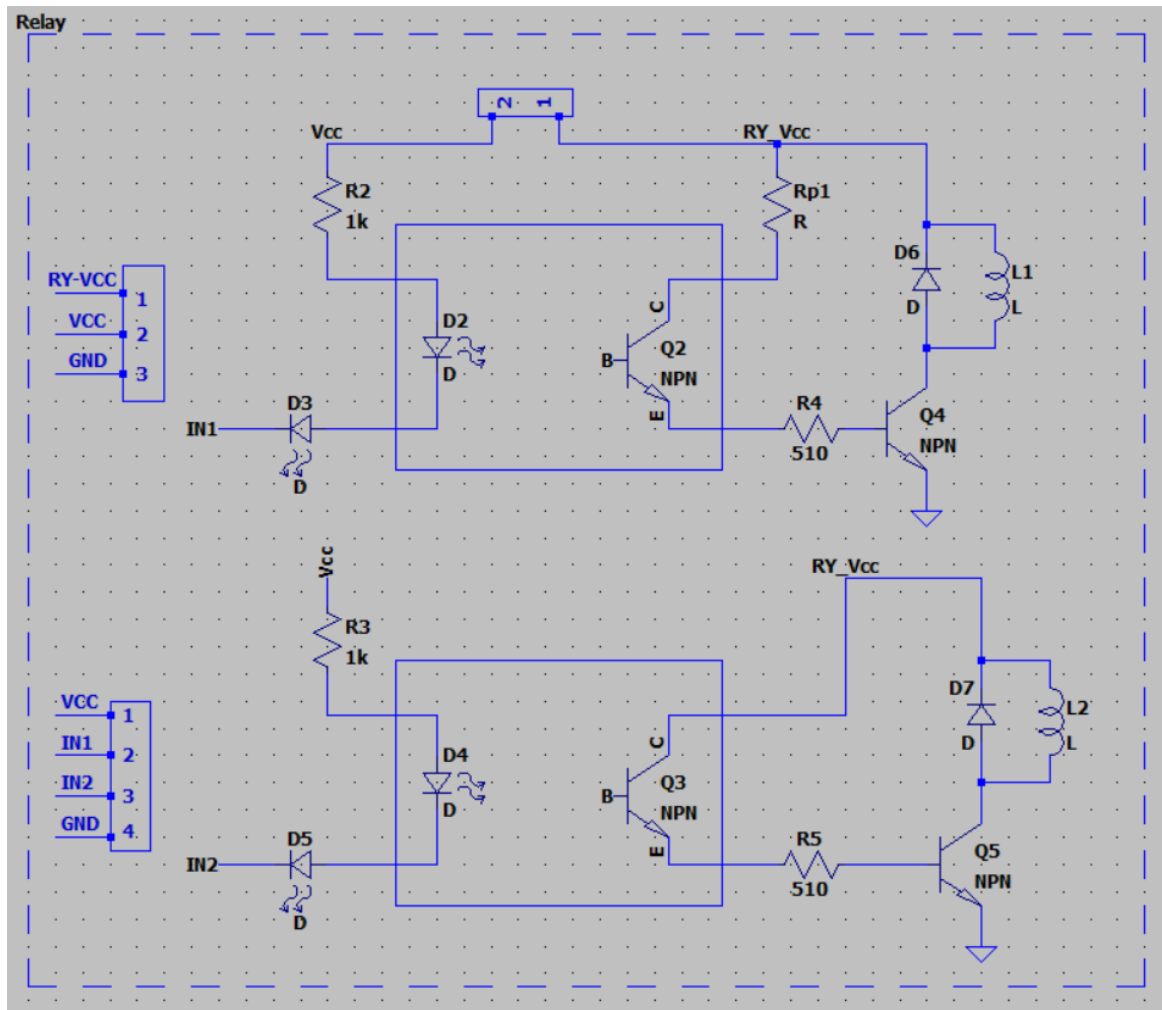


Figure 28. Relay's electrical scheme.

3.3.7. LED Light

After testing the system, there was the need to include some visualization in case some error happened with the system. With this thought in mind, it was added a Led light to the system that would turn on in case of malfunction of the DHT11 or the pzem or even with the mqtt communication.

4. Cofy-Cookie Implementation

This chapter details the implementation process of the Cofy-Cookie system. It includes assembling the Cofy-Cookie device, developing code to handle sensor measurements, Wi-Fi connectivity, and MQTT protocol integration with the broker. The chapter delves deeply into the mechanisms used for transmitting measurements to the server.

4.1. Hardware Implementation

The Cofy-Cookie is the the module that deals with the sensors of the project, connecting the sensors to the ESP32-S2.

After consideration of each component's pins and ways of communication, the proposed circuit schematic can be observed in **Figure 29**, mounted on two breadboards, and tested.

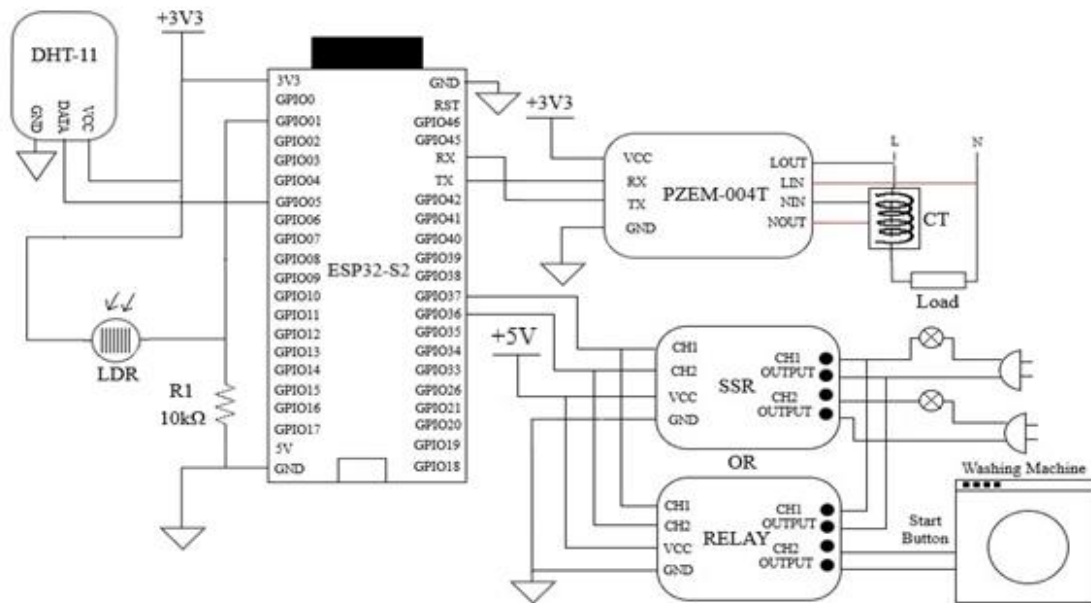


Figure 29. *Electronic Circuit using the ESP32-S2.*

The system will also be implemented using an ESP32-C3, as shown in **Figure 30**. It is the same circuit, on a different microcontroller. The MCU's are both from the same manufacturer and their development tools are the same. Thus, it is possible to reuse the same development to have alternative implementations.

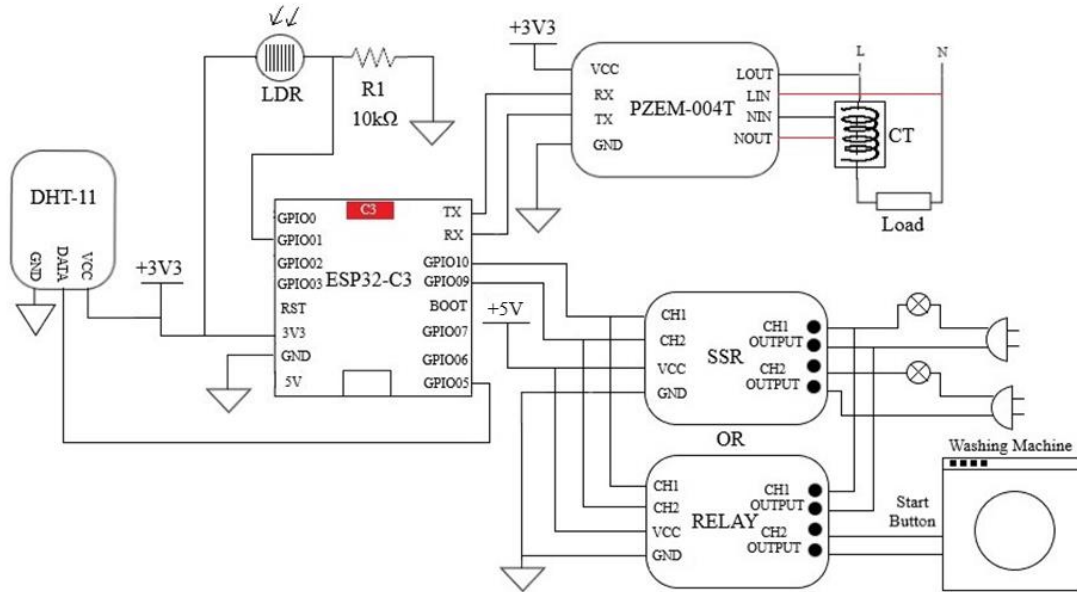


Figure 30. *Electronic Circuit using the ESP32-C3.*

The Cofy-Cookie was mounted on a breadboard connecting all the sensors to the ESP32-S2. **Figure 31** shows the Cofy-Cookie used to test the project, mounted on a breadboard.

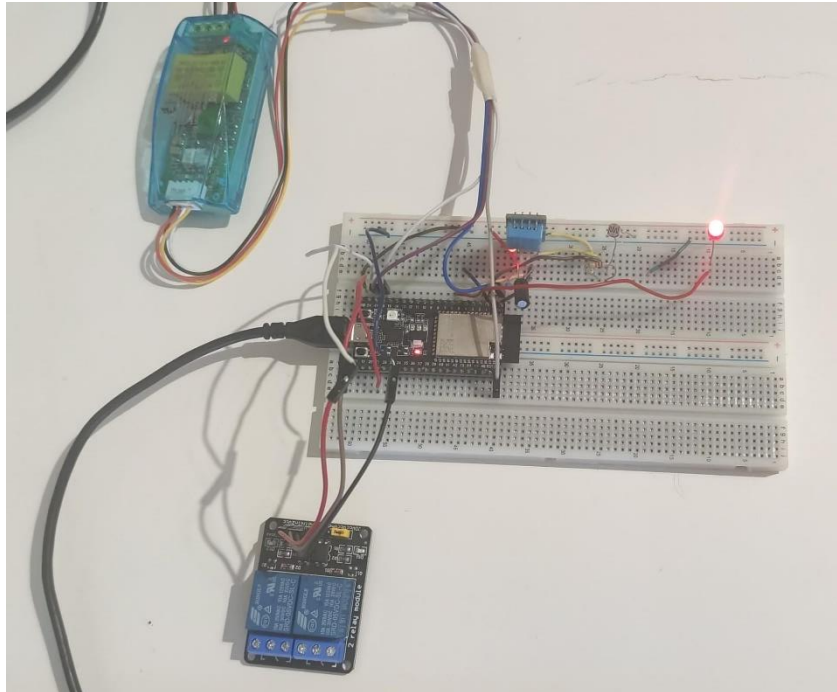


Figure 31. *Cofy-Cookie Mounted on Breadboard.*

4.2. Firmware

The firmware used to program the board used, ensuring the Wi-Fi connection and the MQTT protocol communication is the Arduino IDE due to its ease of use and extensive library support, which facilitates the integration of sensors into projects. This software makes it easy to write programs to implement on different boards, use preinstalled libraries, compile, and debug.

The code consists of 3 functions for reading the measurements. One for the *DHT11*, reading temperature and humidity measurements, one for the *PZEM-004t*, reading all the electrical values and one for the *LDR*, reading the light measurements, having the last one converting the analogue value into a percentage.

Dealing with the relay is done by reading the messages received by MQTT on the topic defined for the relay, dealing with each message in different ways, and turning the relay on or off.

Other than these main functions, there was the need for a function to connect the esp32 to the Wi-Fi and the configuration of the MQTT client.

The program reads the sensors in the main loop and publishes them on the indicated topics.

The Flow chart for the project itself is shown in attachment 1 of the report.

4.3. Wi-Fi Communication

Wi-Fi in this project plays a role in enabling communication between the ESP32 microcontroller and an MQTT broker. The primary purpose of integrating Wi-Fi is to facilitate remote monitoring and control of various sensors and devices connected to the system.

Initially, the ESP32 was configured to connect to a designated Wi-Fi network using the `setup_wifi()` function. This initialisation step enabled the ESP32 to establish a stable network connection for MQTT communication. A router was configured with network credentials, including the SSID and password, to ensure a secure connection. Using the Wi-Fi library available for the ESP32, the device was set up to connect to the selected Wi-Fi network. This process involved initiating the connection and verifying successful connectivity.

4.4. MQTT Protocol

This project utilizes the MQTT protocol to establish communication between the ESP32 microcontroller and a Raspberry Pi running Home Assistant, which hosts the MQTT broker. Several steps were undertaken to ensure a reliable connection.

The MQTT client on the ESP32 was set up to establish a connection with the MQTT broker running on the Raspberry Pi. This setup involved specifying the broker's IP address and utilizing the default MQTT port, 1883.

To handle incoming MQTT messages, a callback function named `callback()` was implemented. This function processes messages received from subscribed topics, enabling the ESP32 to respond to commands and data requests from Home Assistant or other MQTT-enabled devices.

In this project, specific MQTT topics were defined to facilitate the transmission of sensor data and control signals between the ESP32 and the MQTT broker:

1. **home/sensor/data**

This topic publishes a consolidated JSON object containing the following sensor readings:

- Temperature (DHT11) in degrees Celsius
- Humidity (DHT11) as a percentage
- Luminosity (LDR) as a percentage
- Lumens calculated from the LDR sensor
- Voltage, current, power, energy, and frequency from the PZEM-004T

2. **home/relay/command**

This topic accepts commands to control the relay connected to the ESP32. Messages like "ON" to activate or "OFF" to deactivate the relay dictate its state. Subscribing to this topic allows the ESP32 to execute commands received via MQTT.

3. **home/relay/state**

This topic publishes the current state of the relay (ON or OFF) in response to the relay being toggled either via a command or a sensor-driven action.

Each MQTT topic in this project enhances integration and facilitates efficient operation within the MQTT-based home automation system by allowing the ESP32 to send real-time sensor data to Home Assistant and respond to control commands for the relay. The system also ensures reliability by detecting sensor errors, indicated by an error LED, when issues arise with sensor readings.

5. Cofy-Box Implementation

This chapter details the implementation process of the Cofy-Box system. It covers the assembly of the Cofy-Box itself and provides a step-by-step explanation of installing and using Home Assistant on the Raspberry Pi to integrate our system.

5.1. Hardware Implementation

The Cofy-Box functions as the central server module within the system. It consists of a Raspberry Pi connected to the Wi-Fi network established by a configured router. The Raspberry Pi operates an MQTT broker and hosts the Home Assistant application. This setup enables users to access project-specific functionalities, including monitoring electricity and environmental measurements captured by the Cofy-Cookie's sensors. Additionally, users can activate devices such as home appliances using a button interface provided by the Cofy-Box. In **Figure 32**, Raspberry Pi 5 *used as server*, is the Raspberry Pi 5 used as a server.

The Raspberry was also measured with our system in terms of electricity and the raspberry pi5 showed a consumption of 3.10 Watts and a current measure of 40mA.



Figure 32. *Raspberry Pi 5 used as server.*

5.2. Raspberry Setup

The Raspberry Pi Imager was installed and a 64GB SD card was used to set up the Raspberry Pi Imager. The Raspberry Pi OS Lite, which requires at least 16GB of space, was installed on the SD card. During the OS installation process, a name was assigned to the host, and the wireless LAN configuration was set using the previously established network settings.

5.3. Home Assistant Setup

After testing the Wi-Fi connection on the Raspberry Pi, CasaOS was installed via the command line. CasaOS allows the installation of various applications, with Home Assistant being the primary application for this project.

Once CasaOS is installed, the next step is to enter the Raspberry Pi's IP address in a browser, which will redirect to the account creation page on CasaOS. Following this, Home Assistant can be installed from the app store and opened like any other application.

5.3.1. Configuration YAML

The configuration YAML file is used in Home Assistant to define and manage various integrations, including sensors and switches. In this project, it sets up MQTT sensors for monitoring temperature, humidity, luminosity, voltage, current, power, energy, and frequency, as well as an MQTT-controlled switch. This configuration enables real-time data collection and remote control of devices

5.3.2. Home Assistant Overview

The configuration.yaml file in Home Assistant allows users to easily customize and manage the application according to their preferences. In this example, a 'HOME' tab displays electricity and environment metrics, while a 'Bedroom' tab provides the same measurements and includes the ability to control a button connected to the circuit's relay. There's also a section with the statistics, which provides temporal graphics of each measurement.

In this Home section there was also tested a closed source implementation from shelly, that was tested with a DHT11 connected to it.

Figure 33 shows the Home Assistant Home page showing the user every measurement from the sensors.

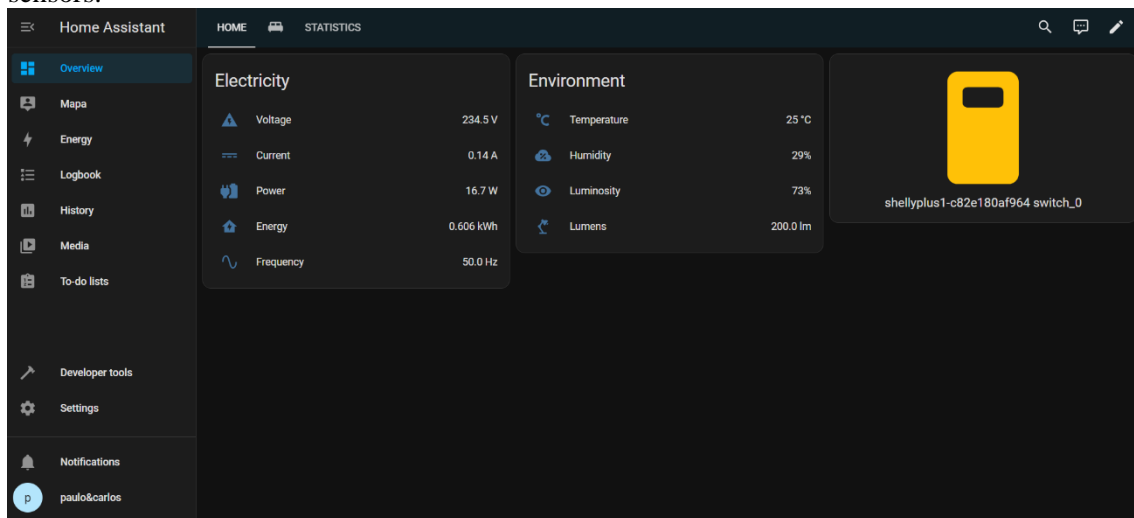


Figure 33. Home Assistant Home Page.

5.3.3. Security

Security is a part of the project. In Home Assistant, administrators can create different user accounts and manage permissions, ensuring that only authorized individuals can access specific features. This setup allows the admin to control who can interact with various parts of the Home Assistant system, preventing unauthorized changes and maintaining system integrity.

5.3.4. Different Devices Usage

The full system was tested on different devices such as smartphones, computers and tablets and seen working in any of them, proving it can be easily accessed by any user with the right credentials, on any device.

6. Results & Discussion

This chapter presents the results obtained from the implementation of the Cofy-Cookie and Cofy-Box systems. It also includes a discussion of the findings and challenges encountered. The primary focus is on the performance of the sensor data collection, Wi-Fi and MQTT communication, and system integration with Home Assistant.

6.1. Sensor Data Collection

The Cofy-Cookie module successfully collected data from various sensors, including temperature, humidity, light intensity, and electrical parameters. The data was continuously monitored and transmitted to the Home Assistant for visualization and control.

Temperature and Humidity: The DHT11 sensor provided temperature and humidity readings. **Figure 34** shows the temperature and humidity values shown by the Cofy-Cookie.

```
Publishing data:
{"temperature":25,"humidity":32
Publishing data:
{"temperature":25,"humidity":24
Publishing data:
{"temperature":25,"humidity":24
```

Figure 34. *Temperature and Humidity Measurements.*

Light Levels: The LDR sensor measured ambient light levels, converting the analog values into percentages. It was tested in bright light using a smartphone's lantern and the sunlight exposure, medium light using ambient light in the afternoon of a summer day and completely covered indicated very low light using a pen cap over the sensor and without sunlight exposure, this is shown in **Figure 35**.

```
luminosity":7,"lumens":8.0

luminosity":34,"lumens":13.0

luminosity":78,"lumens":200.0

luminosity":95,"lumens":10000.0
```

Figure 35. *Luminosity Values.*

Electrical Parameters: The PZEM-004T sensor measured voltage, current, power, frequency and energy consumption. This sensor was tested by reading the measurements with a computer as its load for different power consumption profiles. The measured values can be seen in **Figure 36**.

```
"voltage":234.8,"current":0.12,"power":12.90,"energy":0.608,"frequency":50.0}

"voltage":234.8,"current":0.10,"power":11.10,"energy":0.608,"frequency":50.0}

"voltage":234.8,"current":0.00,"power":0.00,"energy":0.608,"frequency":50.0}

"voltage":234.8,"current":0.00,"power":0.00,"energy":0.608,"frequency":50.0}
```

Figure 36. *Electricity Measurements.*

6.2. Wi-Fi and MQTT Communication

The ESP32 microcontroller connected to the designated Wi-Fi network without any issues. The MQTT protocol facilitated reliable communication between the Cofy-Cookie and the Home Assistant running on the Raspberry Pi.

- **Connection Stability:** The Wi-Fi connection remained stable throughout the testing period, with minimal disconnections.
- **Latency:** There was no relevant latency observed.
- **Message Reliability:** All MQTT messages were successfully transmitted and received, with no data loss or corruption.

This communication can be seen on **Figure 37**, a capture taken with the application wireshark with a filter showing only the messages sent to the MQTT broker.

No.	Time	Source	Destination	Protocol	Length	Info
34	4.649603	192.168.7.112	192.168.7.116	MQTT	219	Ping Request, Publish Received (id=55186)
35	4.652110	192.168.7.112	192.168.7.116	MQTT	213	Ping Response
36	4.652194	192.168.7.116	192.168.7.112	TCP	54	6378 → 8123 [ACK] Seq=1 Ack=325 Win=251 Len=0
41	5.450775	192.168.7.112	192.168.7.116	TCP	214	(Continuation to #35) 8123 → 6378 [PSH, ACK] Seq=325 Ack=1 Win=249 Len=160
42	5.702294	192.168.7.116	192.168.7.112	TCP	54	6378 → 8123 [ACK] Seq=1 Ack=485 Win=256 Len=0
49	7.649500	192.168.7.112	192.168.7.116	TCP	207	(Continuation to #35) 8123 → 6378 [PSH, ACK] Seq=485 Ack=1 Win=249 Len=153
50	7.691934	192.168.7.116	192.168.7.112	TCP	54	6378 → 8123 [ACK] Seq=1 Ack=638 Win=256 Len=0
128	24.338702	192.168.7.116	192.168.7.112	MQTT	70	Ping Request
129	24.344897	192.168.7.112	192.168.7.116	TCP	407	(Continuation to #35) 8123 → 6378 [PSH, ACK] Seq=638 Ack=17 Win=249 Len=353
130	24.377864	192.168.7.116	192.168.7.112	TCP	66	6383 → 8123 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
131	24.382346	192.168.7.112	192.168.7.116	TCP	66	8123 → 6383 [SYN, ACK] Seq=0 Ack=1 Win=32120 Len=0 MSS=1460 SACK_PERM WS=128
132	24.382403	192.168.7.116	192.168.7.112	TCP	54	6383 → 8123 [ACK] Seq=1 Ack=1 Win=65536 Len=0
133	24.382628	192.168.7.116	192.168.7.112	MQTT	1177	Publish Received (id=21332), Publish Ack (id=28270), Publish Ack (id=20052), Publish Release (id=23162), Publish
134	24.384505	192.168.7.112	192.168.7.116	TCP	54	8123 → 6383 [ACK] Seq=1 Ack=1124 Win=31872 Len=0
135	24.386883	192.168.7.112	192.168.7.116	MQTT	292	Publish Ack (id=21584), Publish Release (id=11604), Connect Ack
136	24.386883	192.168.7.112	192.168.7.116	TCP	436	[RST] Seq=1124 Win=0 Len=0
137	24.386915	192.168.7.116	192.168.7.112	TCP	54	6383 → 8123 [ACK] Seq=1124 Ack=621 Win=65024 Len=0
138	24.397891	192.168.7.116	192.168.7.112	TCP	54	6378 → 8123 [ACK] Seq=17 Ack=991 Win=254 Len=0
153	27.347490	192.168.7.112	192.168.7.116	MQTT	1071	Publish Release (id=26415), Publish Complete (id=28526), Publish Received (id=20295), Publish Message[RoundErroror
154	27.355462	192.168.7.112	192.168.7.116	TCP	291	(Continuation to #136) 8123 → 6383 [PSH, ACK] Seq=621 Ack=2141 Win=31872 Len=237
155	27.355462	192.168.7.112	192.168.7.116	TCP	88	(Continuation to #136) 8123 → 6383 [PSH, ACK] Seq=858 Ack=2141 Win=31872 Len=34

Figure 37. *MQTT Wireshark Capture.*

6.3. System Integration

Integration of the Cofy-Cookie with Cofy-Box (Home Assistant) was successful, allowing for real-time monitoring and control of the connected devices.

- **Sensor Data Monitoring:** All sensor readings were accurately displayed on the Home Assistant dashboard.
- **Relay Control:** The relay connected to the ESP32 responded promptly to MQTT commands, enabling remote control of home appliances.
- **User Interface:** The Home Assistant dashboard provided an intuitive interface for users to interact with the system, featuring tabs for overall metrics and specific room controls.

As for the Home Assistant it was configured so the user is able to see every measurement in the Home Page, this can be seen in **Figure 38**.

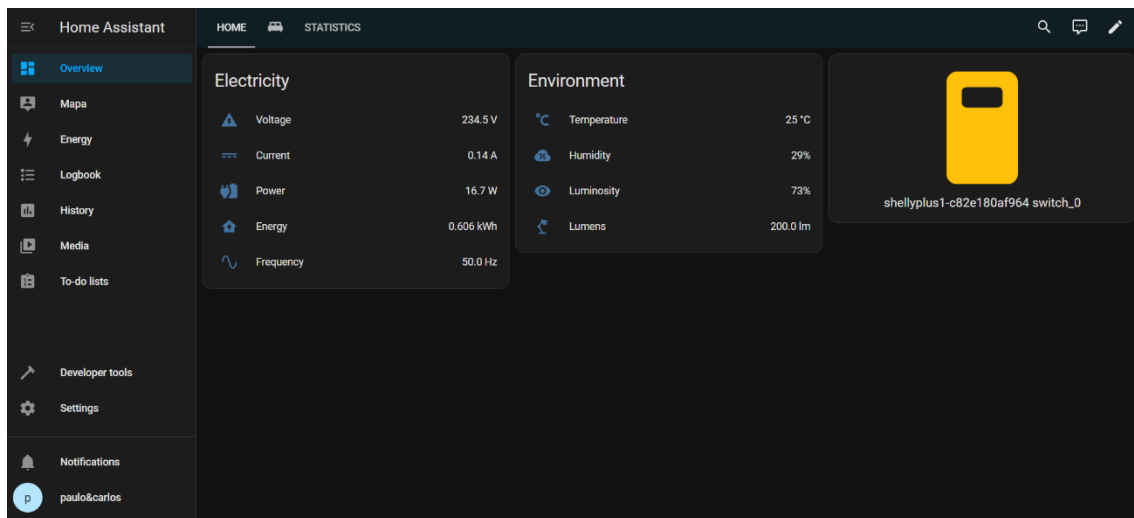


Figure 38. Home Assistant's Home Page.

6.4. Analysis of Results

The data collected from the sensors was consistent and reliable. The temperature and humidity readings were within expected ranges, and the light intensity measurements effectively reflected ambient conditions. Electrical parameter readings provided valuable insights into power consumption values.

Sensor Accuracy: The DHT11, LDR, and PZEM-004T sensors demonstrated acceptable accuracy for home automation applications.

System Performance: The Wi-Fi and MQTT communication were robust, with stable connections and low latency, ensuring timely updates and control.

If the user is allowed, they can also access the Bedroom Page and turn on the lights as well, and see every measurement. This is shown in **Figure 39**.

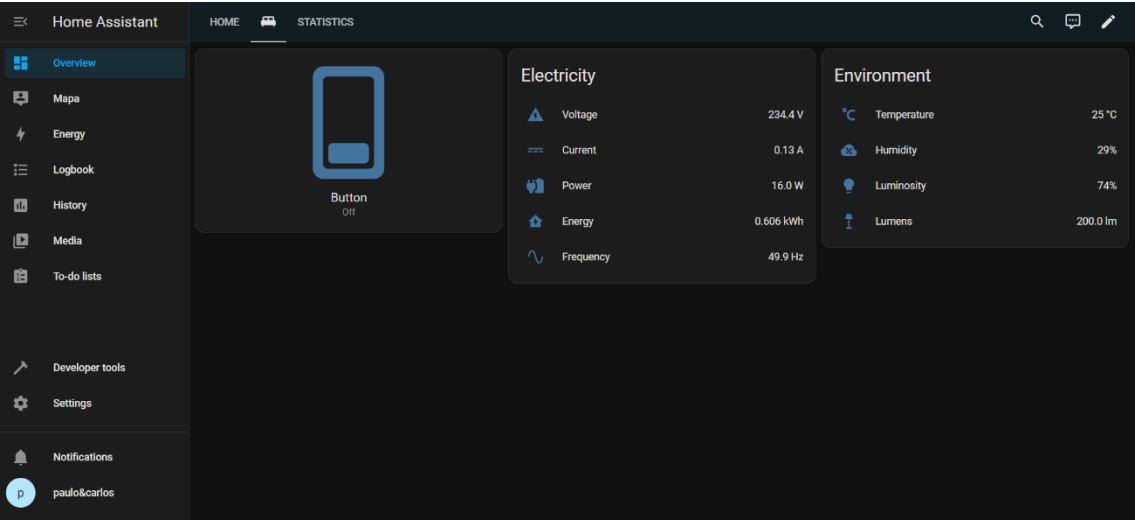


Figure 39. *Home Assistant's Bedroom Page.*

7. Conclusions & Future Work

This chapter provides an overview of the conclusions drawn from the Cofy-Cookie and Cofy-Box implementation and outlines potential areas for future development. The project began with a survey of the required hardware specifications and the establishment of specific milestones to achieve the project's objectives. Following this, the implementation phase involved setting up the server, connecting various components, and ensuring energy-efficient operation. Through this approach, the project successfully met its goals, paving the way for further enhancements and expansions in home automation and energy monitoring systems.

7.1. Conclusions

The implementation of the Cofy-Cookie and Cofy-Box systems demonstrated a successful solution to creating an integrated home automation and energy monitoring solution. Key conclusions from the project are as follows:

1. **Sensor Data Collection:** The Cofy-Cookie module effectively collected and transmitted data from various sensors, including temperature, humidity, light intensity, and electrical parameters. The data was consistent and reliable, enabling real-time monitoring through Home Assistant.
2. **Wi-Fi and MQTT Communication:** The ESP32 microcontroller maintained stable Wi-Fi connectivity and reliable MQTT communication with the Raspberry Pi, ensuring data transmission and control of connected devices.
3. **System Integration:** The integration with Home Assistant allowed for efficient monitoring and control of the system. The intuitive dashboard provided users with easy access to sensor data and remote control capabilities.
4. **Sensor Accuracy and Performance:** The sensors used in the project demonstrated acceptable accuracy for home automation applications. The system's performance in terms of data collection, communication, and control met the project's objectives.
5. **Low Power Consumption and Secure Implementation:** One of the goals of the project was to implement a low power consumption and secure solution. The open code solution demonstrates its security. The fact that it works with the components proves that a low-power solution can be effective.
6. **Scalability:** The project was seen working with multiple different devices such as different Raspberry's and different microcontrollers, showing the potential to use a variety of different components.

Overall, the project achieved its primary goals of creating a reliable and efficient home automation and energy monitoring system, demonstrating its potential for broader applications.

7.2. Future Work

While the project was successful, several areas for future enhancement and expansion have been identified.

1. **Enhanced Sensor Range:** Future work could focus on incorporating sensors with a bigger range of values, such as a sensor that can measure negative temperatures for countries that have these kind of temperatures.
2. **User Interface:** Developing a more customizable and user-friendly interface for Home Assistant could enhance user experience, this could be adding new areas of the house or seeing the measurements in different presentations such as graphic form.
3. **Expanded Functionality:** Adding new sensors and control mechanisms, such as security cameras, advanced energy management features like live-cost for the energy in the area, or additional environmental sensors, for example wind sensors, could be an interesting addition to the system's capabilities.
4. **Security:** Implementing SSL/TLS encryption for MQTT communication enhances security by encrypting data transmitted between devices and the MQTT broker. Embedding static CA certificates in the device firmware ensures the authenticity of the broker while using secure passwords and Regular firmware updates enhance the system's defenses against potential threats.
5. **Integration with Other Systems:** Investigating the integration of the Cofy-Cookie and Cofy-Box systems with other home automation platforms and IoT devices could increase compatibility and functionality. This could be the different devices that have been talked in this report like shelly devices or tuya devices.

Annexes

1. Flowchart of the implemented code for the Cofy-Cookie

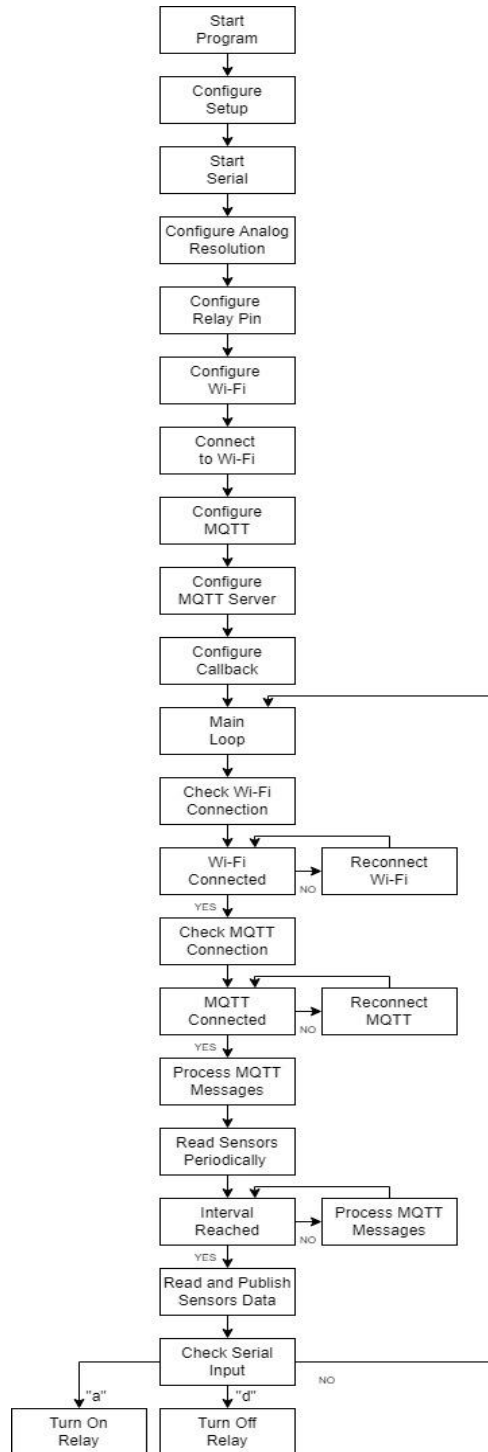


Figure 40. Flowchart of the Cofy-Cookie.

2. Raspberry Pi & Home Assistant installation tutorial

This chapter provides a tutorial of how to install Home Assistant using a Raspberry Pi. The steps are as follows.

1. Remove the SD card of the Raspberry Pi;
2. Insert the SD card into a computer with access to the Internet;
3. Download the Raspberry Pi Imager from the Raspberry Pi official website: <https://www.raspberrypi.com/software/>;
4. Open Raspberry Pi Imager;
5. Select which device you wish to use in this tab as shown in **Figure 41**:

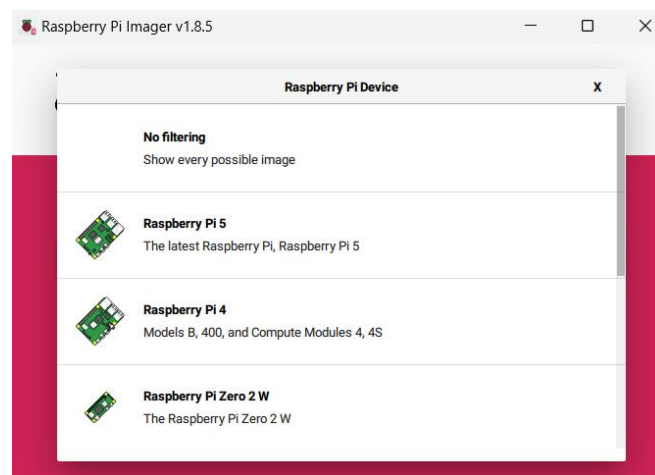


Figure 41. *Choosing Raspberry Pi Device.*

6. The Raspberry Pi OS used is located in the Operating System tab, as shown in **Figure 42** and **Figure 43**.

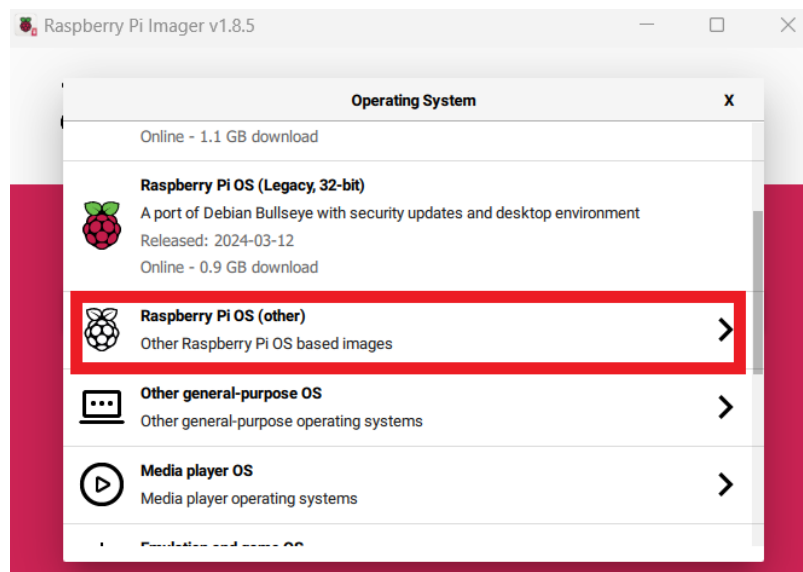


Figure 42. *Choosing the Operating System.*

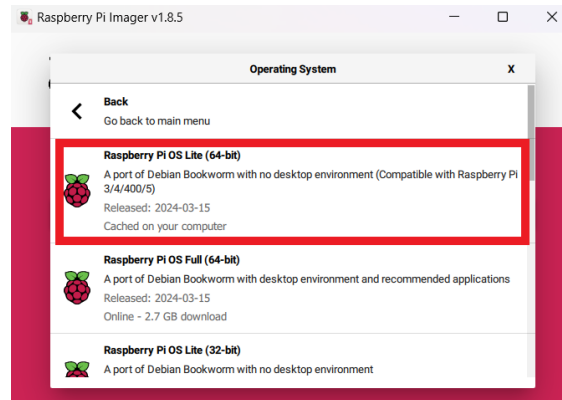


Figure 43. *Choosing the Operating System (64-bit).*

7. Confirm your choices, and then select the edit settings button as seen in **Figure 44**.

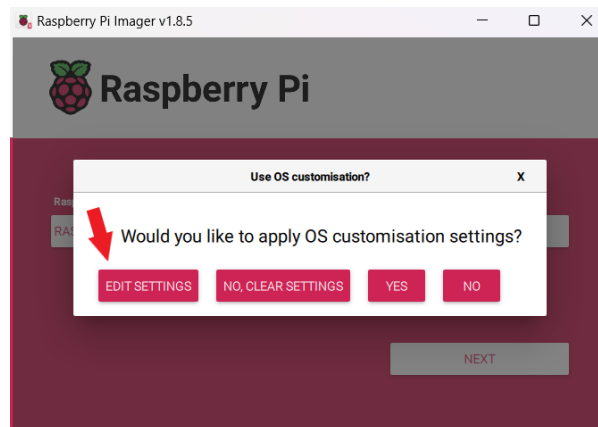


Figure 44. *Selecting edit settings.*

8. In the next tab, pick the names and passwords to add to your device and confirm the network that you will use to communicate with your Raspberry Pi, as seen on **Figure 45**. After these steps hit the save button and start the image installation.

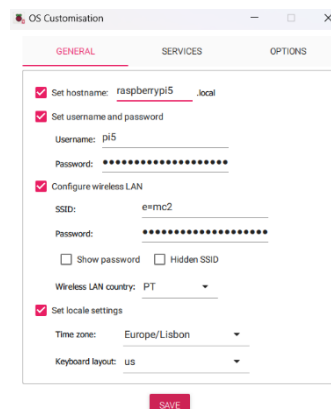
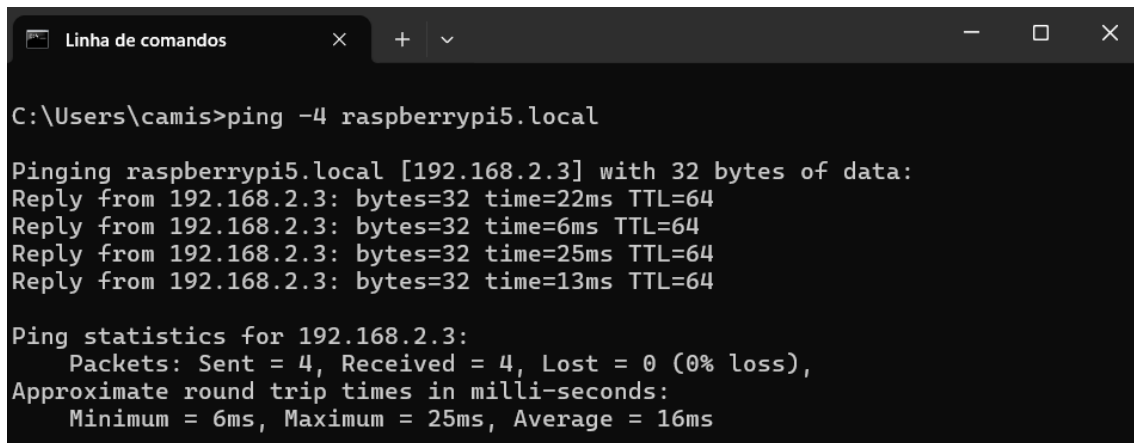


Figure 45. *Wireless LAN Configuration.*

9. After the image installation is completed, remove the SD card from the computer, and insert in the Raspberry Pi again, and turn it on.

10. Using a computer connected to the network that was previously configured in the Raspberry Pi, using the command prompt of your computer, try to reach the Raspberry using the command: `ping -4 chosen_hostname.local`. **Figure 46** shows the command.



```
C:\Users\camis>ping -4 raspberrypi5.local

Pinging raspberrypi5.local [192.168.2.3] with 32 bytes of data:
Reply from 192.168.2.3: bytes=32 time=22ms TTL=64
Reply from 192.168.2.3: bytes=32 time=6ms TTL=64
Reply from 192.168.2.3: bytes=32 time=25ms TTL=64
Reply from 192.168.2.3: bytes=32 time=13ms TTL=64

Ping statistics for 192.168.2.3:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 6ms, Maximum = 25ms, Average = 16ms
```

Figure 46. *Ping Command.*

11. Now it is possible to see the Raspberry Pi address on the chosen network. Accessing the Raspberry Pi using the command: `ssh chosen_name@Raspberry_address`, then hit the enter button and insert the chosen password. **Figure 47** represents the command.



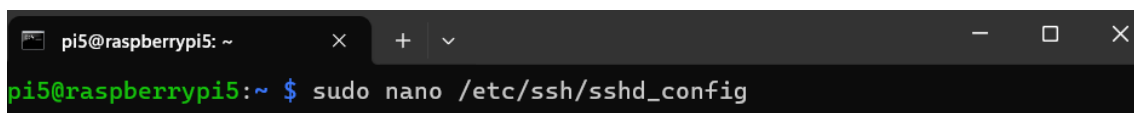
```
C:\Users\camis>ssh pi5@192.168.2.3
pi5@192.168.2.3's password:
Linux raspberrypi5 6.6.20+rpt-rpi-v8 #1 SMP PREEMPT Debian 1:6.6.20-1+rpt1 (
2024-03-07) aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue May  7 15:47:51 2024 from 192.168.2.2
pi5@raspberrypi5:~ $
```

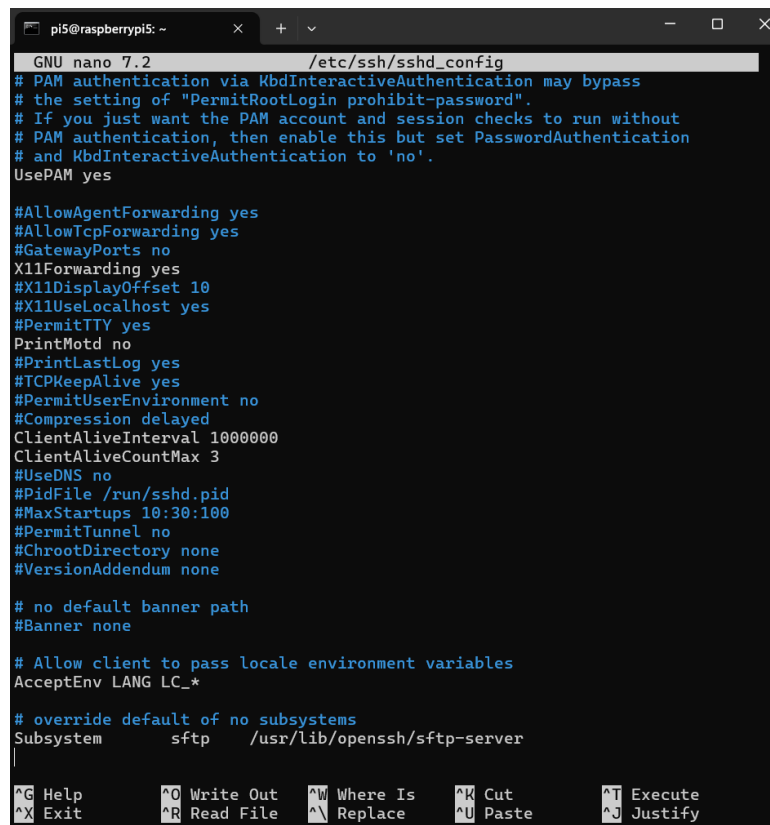
Figure 47. *ssh Command.*

12. Insert the following command and fix the following parameters: `ClientAliveInterval` and `ClientAliveCountMax` must not be commented and the first one must have value 1000000 instead of 0. Then save this changes and exit with the controls `ctrl+o` and `ctrl+x` respectively. This is shown in **Figure 48** and **Figure 49**.



```
pi5@raspberrypi5: ~
pi5@raspberrypi5:~ $ sudo nano /etc/ssh/sshd_config
```

Figure 48. *ssh Configuration.*

A terminal window titled 'pi5@raspberrypi5: ~' showing the nano editor editing the file '/etc/ssh/sshd_config'. The file contains various configuration options for the SSH daemon, such as authentication methods, forwarding, and subsystems. The bottom of the window shows nano editor shortcuts like ^G Help, ^O Write Out, etc.

```
pi5@raspberrypi5: ~
GNU nano 7.2 /etc/ssh/sshd_config
# PAM authentication via KbdInteractiveAuthentication may bypass
# the setting of "PermitRootLogin prohibit-password".
# If you just want the PAM account and session checks to run without
# PAM authentication, then enable this but set PasswordAuthentication
# and KbdInteractiveAuthentication to 'no'.
UsePAM yes

#AllowAgentForwarding yes
#AllowTcpForwarding yes
#GatewayPorts no
X11Forwarding yes
#X11DisplayOffset 10
#X11UseLocalhost yes
#PermitTTY yes
PrintMotd no
#PrintLastLog yes
#TCPKeepAlive yes
#PermitUserEnvironment no
#Compression delayed
ClientAliveInterval 1000000
ClientAliveCountMax 3
#UseDNS no
#PidFile /run/sshd.pid
#MaxStartups 10:30:100
#PermitTunnel no
#ChrootDirectory none
#VersionAddendum none

# no default banner path
#Banner none

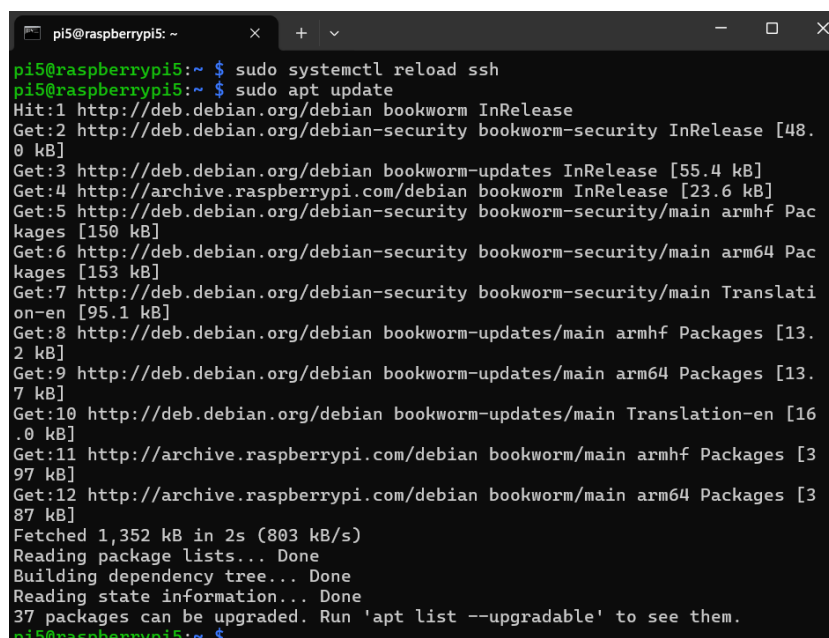
# Allow client to pass locale environment variables
AcceptEnv LANG LC_*

# override default of no subsystems
Subsystem        sftp        /usr/lib/openssh/sftp-server

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify
```

Figure 49. *ssh Configuration.*

13. Do the commands separately as it is shown in the figure below and as it is possible to see in the last line of the command response if there are packages to upgrade do the following command: `sudo apt upgrade`. **Figure 50** shows the update being made, and **Figure 51** shows the command to upgrade the packages.

A terminal window titled 'pi5@raspberrypi5: ~' showing the execution of two commands. The first command is 'sudo systemctl reload ssh' and the second is 'sudo apt update'. The output of 'apt update' shows the process of fetching package lists from various sources and determining that 37 packages can be upgraded.

```
pi5@raspberrypi5:~ $ sudo systemctl reload ssh
pi5@raspberrypi5:~ $ sudo apt update
Hit:1 http://deb.debian.org/debian bookworm InRelease
Get:2 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:3 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:4 http://archive.raspberrypi.com/debian bookworm InRelease [23.6 kB]
Get:5 http://deb.debian.org/debian-security bookworm-security/main armhf Packages [150 kB]
Get:6 http://deb.debian.org/debian-security bookworm-security/main arm64 Packages [153 kB]
Get:7 http://deb.debian.org/debian-security bookworm-security/main Translation-en [95.1 kB]
Get:8 http://deb.debian.org/debian bookworm-updates/main armhf Packages [13.2 kB]
Get:9 http://deb.debian.org/debian bookworm-updates/main arm64 Packages [13.7 kB]
Get:10 http://deb.debian.org/debian bookworm-updates/main Translation-en [16.0 kB]
Get:11 http://archive.raspberrypi.com/debian bookworm/main armhf Packages [397 kB]
Get:12 http://archive.raspberrypi.com/debian bookworm/main arm64 Packages [387 kB]
Fetched 1,352 kB in 2s (803 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
37 packages can be upgraded. Run 'apt list --upgradable' to see them.
pi5@raspberrypi5:~ $
```

Figure 50. *Update Command.*

```
pi5@raspberrypi5:~ $ sudo apt upgrade
```

Figure 51. Upgrade Command.

14. After the upgrade finish copy the following URL that will install CasaOS in your Raspberry Pi. This is the image containing the operating system where the Home Assistant application can be installed on. The command is represented in **Figure 52**.

```
pi5@raspberrypi5:~ $ curl -fsSL https://get.casaos.io | sudo bash
```

Figure 52. CasaOS installation.

15. After installing CasaOS in your Raspberry Pi do the following commands in your computer prompt to see if the image is running by checking the Active parameter after the command: `sudo systemctl status casaos`. The status of the CasaOS image is seen in **Figure 53**.

```
pi5@raspberrypi5:~ $ sudo reboot now

Broadcast message from root@raspberrypi5 on pts/1 (Tue 2024-05-07 16:46:11 W
EST):

The system will reboot now!

pi5@raspberrypi5:~ $ Connection to 192.168.2.3 closed by remote host.
Connection to 192.168.2.3 closed.

C:\Users\camis>ssh pi5@192.168.2.3
pi5@192.168.2.3's password:
Linux raspberrypi5 6.6.28+rpt-rpi-v8 #1 SMP PREEMPT Debian 1:6.6.28-1+rpt1 (
2024-04-22) aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue May  7 16:21:43 2024 from 192.168.2.2
pi5@raspberrypi5:~ $ sudo systemctl status casaos
● casaos.service - CasaOS Main Service
   Loaded: loaded (/lib/systemd/system/casaos.service; enabled; preset: ena
   Active: active (running) since Tue 2024-05-07 16:46:21 WEST; 3min 58s
   Main PID: 777 (casaos)
     Tasks: 7 (limit: 3917)
        CPU: 3.359s
    CGroup: /system.slice/casaos.service
            └─777 /usr/bin/casaos -c /etc/casaos/casaos.conf

May 07 16:49:58 raspberrypi5 casaos[777]: /bin/bash -c source /usr/share/ca
May 07 16:50:00 raspberrypi5 casaos[777]: 消息来了, message: {"type":"ping"}
May 07 16:50:03 raspberrypi5 casaos[777]: /bin/bash -c source /usr/share/ca
May 07 16:50:03 raspberrypi5 casaos[777]: /bin/bash -c source /usr/share/ca
May 07 16:50:08 raspberrypi5 casaos[777]: /bin/bash -c source /usr/share/ca
May 07 16:50:08 raspberrypi5 casaos[777]: /bin/bash -c source /usr/share/ca
May 07 16:50:13 raspberrypi5 casaos[777]: /bin/bash -c source /usr/share/ca
```

Figure 53. CasaOS status.

16. Insert the IP address of the Raspberry Pi in a new searching tab in your computer and if this is the response that you obtain then the image was successfully installed. **Figure 54** shows the page to create the account.

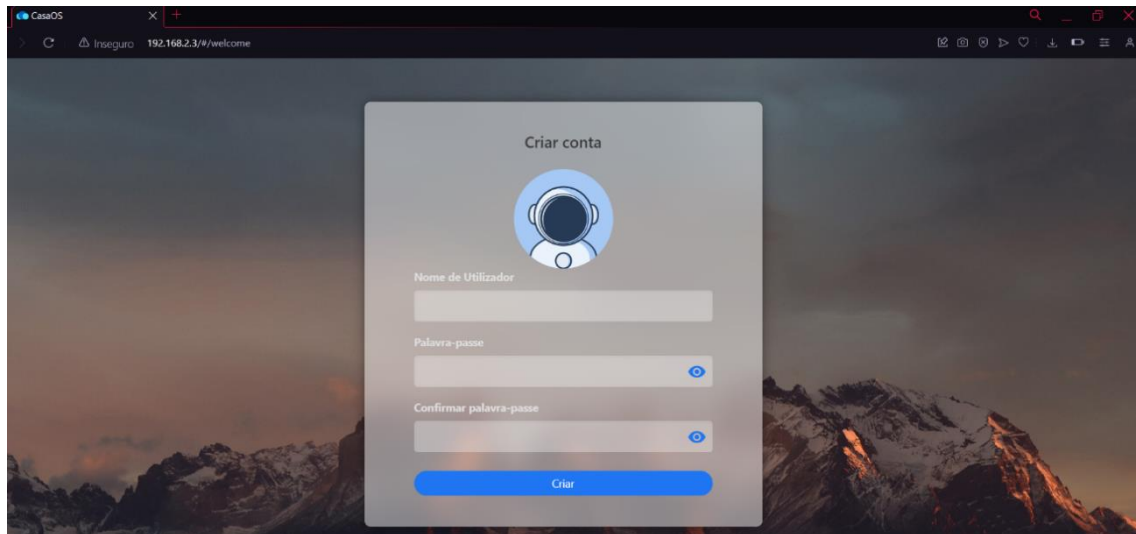


Figure 54. *CasaOS Creating Account.*

17. After creating your account you will reach the main menu showing the files installed as well as the App Store to install every application available. This overview page is shown in **Figure 55**.

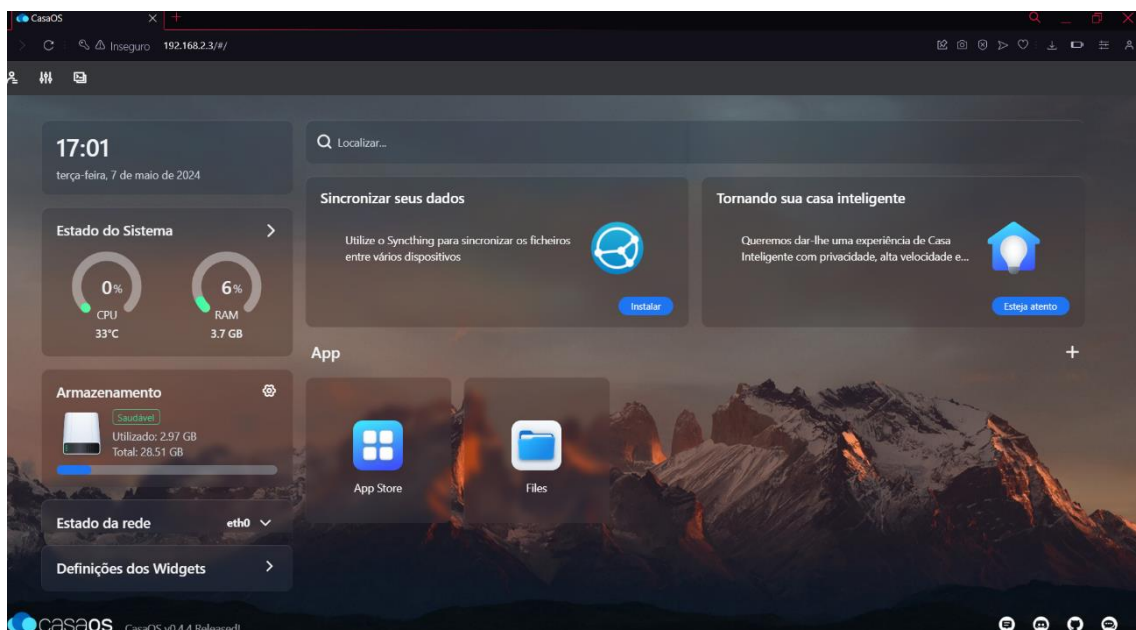


Figure 55. *Overview Page.*

18. After installing the the Home Assistant, open it's application and create your account. After having a similar menu as the shown bellow your Home Assistant is officially operative and ready to customize. **Figure 56** shows the Home Assistant Page.

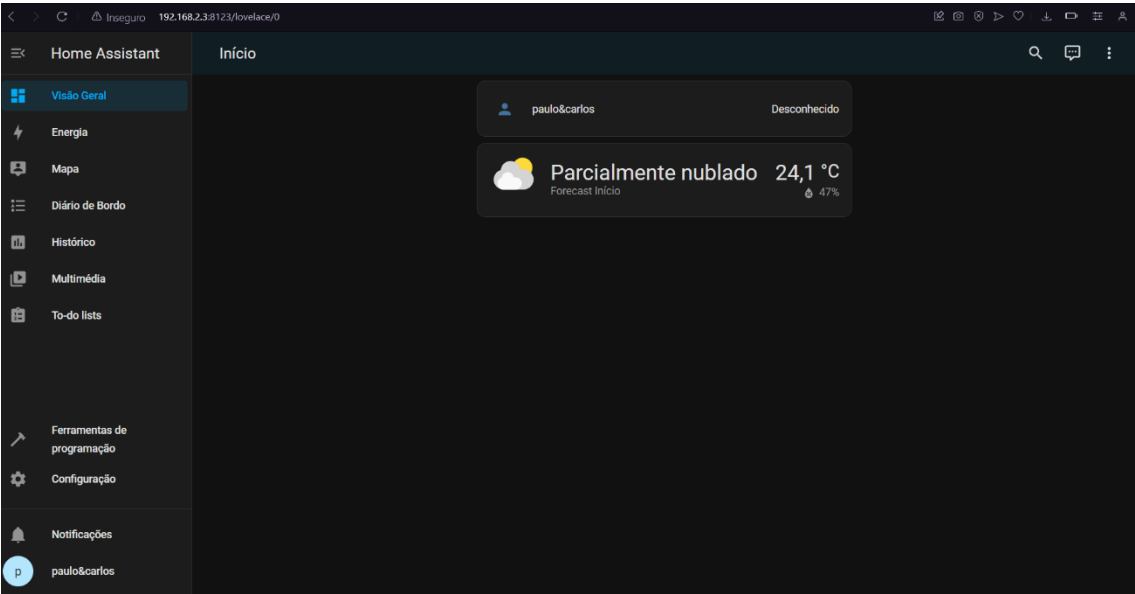


Figure 56. *Home Assistant Page.*

3. GitHub & Cloud Repository For The Project.

<https://github.com/prodrigues0201/Home-Automation-System>

<https://iselpt->

my.sharepoint.com/:f:/g/personal/a47118_alunos_isel_pt/EtBTJtNIj19IsXWTkGRipasBaDdFPGp-ZwGF3Frw040NkA?e=ZUXREc

References

- [1] Aylott, Ben, et al. DELIVERABLE D3.1 Final Design/Specification Project Title: Smart Building Ecosystem for Energy Communities.
- [2] Srivastava D., Kesarwani A., & Dubey S. (2018). Measurement of Temperature and Humidity by using Arduino Tool and DHT11. *International Research Journal of Engineering and Technology*, 5(12), 876-878. www.irjet.net
- [3] Harahap, P., Pasaribu, F. I., & Adam, M. (2020). Prototype Measuring Device for Electric Load in Households Using the Pzem-004T Sensor. *Budapest International Research in Exact Sciences (BirEx) Journal*, 2(3), 347-361.
- [4] Pratama, Erik Wahyu, and Agus Kiswantono. "Electrical Analysis Using ESP-32 Module in Realtime." *JEECS (Journal of Electrical Engineering and Computer Sciences)*, vol. 7, no. 2, 13 Jan. 2023, pp. 1273–1284, <https://doi.org/10.54732/jeeecs.v7i2.21>.
- [5] Rocha, Miguel, and Pedro Silva. IoT System for pH Monitoring in Industrial Facilities. July 2023.
- [6] Peña, Eric, and Mary Grace Legaspi. "Uart: A hardware communication protocol understanding universal asynchronous receiver/transmitter." *Visit Analog* 54.4 (2020): 1-5.
- [7] "Orange Pi Zero Plus vs Raspberry Pi 4 Model B: What Is the Difference?" *VERSUS*, [versus.com/en/orange-pi-zero-plus-vs-raspberry-pi-4-model-b](https://www.versus.com/en/orange-pi-zero-plus-vs-raspberry-pi-4-model-b). Accessed 20 Mar. 2024.
- [8] "Chip Series Comparison - ESP32-S3 - — ESP-IDF Programming Guide V5.0 Documentation." *Docs.espressif.com*, docs.espressif.com/projects/esp-idf/en/v5.0/esp32s3/hw-reference/chip-series-comparison.html. Accessed 20 Mar. 2024.
- [9] JackSoldanoJacksoldano.com. "Sensor Comparison: DHT11 vs DHT22 vs BME680 vs DS18B20." *Instructables*, www.instructables.com/Sensor-Comparison-DHT11-Vs-DHT22-Vs-BME680-Vs-DS18/. Accessed 21 Mar. 2024.
- [10] M1. "Solid State Relay vs. Mechanical Relay - What Is Different?" *GEYA Electrical Equipment Supply*, 18 Oct. 2022, www.geya.net/solid-state-relay-vs-mechanical-relay-what-is-different/.
- [11] Khan, M.A.; Khan, M.A.; Jan, S.U.; Ahmad, J.; Jamal, S.S.; Shah, A.A.; Pitropakis, N.; Buchanan, W.J. A Deep Learning-Based Intrusion Detection System for MQTT Enabled IoT. *Sensors* 2021, 21, 7016. <https://doi.org/10.3390/s21217016>
- [12] Yifeng Liu and Eyhab Al-Masri. 2022. Slow Subscribers: a novel IoT-MQTT based denial of service attack. *Cluster Computing* 26, 6 (Dec 2023), 3973–3984. <https://doi.org/10.1007/s10586-022-03788-9>
- [13] S. Saxena, S. Jain, D. Arora and P. Sharma, "Implications of MQTT Connectivity Protocol for IoT based Device Automation using Home Assistant and OpenHAB," 2019 6th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2019, pp. 475-480.
- [14] DFRobot. GitHub - DFRobot/DFRobot_DHT11. GitHub. Published 2018. Accessed September 4, 2024. https://github.com/DFRobot/DFRobot_DHT11?tab=readme-ov-file

