This project focuses on supervising the update, optimization, and expansion of a home automation system in a single-family house using OpenHAB. Key tasks include updating the software, improving system performance, integrating new smart devices, resolving existing issues, and adding new functionalities. These enhancements aim to ensure better efficiency, reliability, and scalability of the system.

To achieve these objectives, software engineering practices will be applied, including system analysis, project planning, and iterative development. The system will be reviewed to identify weaknesses, and structured updates will be implemented. Device integration and automation features will be developed and tested. All tasks will follow a defined workflow to ensure reliability and maintainability.

Hicham Bouchemma Bouhou is the student in charge of the implementation and configuration of the project, and with this work he finishes his degree in Computer Engineering.

Andrés María Roldán Aranda is the academic head of the present project, and the student's tutor. He is professor in the Departament of Electronics and Computers Technologies

BACHELOR THESIS

Home automation installation supervision using OpenHAB

Hicham Bouchemma Bouhou

COMPUTER ENGINEERING

UNIVERSITY OF GRANADA

Degree in Computer Engineering

Bachelor Thesis

Home automation installation supervision using OpenHAB

Hicham Bouchemma Bouhou

2024/2025

Tutor: Andrés María Roldán Aranda

Credits for the cover: **NASA**.
Printed in Granada, June 2025.

“Home automation installation supervision using OpenHAB”

DEGREE IN
COMPUTER ENGINEERING

Bachelor Thesis

"Home automation installation supervision using OpenHAB"

ACADEMIC COURSE: 2024-2025

Hicham Bouchemma Bouhou

DEGREE IN COMPUTER ENGINEERING

"Home automation installation supervision using OpenHAB"

AUTHOR:

Hicham Bouchemma Bouhou

SUPERVISED BY:

Prof. Andrés Roldán Aranda

DEPARTMENT:

Electronics and Computers Technologies

"Home automation installation supervision using OpenHAB"

Hicham Bouchemma Bouhou

KEYWORDS:

OpenHAB, Domotic, Binding, Amazon Alexa, Docker, Automatization, Centralization, Rules, OpenHAB Cloud, OpenHAB Mobile App, Duplicati, Migration, Pages, InfluxDB.

ABSTRACT:

The main objective of this project is to supervise the installation, update, and improvement of a home automation portal in a single-family house based on OpenHAB. This work continues previous efforts in initializing the system and configuring electronic devices.

The project involves updating the software to the latest versions, ensuring compatibility and security enhancements, and supervising the proper installation and configuration of these updates. Optimizing the system architecture to enhance efficiency, performance, and scalability is another key focus. This includes network reconfiguration, data storage optimization, and improved communication between devices.

Additionally, new smart devices will be selected, installed, and configured to expand the system's capabilities. Troubleshooting and bug fixing are essential tasks to resolve existing issues in the current system configuration and operation.

New functionalities, such as advanced task automation and user interface customization, will be developed to enhance the system further. A regular maintenance plan will be established to ensure the long-term proper functioning of the system. This plan includes constant monitoring, periodic software updates, and automated backups.

This project adapts to current needs in home automation and contributes to education in Computer Engineering, providing a more robust and versatile home automation system.

"Home automation installation supervision using OpenHAB"

Hicham Bouchemma Bouhou

PALABRAS CLAVE:

OpenHAB, Domotic, Binding, Amazon Alexa, Docker, Automatization, Centralization, Rules, OpenHAB Cloud, OpenHAB Mobile App, Duplicati, Migration, Pages, InfluxDB.

RESUMEN:

El objetivo principal de este proyecto es supervisar la instalación, actualización y mejora de un portal domótico en una casa unifamiliar basado en OpenHAB. Este trabajo continúa los esfuerzos previos en la inicialización del sistema y la configuración de dispositivos electrónicos.

El proyecto implica actualizar el software a las versiones más recientes, asegurando compatibilidad y mejoras de seguridad, y supervisar la correcta instalación y configuración de estas actualizaciones. Otro enfoque clave es la optimización de la arquitectura del sistema para mejorar la eficiencia, el rendimiento y la escalabilidad. Esto incluye la reconfiguración de la red, la optimización del almacenamiento de datos y la mejora de la comunicación entre dispositivos.

Además, se seleccionarán, instalarán y configurarán nuevos dispositivos inteligentes para ampliar las capacidades del sistema. La resolución de problemas y la corrección de errores son tareas esenciales para resolver los problemas existentes en la configuración y funcionamiento del sistema actual.

Se desarrollarán nuevas funcionalidades, como la automatización avanzada de tareas y la personalización de la interfaz de usuario, para mejorar aún más el sistema. Se establecerá un plan de mantenimiento regular para asegurar el correcto funcionamiento del sistema a largo plazo. Este plan incluye la monitorización constante, las actualizaciones periódicas del software y las copias de seguridad automáticas.

Este proyecto se adapta a las necesidades actuales en domótica y contribuye a la formación en Ingeniería Informática, proporcionando un sistema domótico más robusto y versátil.

Acknowledgments:

I would like to express my sincere gratitude to all the people who have made the completion of this Bachelor's Thesis possible.

First and foremost, I am deeply grateful to my family, especially my parents, for their unconditional love and unwavering support throughout all these years of life and study. Only they truly understand what it means to raise and inspire a child in a challenging environment—and still find a way to nurture his dreams and help him reach his full potential. Without their strength, belief, and guidance, this achievement would not have been possible.

I would also like to thank my supervisor, Andrés Roldán Aranda, for his valuable support in helping me resolve the issues that arose during the development of this project. His guidance enabled me to overcome my doubts and move forward with confidence.

Lastly, I wish to extend my gratitude to all the professors at the School of Computer Engineering and Telecommunications who have taught me, as well as to the classmates I have had the pleasure of meeting and sharing this important stage of my life with. Their support and companionship have made this long journey more enjoyable and enriching.

Agradecimientos:

Quisiera expresar mi más sincero agradecimiento a todas las personas que han hecho posible la realización de este Trabajo de Fin de Grado, con el cual culmino una etapa muy significativa en mi vida.

En primer lugar, agradezco profundamente a mi familia, y en especial a mis padres, por su amor incondicional y su apoyo constante a lo largo de todos estos años de vida y estudio. Solo ellos saben lo que significa criar y motivar a un niño en un contexto difícil, y aun así encontrar la forma de alimentar sus sueños y sacar lo mejor de él. Sin su esfuerzo, fe y guía, este logro no habría sido posible.

También deseo agradecer a mi tutor, Andrés Roldán Aranda, quien me brindó su ayuda en la resolución de los problemas que surgieron durante el desarrollo del proyecto, permitiéndome así superar mis dudas con éxito.

Finalmente, quiero expresar mi gratitud a todos los profesores de la Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones que me impartieron clases, así como a los compañeros que he tenido el placer de conocer y que me han acompañado durante esta importante etapa de mi vida. Su compañía ha hecho que este largo camino sea mucho más ameno y enriquecedor.

Contents

List of Figures

List of Tables

Glossary

Add-on is a modular software component that extends the functionality of openHAB. Add-ons can provide support for new devices, technologies, user interfaces, automation tools, or integrations with external services. They are essential for customizing openHAB to meet specific smart home needs.

Application Programming Interface is a set of subroutines, functions and procedures (or methods, in object-oriented programming) that provides a certain library to be used by other software as an abstraction layer [50].

Binding can be considered as software adapters, which makes the Things available to your home automation system, in other words it is the communication link between the Thing and your system [1].

Bucket is a named location where time series data is stored. All buckets have a retention period. A bucket belongs to an organization [20].

container is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

DNS Domain Name System; a hierarchical and decentralized naming system that translates human-readable domain names (like `example.com`) into IP addresses used for locating and identifying devices on a network.

Docker is an open source software platform to create, deploy and manage virtualized application containers on a common operating system (OS), with an ecosystem of allied tools [46].

Docker Compose is a tool for defining and running multi-container Docker applications. A YAML file is used to configure the Docker application services. Then, with a single command, all the services in the configuration are created and started [6].

Docker Image is a snapshot or blueprint of the libraries and dependencies required inside a container.

Dockerfile is a simple text file or document that includes a series of instructions that need to be executed consecutively to accomplish the processes necessary for the creation of a new image [4].

Domotic a set of techniques aimed at automating a home, integrating technology in the security, energy management, welfare or communications systems.

Duplicati is a free, open-source backup tool that securely creates encrypted, incremental backups to cloud or local storage.

Field is the key-value pair in an InfluxDB data structure that records metadata and the actual data value [18].

Flux is a powerful scripting language designed specifically to query, analyze, and act on time series data [18].

Hypertext Transfer Protocol is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems [51].

InfluxDB is an open-source time series database (TSDB) optimized for high-ingest and query speeds of timestamped data, commonly used for application metrics, IoT sensor data, and real-time analytics.

InfluxQL is an SQL-like query language designed to work with time series data in InfluxDB.

Internet of Things describes the network of physical objects "things" that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet [38].

Java is a general-purpose, class-based, object-oriented programming language designed for having lesser implementation dependencies. It is a computing platform for application development. Java is fast, secure, and reliable, therefore. It is widely used for developing Java applications in laptops, data centers, game consoles, scientific supercomputers, cell phones, etc [15].

Measurement is the part of the InfluxDB data structure that describes the data stored in the associated fields [18].

Metastore contains internal information about the status of the system. The metastore contains the user information, databases, retention policies, shard metadata, continuous queries, and subscriptions [18].

Migration in software engineering, migration refers to the process of moving data, applications, or systems from one environment to another.

NAT a method used in routers to remap one IP address space into another by modifying network address information in the IP header of packets while they are in transit.

OpenHAB is an open source home automation platform or system that is the controller or center of your Smart Home [43].

OpenHAB Cloud is a companion cloud service and backend for the OpenHAB open-source home automation software. The OpenHAB Cloud backend provides secure remote access and enables OpenHAB users to remotely monitor, control and steer their homes through the internet, collect device statistics of their OpenHABs, receive notifications on their mobile devices or collect and visualize data etc [37].

Organization is a workspace for a group of users. All dashboards, tasks, buckets, members, and so on, belong to an organization [20].

Pages is the new user interface released in OpenHAB version 3, to interact with the items. It is not very explored yet but you can create custom widgets or use the default ones to represent the items [34].

Point represents a single data record, similar to a row in a SQL database table. Each point has a measurement, a tag set, a field key, a field value, and a timestamp; is uniquely identified by its series and timestamp [18].

Proof of Concept an implementation of a certain method or idea to demonstrate its feasibility and practical potential before full-scale development.

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics developed by Guido van Rossum. It was originally released in 1991. Designed to be easy as well as fun, the name "Python" is a nod to the British comedy group Monty Python [47].

Raspberry Pi is a series of small-board, low-cost single-board computers developed in the UK by the Raspberry Pi Foundation, with the aim of putting the power of computing and digital creation into the hands of people around the world [55].

REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer and was created by computer scientist Roy Fielding [42].

Retention Period is the duration of time that a bucket retains data. InfluxDB drops points with timestamps older than their bucket's retention period. The minimum retention period is one hour [20]..

Retention Policy describes how long InfluxDB keeps data (duration), how many copies of the data to store in the cluster (replication factor), and the time range covered by shard groups (shard group duration) [18].

Rules are rules that are used for process automation. A rule can be activated in many ways, for example when an item changes state (plug goes from ON to OFF) or at a specific time expressed by a cron expression [1].

Series is a logical grouping of data defined by shared measurement, tag set, and field key [18].

Sitemap are a way to select and compose items in a user-oriented representation through user interfaces (UI), including the OpenHAB application for Android. That is, it is a UI for the user to interact with the items, being able to modify the status of the items or view data that has been set, such as weather, humidity, etc [34].

SMA Energy Meter also known as the Home Manager energy meter enables precise electrical metering for each phase conductor and in the form of heat balances, i.e. as a feed-in meter or current consumption meter [44].

SQL Structured query language (SQL) is a standard language for database creation and manipulation.

SSH or Secure Shell, is a remote administration protocol that allows users to control and modify their remote servers over the Internet through an authentication mechanism [17].

Tag is the key-value pair in the InfluxDB data structure that records metadata [18].

Telegram bot is an automated program that interacts with users on the Telegram messaging platform. Telegram bots can receive messages, send replies, perform tasks, and integrate with APIs or systems to automate workflows or provide services.

Thing are the entities that can be added to the system, they can be as many physical devices or sensors, as they can also represent a web service or any other manageable source of information and functionality [1].

Time series database is a software system that is optimized for storing and serving time series through associated pairs of time(s) and value(s).

Time Series Index uses the operating system's page cache to pull frequently accessed data into memory and keep infrequently accessed data on disk. [18] .

Time Structured Merge tree is the purpose-built data storage format for InfluxDB. TSM allows for greater compaction and higher write and read throughput [18] .

Tmate is a terminal multiplexer with instant terminal sharing: it enables a number of terminals to be created, accessed, and controlled from a single screen and be shared with another mates. tmate may be detached from a screen and continue running in the background, then later reattached, like as a daemon [18].

Virtual Private Network an arrangement whereby a secure, apparently private network is achieved using encryption over a public network, typically the internet.

volume A persistent storage mechanism in Docker that allows data to be stored and shared between containers or between a container and the host system, independent of the container lifecycle.

WiFi is a family of wireless network protocols, based on the IEEE 802.11 family of standards, which are commonly used for local area networking of devices and Internet access, allowing nearby digital devices to exchange data by radio waves [56].

Write Ahead Log is the temporary cache for recently written points. To reduce the frequency with which the permanent storage files are accessed, InfluxDB caches new points in the WAL until their total size or age triggers a flush to more permanent storage [18].

YAML is a human-readable data serialization standard that uses indentation and simple syntax to represent structured data like lists and dictionaries in a clean text format. It's language-agnostic and commonly used for configuration files and data exchange due to its readability.

Zigbee is an IEEE 802.15.4-based specification for a suite of high-level communication protocols used to create personal area networks with small, low-power digital radios, such as for home automation, medical device data collection, and other low-power low-bandwidth needs, designed for small scale projects which need wireless connection [57] .

Acronyms

AC Altern Current.

API Application Programming Interface.

DC Direct Current.

DIY Do It Yourself.

HTTP Hypertext Transfer Protocol.

IP Internet Protocol.

NAT NAT.

PaaS Plataform as a service.

PoC Proof of Concept.

RP Retention Policy.

TSDB Time Series Database.

TSI Time Series Index.

TSM Time Structured Merge tree.

UGR University of Granada.

VPN Virtual Private Network.

WAL Write Ahead Log.

Chapter 1

Introduction

This Bachelor Thesis presents the Final Degree Project for the Computer Engineering program, with a specialization in Software Engineering, carried out by the student Hicham Bouchemma Bouhou at the School of Engineering and Telecommunications in Granada. The main objective of this work is to demonstrate the knowledge and skills acquired throughout the degree, while also exploring and gaining experience in a completely new area. To achieve this, the project focuses on supervising the installation of a home automation system using OpenHAB.

This Final Degree Project has been developed in collaboration with the academic initiative GranaSAT, an aerospace research group at the University of Granada (UGR). GranaSAT is composed exclusively of students from various engineering disciplines, such as Aerospace, Electronics, Computer, and Telecommunications Engineering, who work together under the guidance of Dr. Andrés María Roldán Aranda.

Figure 1.1 – *Granasat logo*

The GranaSAT laboratory provided us with a dedicated workspace and all the necessary equipment and materials to carry out this project. The laboratory is located in I+D Josefa Castro Visozo building, located next to the lecture halls of the International Graduate School of Granada and the Clinical Hospital of Granada (Spain).

1.1 Motivation

The growing popularity and accessibility of Internet of Things devices have led an increasing number of people to adopt these technologies in their daily lives, as they help automate repetitive tasks and save time. When combined with software solutions, this technology becomes even more powerful, making this sector a valuable and promising area for commercial applications.

1

This type of project offers valuable learning opportunities across multiple disciplines. It involves concepts and practical skills from electronics, computer science, and telecommunications. Within computer science, it also touches on areas such as software development, databases, and system integration. This multidisciplinary approach promotes a deeper understanding of how different technological fields interact in real-world applications.

This work consisted of supervising the installation of a house automation system using a central Domotic system called OpenHAB, whose main mission is to unify all the information and functionalities of different ecosystems from different domotic companies and Internet of Things devices under a single system [28].

It is worth noting that this Final Degree Project can be considered a continuation of the project titled *"Home Automation Portal for Single-Family Homes Based on OpenHAB"*, previously developed by Francisco Javier Jiménez Lagaza. This work builds on its development, reusing and extending parts of the original project. This approach reflects a more realistic software development process, as most modern projects are rarely developed entirely from scratch.

1.2 Project objectives

The main objective of the project from the developer's perspective is to gain insight into the development process of this type of system and to acquire hands-on experience. A summary of the specific goals includes:

1. Understanding the structure and operation of home automation systems.
2. Learning how to configure and integrate various Internet of Things devices using OpenHAB.
3. Reuse and extend an existing OpenHAB-based automation platform to understand the software life cycle.
4. Gain experience in system supervision, troubleshooting, and real-world deployment.
5. Developing skills in software configuration, network communication, and user interface design.
6. Learning to integrate different components of the system, such as databases, API, and interfaces, into a cohesive and efficient application architecture.

1.3 Project structure

The structure of the project is as follows:

- Chapter 1: Introduction

  This is the current chapter, in which a short introduction, the objectives to be achieved, and the structure of the project have been shown.

- Chapter 2: Initial System Evaluation

  Section outlining the initial analysis of the existing system, including the devices in use, the house design, and the identified issues.

- Chapter 3: Project Planning & Requirements Analysis

  This section outlines the general planning and key requirements necessary to meet the objectives of the project. It also includes a brief software analysis in light of the newly defined goals.

- Chapter 4: Home Automation Design: Key Technologies and Devices

  Section that presents the new home automation design, along with the definition and explanation of the technologies, software, and devices used.

- Chapter 5: Implementation and configuration

  Section detailing installation, execution of sprint tasks, system configuration, performance analysis, and troubleshooting.

- Chapter 6: System Verification and Testing

  Section to show the validation tests done during the implementation to ensure the correct functioning of the OpenHAB system, as well as those of all devices and software installed.

- Chapter 7: Conclusions, Future Work and Lessons Learned

  Section expressing the opinion on the achievements of the project and discussing possible future improvements of the project.

Chapter 2

Initial System Evaluation

The purpose of this section is to review and analyze the current state of the existing home automation system, with a focus on its configuration, components, design, and functionalities.

The client expressed a desire to enhance the system and shared several ideas for improvements. During the initial interviews, the client provided an overview of the current home automation setup, including its network architecture, and highlighted the issues and dissatisfaction with the existing system. This feedback provided valuable information for the project.

The first step was to analyze the system based on the client's explanations to understand its current state, identify areas for improvement, and determine what can be fixed or improved.

First, the chapter will describe the devices currently in use, followed by an overview of the layout of the home. Finally, it will address the issues identified during system analysis. Much of this information is based on documentation from previous development work, which provides a detailed discussion of the devices and design. That documentation will be referenced where appropriate.

2.1 Actual Devices

This chapter presents a detailed overview of all devices currently integrated into the home automation system.

2.1.1 Domotic system

A **domotic system** is a home automation software designed to **centralize and integrate devices from different ecosystems** into a single platform. Using various **communication protocols**, allows for seamless onboarding of devices, eliminating the need for multiple applications to manage each one individually and providing unified control over all connected devices within the home. The system allows users to **automate tasks**, schedule routines, and customize the interface to fit individual preferences. In addition, it supports the **integration of DIY devices**, which can be connected via standard communication protocols, allowing them to be managed alongside commercial devices [28].

OpenHAB, a widely recognized open-source home automation system developed in Java, supports a wide array of technologies and brands due to its **modular architecture** based on "bindings" (plugins), which

enable communication with a wide range of devices. It can be installed on multiple platforms, including local servers, Docker or Raspberry Pi, offering **flexibility in data control**. **This domotic system was selected** for its large community, extensive device compatibility, thorough documentation, and high level of customization [1].

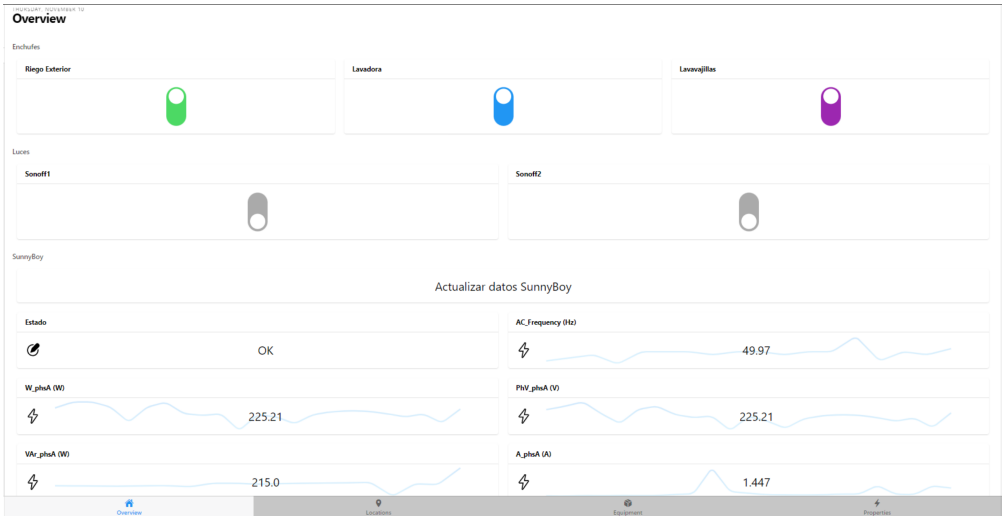


Figure 2.1 – *OpenHAB interface*

2.1.2 Virtual Assist

Virtual assistants are artificial intelligence systems that respond to voice commands. These systems continuously learn from user interaction and improve their functionality over time. Primarily used in smart home devices like speakers, they remain in standby mode, listening for **specific activation keywords** (e.g., "Alexa" for Amazon, "Ok Google" for Google, "Hey Siri" for Apple). Upon detecting the keyword, the system activates, **records the user's command**, and sends it to a cloud-based voice recognition system for processing. The system then **returns a comprehensible response or executes a command**, such as playing music or controlling connected devices. These assistants utilize intelligent algorithms to improve their understanding of natural language, providing more accurate and natural responses. [16] [30]

In this project, **Alexa was selected as the virtual assistant** to control most of the elements connected to OpenHAB. The choice was driven by Alexa's faster response times compared to competitors, its extensive customization options through exclusive Amazon Alexa "skills", and the wide range of routines it supports. Alexa routines can even be triggered by sounds like barking dogs or breaking glass [28].

Figure 2.2 – *Amazon Alexa*

2.1.3 Plugs

Smart plugs are versatile devices designed to control the power supply to appliances, offering a convenient alternative to manually plugging and unplugging devices. These plugs fit standard power sockets and allow users to remotely operate any connected appliance via a mobile device. Primarily connected through WiFi, some smart plugs also support Zigbee, although a compatible Zigbee hub is required for this protocol.

For this project, the client required **smart plugs to control various kitchen appliances**, including the **washing machine** and **dishwasher**. The selection of smart plugs was driven by the need not only to **automate device control** but also to **reduce electricity bills** and **improve time efficiency**. The chosen models were **Tapo P100** and **Amazon Smart Plug**, both of which are compatible with **Amazon Alexa**, allowing voice-controlled operation through the virtual assistant. **Tapo 110** model, which offers improved performance and additional features, was considered for future upgrades. Although it was not selected for this project, it presents a promising option for **future improvements** to better meet the customer's needs.

(a) *Amazon Smart Plug* **(b)** *Tapo P100*

Figure 2.3 – *Smart plugs*

2.1.4 Relays

Relays, or **intelligent switches**, are devices that **control the flow of electrical current**, allowing traditional switches or plugs to be upgraded with smart capabilities. These intelligent relays integrate seamlessly into **home automation systems**, providing enhanced control and functionality over standard electrical components. By using relays, users can **automate and manage** their existing electrical setups more efficiently.

Sonoff Zigbee relays were selected due to their reputation as a **leading brand in the market**. These relays **control the lighting** within the rooms, aligning with the **client's requirements**. To support these Zigbee relays, a Zigbee gateway/hub is used, specifically the **Tuya hub**, along with its **Smart Life application** interface. This setup helps manage the relays without overloading the home's WiFi network, ensuring a **stable and efficient operation**.[28]

(a) *Sonoff mini Zigbee* **(b)** *Tuya Hub Zigbee*

Figure 2.4 – *Relay and hub Zigbee*

2.1.5 Weather station

A **weather station** comprises a collection of instruments designed to **measure various meteorological parameters**, enabling the prediction of current and short-term weather conditions. Its primary function is to **gather sufficient data** to accurately forecast weather patterns, providing **valuable information on atmospheric conditions** [13].

For this project, the **Froggit WH3000 SE** weather station was selected. This model was chosen for its **excellent quality-to-price ratio**, making it a **cost-effective choice** while offering **reliable and accurate weather measurements** [28].

Figure 2.5 – *Froggit WH3000 SE*

2.1.6 Weather API

A **weather** API is a tool used to obtain **weather information** for a specific location. APIs facilitate interaction between systems by allowing **data retrieval** or the execution of functions. **Weather** APIs, specifically REST APIs, use HTTP requests to transfer a representation of the requested resource to the requester. These requests include **headers and parameters** that specify the desired information, such as **geolocation or coordinates**, to obtain accurate weather measurements [5, 50].

The **Openweathermap** API was selected. **OpenHAB** includes a binding that simplifies the integration of this API into the system. **Openweathermap** was chosen due to its **high number of daily calls allowed**, which ensures more frequent updates of weather data compared to its competitors [28].

Figure 2.6 – *OpenWeatherMap*

2.1.7 Solar Inverter

A **solar inverter** is a critical component in **solar energy systems**, responsible for converting the **direct current (DC)** generated by photovoltaic panels into **alternating current (AC)**, which is suitable for use in homes and businesses. It plays a central role in the overall **efficiency and performance** of the solar installation [52].

In this project, the **SMA Sunny Boy** hybrid solar inverter is utilized. Known for its **high efficiency** in energy conversion, it ensures optimal transformation of DC into AC while also providing **robust monitoring capabilities**. This selection included a **complete solar power kit** from **SMA**, solar panels, the SMA Sunny Boy solar inverter, SMA Energy Meter, and SMA Sunny Island batteries [28].

(a) *Sunny Boy* **(b)** *Sunny Island Battery*

Figure 2.7 – *SMA kit: Sunny Boy and Battery*

2.2 Actual House Design

To understand the **house design** and how the **home automation system** was intended to integrate, existing development documentation was reviewed, including the **house schema**. In addition, a **site visit** to the house was conducted to gain a practical understanding of the layout and validate the documented plans. This provided valuable information on the **spatial distribution** and the planned locations of the devices [28].

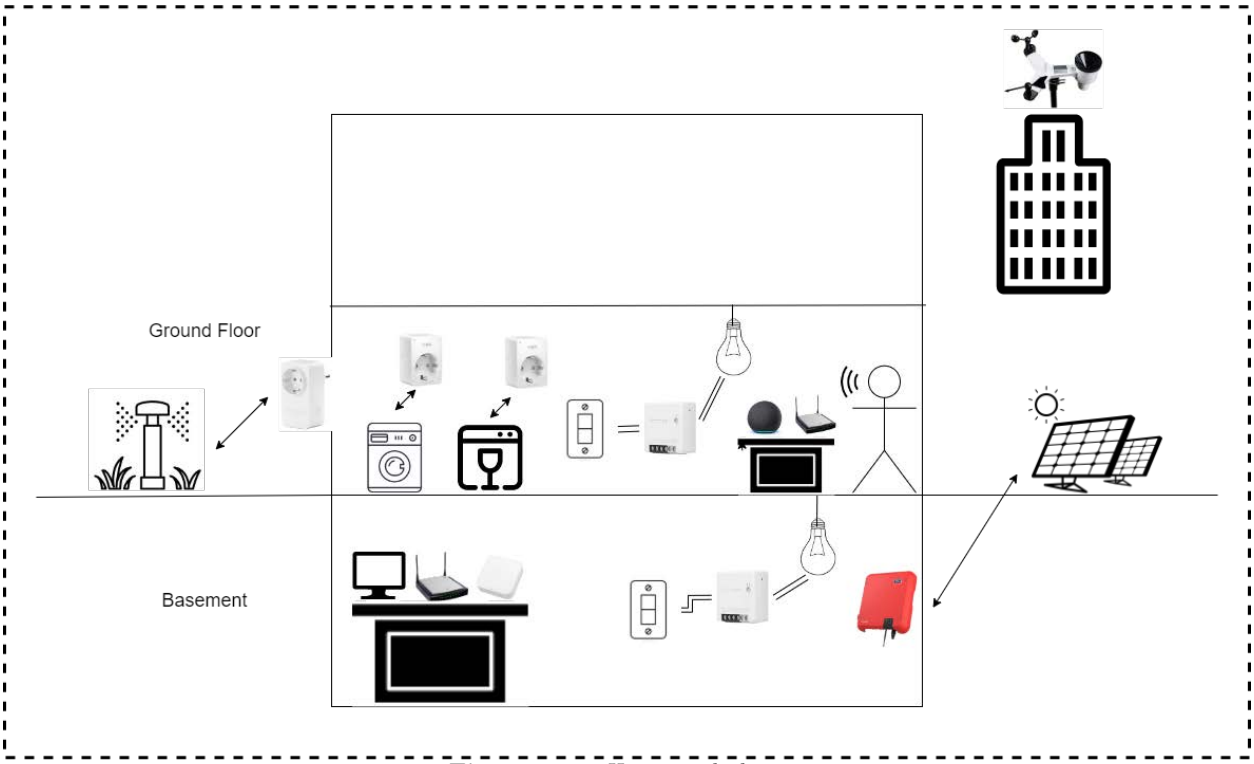


Figure 2.8 – *House with devices*

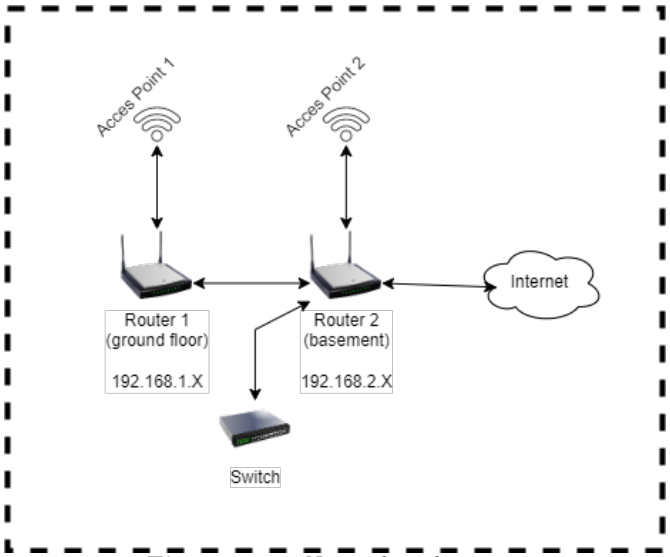


Figure 2.9 – *Network architecture*

2

As illustrated in the **diagrams**, the **house design** is clearly depicted. It is worth noting, and this will be explained in more detail in the **deployment section**, that OpenHAB is deployed inside a Docker container, while InfluxDB, the **time series database**, resides on the **host system**. A possible improvement would involve migrating the database to a Docker container, thus streamlining the orchestration of OpenHAB and InfluxDB.

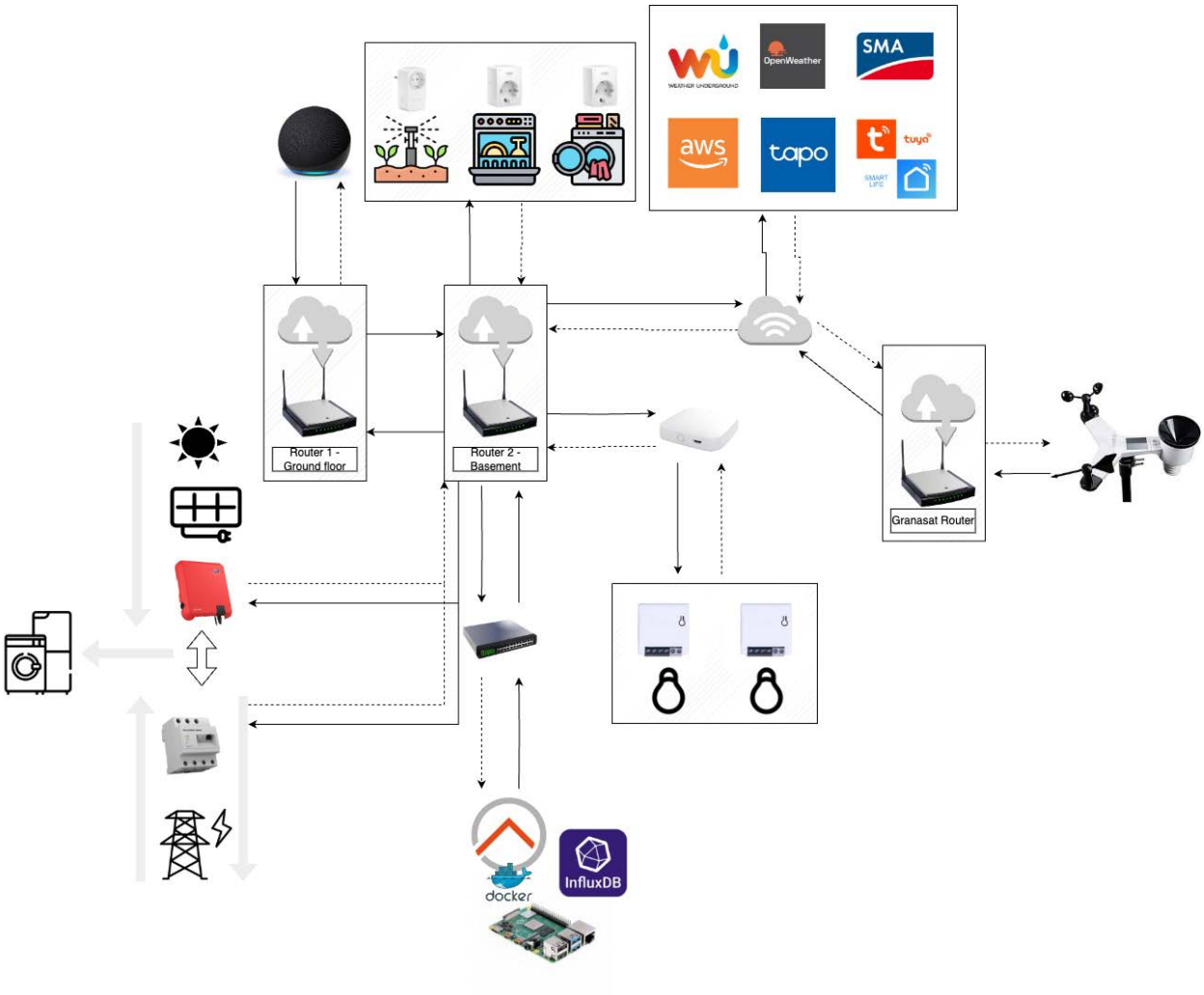


Figure 2.10 – *Home automation network architecture*

Subsequently, the **client's residence** was visited to verify the actual **device placements** and **network connections**. To confirm proper connectivity, **network traffic** was analyzed to ensure that each device's **IP address** responded to our requests. After performing several tests, it was concluded that all devices appeared to be connected correctly [28].

```
granasat@granasat:~ $ sudo arp-scan --localnet
Interface: eth0, type: EN10MB, MAC: dc:a6:32:66:6f:09, IPv4: 192.168.2.149
WARNING: Cannot open MAC/Vendor file ieee-oui.txt: Permission denied
WARNING: Cannot open MAC/Vendor file mac-vendor.txt: Permission denied
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
192.168.2.1     e0:19:54:bd:64:c7       (Unknown)
192.168.2.2     18:d6:c7:4e:8a:fd       (Unknown)
192.168.2.13    00:1b:9c:00:4a:88       (Unknown)
192.168.2.133   48:10:0b:23:03:db       (Unknown)
192.168.2.134   00:d0:93:53:da:8b       (Unknown)
```

Figure 2.11 – *Network traffic*

After verifying device connectivity, the next step was to test **connectivity within the home automation system's web interface**. To grant us access, the **client**, who is the actual **administrator**, registered our email address on MyOpenHAB org. MyOpenHAB is a **free cloud service** from OpenHAB Foundation that allows **remote access** to your home automation system from anywhere with an internet connection. By registering through **MyOpenHAB**, we gained access to the system not only through the **web interface** but also through the app available in Apple Store and Play Store. As this configuration is comprehensively detailed in section 4.2.4.1, it is not reiterated here to prevent redundancy [32] [28].

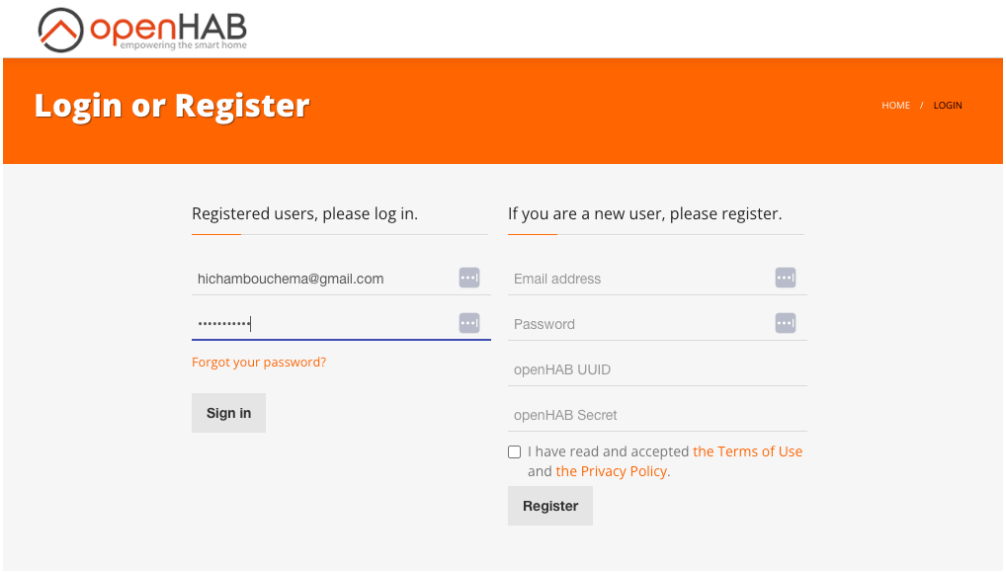


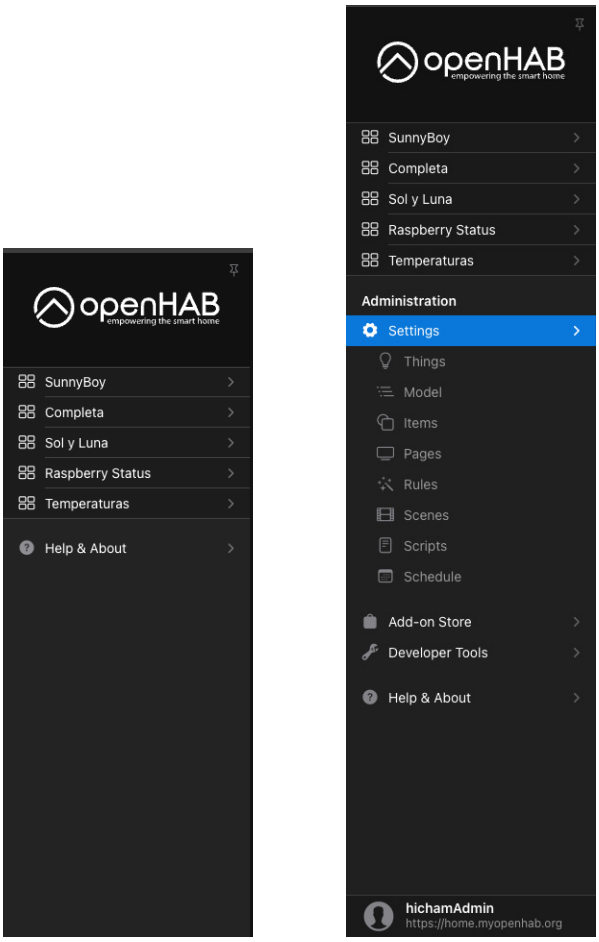
Figure 2.12 – *MyOpenHabOrg interface*

To see the devices connected to the system, we navigated to the **"Things" tab**. **"Things"** represent entities that can be integrated into the system. These can range from **physical devices** such as sensors to **virtual entities** such as web services, each providing a unique source of information or functionality [1, 34].

As **standard users**, we did not have access to the **"Things" tab**. To gain access to this tab and perform **administrative functions**, we needed to interact with the Docker container hosting the OpenHAB instance to create an **administrator user account** that would be used for future system management tasks.

```
openhab> users help
Unknown command 'help'
Usage: openhab:users list - lists all users
Usage: openhab:users add <userId> <password> <role> - adds a new user with the specified role
Usage: openhab:users remove <userId> - removes the given user
Usage: openhab:users changePassword <userId> <newPassword> - changes the password of a user
Usage: openhab:users listApiTokens - lists the API tokens for all users
Usage: openhab:users addApiToken <userId> <tokenName> <scope> - adds a new API token on behalf of the specified user for the specified scope
Usage: openhab:users rmApiToken <userId> <tokenName> - removes (revokes) the specified API token
Usage: openhab:users clearSessions <userId> - clear the refresh tokens associated with the user (will sign the user out of all sessions)
openhab> users add hichamAdmin 12345678 administrator
hichamAdmin (administrator)
User created.
openhab> users list
amroldan (administrator)
fran (administrator)
hichamAdmin (administrator)
```

Figure 2.13 – *Access to OpenHab container*



(a) *Standard User* **(b)** *Admin User*

Figure 2.14 – *Things tab*

At this stage, we observed that some **connections were not functioning as expected**. However, for those devices listed as **connected within the application**, we conducted tests to **verify their status**. These tests confirmed connectivity. The OpenHAB interface proved to be valuable in providing **real-time information** on the **connection status of individual devices**.

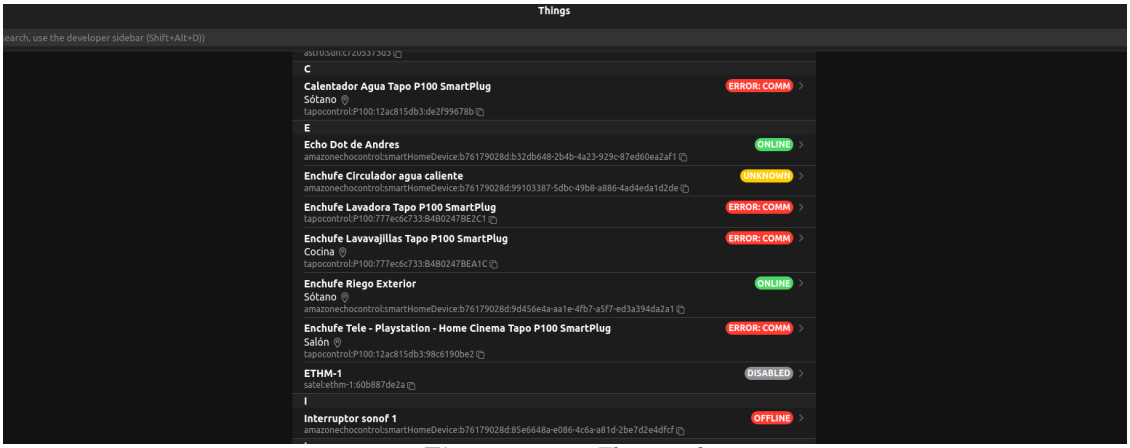


Figure 2.15 – *Things tab*

As mentioned previously, the client desired to **maintain the existing home automation system and network architecture** while incorporating initial modifications, such as **resolving device connectivity issues** and **migrating the database to a Docker container**. Recognizing the client's preference for preserving the existing infrastructure, our initial focus was on addressing critical issues, **identifying potential improvements**, and **conducting interviews to gather requirements**. These findings and proposed improvements will be detailed in subsequent sections.

Chapter 3

Project Planning & Requirements Analysis

This section will detail the strategy for improving the home automation system. The development team will work closely with the client, presenting the evaluation findings and working together to define specific requirements. To ensure minimal disruption to the client's daily routine, a phased implementation plan will be established allowing the system to remain operational while the necessary enhancements are made. The approach will prioritize tasks and focus on delivering value efficiently and effectively.

3.1 Requirement list

The list of requirements is paramount as it provides the basis for the improvement plan. A thorough understanding of these requirements, gathered through collaboration with the client and careful analysis, is essential to develop a comprehensive and effective strategy.

The following is a detailed list of requirements that will guide the improvement strategy.

Ref.	Main Objetives
MObj.1	Host and operate the system on a platform running Raspberry Pi OS
MObj.2	Install a headless version of Raspberry Pi OS to optimize system resources
MObj.3	Update OpenHAB to the latest version, including all security patches
MObj.4	Upgrade Docker to the latest stable version
MObj.5	Deploy the home automation system along with the required applications using Docker containers
MObj.6	Analyze and migrate the existing database to a Docker container alongside the home automation system
MObj.7	Integrate new smart plugs (4.1.2) for the garage door, water heater, underfloor heating, and television
MObj.8	Enhance the OpenHAB user interface in both the app and web platform
MObj.9	Create a page in the web interface and app to monitor and display the status of the Raspberry Pi
MObj.10	Create a page in the web interface and app to monitor and display data from the Sunny Boy inverter and Home Manager
MObj.11	Create a page in the web interface and app to display current temperature readings, weather forecasts, and sun and moon information
MObj.12	Troubleshoot and fix connectivity issues with Thing
MObj.13	Implement automatic weekly backups of the entire system
MObj.14	Implement a backup and cloning system for the Raspberry Pi SD card project folder to ensure full recovery in case of failure or data loss
MObj.15	Implement notification delivery through Telegram and WhatsApp of solar inverter status
MObj.16	Implement notification delivery via Telegram to report the water heater status, and enable interaction through a Telegram bot
MObj.17	Implement light automation to turn lights off at sunrise and on at sunset
MObj.18	Enable notification delivery via Telegram to report the status of the lights and allow interaction through a Telegram bot.
MObj.19	Implement irrigation automation to activate it daily between 10:30 and 11:30
MObj.20	Enable notification delivery via Telegram to report irrigation status and allow user interaction through a Telegram bot
MObj.21	Develop a system for analyzing OpenHAB logs
MObj.22	Document any issues that arise during the development of project objectives
MObj.23	Create comprehensive step-by-step documentation for installation and configuration of each device
MObj.24	Create a new home design aligned with the updated objectives, incorporating the latest technologies and newly integrated devices
MObj.25	Utilize GitHub for version control of project code and configurations
MObj.26	Use Tmate to enable secure remote access for development, due to the lack of a public IP address
MObj.27	Free up database space by purging outdated or unnecessary records and implement an automated process to delete obsolete data
MObj.29	Move light control via relays from the room to the porch/entrance

Table 3.1 – *List of main objectives of the project*

As this project is a continuation of previous work, only a limited number of additional functional requirements have been identified. However, before integrating the new requirements, it is necessary to update the system to ensure full compliance with those established in the earlier phase of the project [28]. Some of these previous requirements have been revised—certain ones have been updated to reflect current needs, while others have been deprecated due to changes in system scope or relevance.

Ref.	Description
FR.1	The system shall allow only authenticated and verified administrator users to modify system configurations
FR.2	The system shall store data collected from various devices in a database to ensure data persistence
FR.3	The system shall visualize collected data using graphical representations for improved readability and analysis
FR.4	The system shall support voice control via a virtual assistant, ensuring full compatibility with all integrated devices
FR.5	The system shall retrieve and display real-time data from solar energy devices, including the Sunny Boy solar inverter and the Home Manager energy meter
FR.6	The system shall collect and display weather data from a connected weather station
FR.7	The system shall retrieve and display meteorological data from an external weather API
FR.8	The system shall provide user-friendly interfaces for controlling connected devices
FR.9	The user shall have the ability to adjust the time intervals displayed in the graphical representations of collected data
FR.10	The user shall be able to operate the system remotely via a device (computer/mobile) outside the local network
FR.11	The user shall have the ability to activate and deactivate installed devices as needed
FR.12	The user shall be able to control and play music through the system

Table 3.2 – *List of previous functional requirements*

Ref.	Description
FR.1	The system shall display information from the Sunny Boy solar inverter and the Home Manager energy meter in a dedicated tab.
FR.2	The system shall display temperature readings, weather forecasts, and sun & moon information in a dedicated tab.
FR.3	The system shall present Raspberry Pi information in a dedicated tab.
FR.4	The user shall receive notifications about the solar inverter status via Telegram and WhatsApp.
FR.5	The user shall receive notifications about the outdoor irrigation status via Telegram.
FR.6	Lights shall automatically turn on at sunset and off at sunrise.
FR.7	The user shall receive notifications about the lights' status via Telegram.
FR.8	Upon receiving a notification from the lighting system, users shall be able to interact with a Telegram bot to query the current status or change the state.
FR.9	The user shall receive notifications about the water heater status via Telegram.
FR.10	Upon receiving a notification that the water heater is on, users shall be able to interact with a Telegram bot to query the current status or change the state.
FR.11	Irrigation shall automatically turn on at 10:30 and off at 11:30, and send a notification via the Telegram bot.
FR.12	Upon receiving a notification about irrigation being active, users shall be able to interact with a Telegram bot to query the current status, postpone the automation, or change the irrigation state.
FR.13	The system shall identify the last person who replied to the Telegram notification, in order to interact only with them.
FR.14	The system shall allow authenticated and verified administrator users to modify or disable automations related to lights, irrigation, and notifications.
FR.15	Users shall be able to access the system through the mobile app, with the same functionalities available as on the web interface.

Table 3.3 – *List of new functional requirements*

For the non-functional requirements, the previous set was reviewed and revised, with modifications and additions made to better align them with the objectives of the current project.

Ref.	Description
NFR.1	Software updates, including OpenHAB and Docker components, should be easily deployable without disrupting system functionality.
NFR.1	The web interface should load and display data within 5 seconds, even under high usage conditions.
NFR.2	The system should be designed to scale easily, supporting additional devices and integrations without requiring significant changes to the underlying infrastructure.
NFR.4	The Docker-based architecture must allow for easy addition of new applications or services in separate containers, ensuring the system can grow seamlessly.
NFR.6	The system must maintain 95% uptime, with minimal downtime for updates or maintenance.
NFR.7	Backups must be automatically verified to ensure data integrity and retrievability.
NFR.7	Backups should be stored in a secure, offsite location to prevent data loss in case of hardware failure.
NFR.8	The system shall display a message for any device or service error.
NFR.9	The user interface (both app and web) should be intuitive, with an easy-to-navigate design that requires minimal training for users.
NFR.10	Notifications and alerts should be clearly understandable, with sufficient information for the user to take appropriate actions.

Table 3.4 – *List of non-functional requirements*

3.2 Software analysis

Once the primary requirements of the customer are established, the software is analyzed and proof-of-concept (PoC) tests are performed to determine the most effective solutions that meet the updated customer needs.

3.2.1 Software for Automatic Backups

A backup system is defined as a process or software that creates copies of data to enable recovery in the event of data loss, corruption, or system failure. When designing an automatic backup solution, **two approaches** must be considered: one **executed directly by the operating system** and another **integrated within Docker containers**. Integrating backups within Docker allows the backup system and the containers to be backed up simultaneously, facilitating easier migration and avoiding the need to develop a backup system from scratch. Therefore, the strategy involves implementing one backup method at the OS level and another within Docker environments.

For OS-level backup, the selected approach involves using `cron` for scheduling a Bash script for automation, and Rclone - a command line program for manage and synchronize files to various cloud storage providers due to its reliability and flexibility [39].

Regarding Docker compatible backup technologies, a study was conducted to compare the most popular solutions:

- Duplicati: An open-source backup software featuring strong encryption and incremental backups. It supports a wide range of cloud storage providers and runs effectively as a Docker container, making it suitable for deployment alongside home automation systems.
- BorgBackup: A secure, deduplicating backup tool with encryption capabilities. BorgBackup can be deployed using Docker images, enabling seamless integration with home automation services to provide consistent and reliable backups.
- Restic: A modern backup solution supporting encryption and multiple cloud backends. Official Docker images are available, allowing containerized deployment for straightforward management in Dockerized environments.
- Timeshift: A Linux system snapshot tool primarily designed to operate on the host OS. It is not intended for deployment within Docker containers and is therefore unsuitable for backup strategies involving containerized applications, but can be installed in a Linux container.

A comparative table will be made between all of them, to see which one is better:

Features	**Duplicati**	**BorgBackup**	**Restic**	**Timeshift**
Encryption	Yes	Yes	Yes	No
Incremental backups	Yes	Yes	Yes	Snapshot-based
Cloud storage support	Yes	Partial	Yes	No
Automation (scheduling)	Yes	Yes	Yes	Yes
User-friendly UI	Yes (web UI)	No (CLI)	No (CLI)	Yes (GUI)
Docker deployment	Yes	Yes	Yes	No (with OS only)
Community usage	Medium	Large	Growing	Medium
Use case	Encrypted scheduled backups, easy deployment alongside home automation	Secure, deduplicated, encrypted backups, CLI-focused	Modern, multi-backend encrypted backups, CLI-focused	System snapshots and rollback for Linux desktops
Election	 (1º)	 (2º)	 (3º)	 (4º)

Table 3.5 – *Comparison of Backup Solutions with Docker Deployment*

After evaluating the different options, **Duplicati** was selected due to its support for **Docker deployment**, built-in **encryption**, and compatibility with multiple **cloud storage providers**. Furthermore, it offers a **user-friendly web interface**, which improves ease of use and management [12].

Figure 3.1 – *Duplicati logo*

3.2.2 Persistence Databases for OpenHAB

In the context of OpenHAB, **persistence** refers to the ability to **store the historical state of items** (such as sensor readings, switch states, and other values) over time. This allows users to **visualize data trends**, **restore item states after restarts**, and **enable advanced automation** based on historical values [33].

Several persistence services are available for OpenHAB, each with different features, performance characteristics, and deployment flexibility — particularly in Docker-based environments.

The current persistence database in use is InfluxDB. However, due to an upcoming migration, it is necessary to evaluate whether continuing with InfluxDB remains the optimal choice or if switching to an alternative database would be more advantageous [19].

The following table compares some of the most commonly used persistence solutions for OpenHAB:

- InfluxDB: A time-series database optimized for storing timestamped data. It integrates seamlessly with OpenHAB via the InfluxDB persistence Add-on and offers strong Docker support, scalability, and compatibility.
- MongoDB: A NoSQL document-oriented database that can be used with OpenHAB persistence through the community Add-on or custom bindings. It offers flexible schema design and scalability, but is less common for time-series data.
- PostgreSQL: A powerful relational database system compatible with JDBC persistence in OpenHAB. It is open-source, highly reliable and suitable for complex queries and data integrity.
- RRD4J: A lightweight round-robin database designed for low-resource environments. It is included by default in OpenHAB and does not require external services or Docker containers.

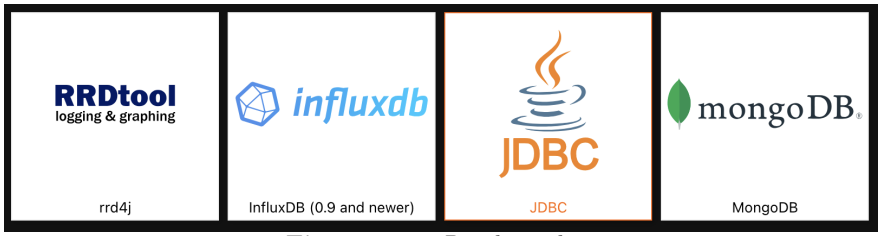


Figure 3.2 – *Databases logo*

Features	InfluxDB	MongoDB	PostgreSQL	RRD4J
Time-series optimized	Yes	No	No	Yes
Docker support	Yes	Yes	Yes	Not needed
Resource usage	Medium	Medium	Medium-High	Low
Setup complexity	Moderate	Moderate	Moderate	Very low
Integration in OpenHAB	Excellent (addon available)	Community addons / custom bindings	Via JDBC binding	Native
Best use case	IoT time-series with visualizations	Flexible document data storage	Complex queries and reporting	Lightweight local logging
Election	✓ (1º)	✗ (3º)	✗ (4º)	✗ (2º)

Table 3.6 – *Comparison of Persistence Databases for OpenHAB*

After evaluating the available **persistence options**, **InfluxDB** was selected due to its **native support for time-series data** — the primary format used in **OpenHAB** — its **reliable Docker deployment**, and its **seamless integration with the OpenHAB platform**.

Figure 3.3 – *InfluxDB logo*

3.3 Project Planning

Following the requirement gathering phase, the scope of the project has been clearly defined, enabling effective planning. An agile development methodology will be used, structured around three-week sprints. At the end of each sprint, progress will be presented to the client for feedback and evaluation.

The overall development process is capped at a maximum of five sprints, promoting iterative refinement and adaptability. Specific tasks within each sprint will not be predetermined, as the complexity and duration of each task can vary significantly. Instead, the team will adhere to a high-level execution plan. The completed items from this plan will be reviewed at the end of each sprint to facilitate retrospectives and guide the next iteration.

Execution Plan:

1. **Database Migration:** Existing data will be analyzed and migrated to a containerized database environment. The structure and content of the database will be thoroughly documented to facilitate understanding and ensure smooth maintenance for future developers.
2. **System Preparation:** The setup will begin with the installation of a headless version of the Raspberry Pi OS. This step also involves updating essential software components such as Docker, Docker Compose, and OpenHAB to establish a stable and optimized foundation for future development activities.
3. **Containerization and Deployment:** The home automation system, including the database and auxiliary services, will be deployed using Docker containers. This strategy ensures modularity, isolation, and scalability, aligning with the project's non-functional requirements.
4. **Device Integration:** Smart devices—such as plugs, will be integrated. Existing devices requiring maintenance will be inspected and repaired as needed. Each new integration will be followed by tests to confirm compatibility within the OpenHAB ecosystem.
5. **Remote Access and Notifications:** Remote access capabilities will be configured using Tmate. Furthermore, real-time notification systems will be implemented through platforms such as Telegram and WhatsApp to improve accessibility and user engagement.
6. **User Interface Enhancements:** Enhancements will be applied to both the web and mobile user interfaces to provide clear visualizations of system status, device information, and energy metrics. Additional interface components will be developed to monitor the Raspberry Pi and solar energy system. All changes will prioritize usability and simplified design.
7. **Automation and Control:** Automation features will include periodic system backups and recovery procedures. Weekly backups will be scheduled to protect the integrity of the system and reduce downtime risks.
8. **Documentation and Version Control:** Detailed documentation covering system configurations, procedures, and encountered issues will be maintained throughout the project. Version control will be handled using GitHub to ensure traceability and collaborative development.

It is important to note that this **execution plan** serves primarily as a **conceptual framework**. Although it outlines the **key goals** of the project, the **actual order of execution** may vary in practice due to **technical constraints** or **evolving project needs**. However, the plan provides a **structured foundation** to guide development efforts.

Based on this execution plan, the following diagram illustrates the project's implementation flow:

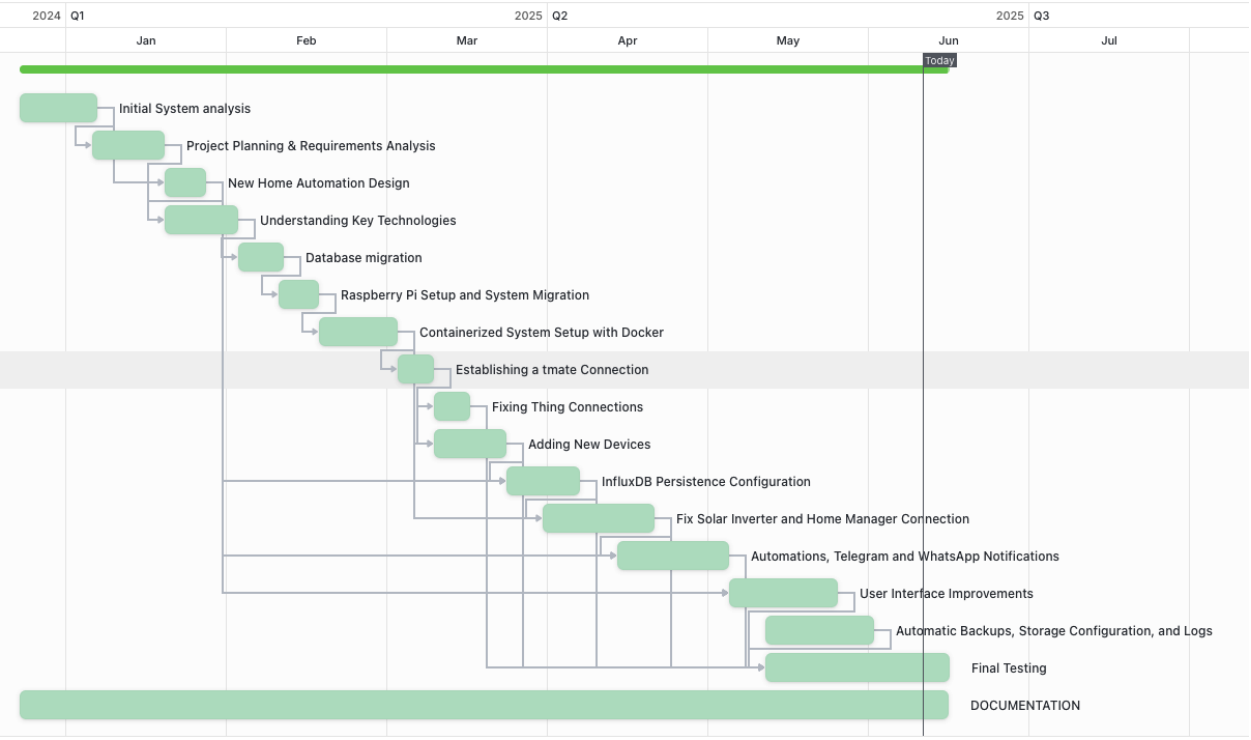


Figure 3.4 – *Project execution plan*

3.4 Project budget

This section presents an analysis of the costs involved in the project, covering both labor and materials. This analysis enables an accurate estimation of the total financial investment required for the project.

To provide a clear and organized overview, the cost analysis is divided into three main categories:

- **Hardware costs:** Physical components and materials used throughout the project.
- **Software costs:** Services, licenses, or applications necessary for implementation.
- **Labour costs:** Human resources and time allocated to the development and execution phases.

3.4.1 Hardware Resources

This section outlines the hardware resources used specifically for the current project. Equipment or materials from previous projects are not included in this analysis.

Item	Quantity	Total Cost [€]
Personal Laptop	1	900.00
Amazon Smart Plug	2	14.99
TP-Link Tapo P100	2	36.04
Raspberry Pi 5	1	80.00
Total Hardware Cost		**1,082.06**

Table 3.7 – *Hardware resources used in the project*

3.4.2 Software Resources

This section presents the software tools and platforms used throughout the development of the project. All of the software listed below was used under free licenses or free-tier plans, which resulted in no additional cost to the overall project budget.

Software	Total Cost [€]
Visual Studio Code	0.00
GitLab	0.00
Docker	0.00
Duplicati	0.00
Draw.io	0.00
ClickUp	0.00
Telegram	0.00
OpenHAB	0.00
OpenHAB Mobile App	0.00
Tmate	0.00
Amazon Alexa App	0.00
Tapo App	0.00
OpenWeatherMap API	0.00
OpenHAB Cloud	0.00
InfluxDB v2	0.00
Total Software Cost	**0.00**

Table 3.8 – *Software resources used in the project*

3.4.3 Human Resources

Labor costs were estimated based on salary data obtained from websites such as Glassdoor and Indeed, which aggregate information submitted by current and former employees, as well as employers. The average hourly rate for a junior software engineer is approximately €15.00. For this project, an estimate of 500 hours of labor will be invested.

Labor Costs	Total cost [€]
Junior Software Engineer	7,500.00
Total Labor Costs	7,500.00

Table 3.9 – *Project human resource costs*

3.4.4 Overall Project Budget

After calculating the costs for each individual category, we add these amounts to determine the total estimated cost of the project.

Project Costs	Total Budget [€]
Physical resources	1,082.06
Software resources	0,00
Human resources	7,500.00
TOTAL BUDGET	8,582.06

Table 3.10 – *Project total budget*

Chapter 4

Home Automation Design: Key Technologies and Devices

This section introduces the new home automation design and details how the main technologies, software, and devices will be integrated to meet the project objectives. The key technologies will also be briefly explained to demonstrate their purpose, functionality, and how to maximize their benefits.

4.1 House Design

The new design of the home will **remain largely unchanged**; however, the **web and the app interfaces** will be enhanced to align with the updated design. These improvements will include the **integration of controls for new devices** and **new software**.

Each component of the **house design** will be explained in this section, while the **server-side elements** will be addressed separately in the next section. These **server technologies** are considered essential from a development point of view because of their relevance and impact on the overall system.

4.1.1 Virtual Assistant

The primary function of the selected virtual assistant, **Amazon Alexa**, is to assist the customer in simplifying the use of devices connected to OpenHAB [28].

The **Echo Dot with Alexa** is connected to Router 1 as shown in Figure 4.1, located on the ground floor, while OpenHAB is connected to Router 2, located in the basement. Any information transmitted by Alexa must pass through both routers before reaching the Internet, where it is processed by **Amazon's servers**, and the corresponding command is executed. OpenHAB then receives this information through Amazon's binding, ensuring that the relevant items in the OpenHAB system are updated, particularly when the device app is used instead of the OpenHAB app.

4.1.2 Smart Plugs

Two types of smart plugs are utilized, as detailed in 2.1.3, to enable the activation and deactivation of devices as required.

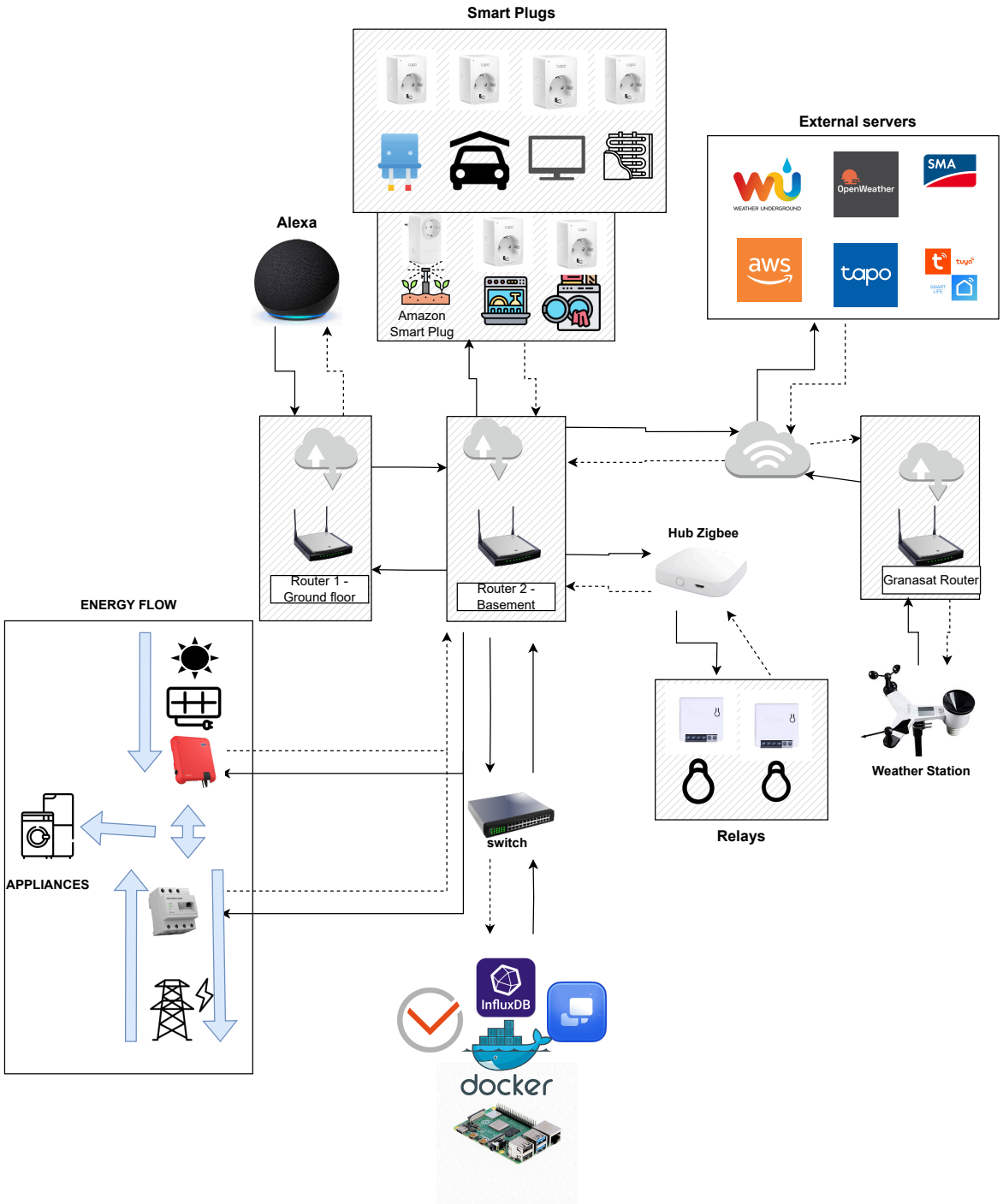


Figure 4.1 – *Home automation network architecture*

4.1.2.1 Amazon Smart Plug

This plug is integrated into the **Amazon ecosystem** and is managed through the **Alexa app**. Its primary function is to control the **outdoor irrigation system** of the home.

The plug is connected to Router 2 (Figure 4.1) and communicates with Alexa via WiFi. When a voice command is issued to activate the plug, Alexa transmits the request to **Amazon's servers** through the network routers. Amazon then processes the request and responds with the appropriate action. Since this plug operates through Alexa, OpenHAB receives the corresponding information through the **Amazon binding** of the system.

4.1.2.2 Tp-Link Tapo P100

These **TP-Link Tapo smart plugs** operate within their own **ecosystem**, utilizing dedicated servers and a proprietary app. However, they can be seamlessly integrated with Alexa via **Amazon Skills**, enabling **voice-controlled operation**.

These plugs are designated for the management of **household appliances** that appear in the house design. As shown in the system diagram, the Tapo plugs are connected to Router 2 (Figure 4.1), which facilitates communication with **Tapo's servers**. These servers, in turn, interact with **Amazon's infrastructure**, allowing Alexa to retrieve the plug status and execute commands to modify their state.

4.1.3 Relays

The **Smart Relays** were previously responsible for controlling the lighting within a room. They are now used to control the lighting in the porch and are installed behind traditional switches. Unlike standard WiFi-based devices, these relays communicate using the **Zigbee** protocol. As they rely on Zigbee, they must be connected to a **Zigbee hub**, which acts as the central controller for their operation.

These relays are fully compatible with **Alexa** and operate via the Zigbee protocol, connecting to a dedicated hub. This hub, in turn, is linked to **Router 2** (Figure 4.1), enabling communication with **Smart Life servers**. Furthermore, when the Sonoff Mini relay is manually switched using the traditional switch, since it is installed behind it, the hub detects the status change and relays this information to the servers in real time.

4.1.4 Weather API

The **Weather** API retrieves data from the OpenWeatherMap API, a **weather forecasting service**. It uses specified **coordinates** to return the most relevant weather data for that location, which is accessed using an OpenWeatherMap API key and an OpenWeatherMap account.

OpenHAB obtains weather information by making **requests** (Figure 4.1) through the designated **binding**, utilizing both the **coordinates** and the API key associated with OpenWeatherMap.

4.1.5 Weather Station

The weather station is located on the roof of the GranaSAT laboratory building.

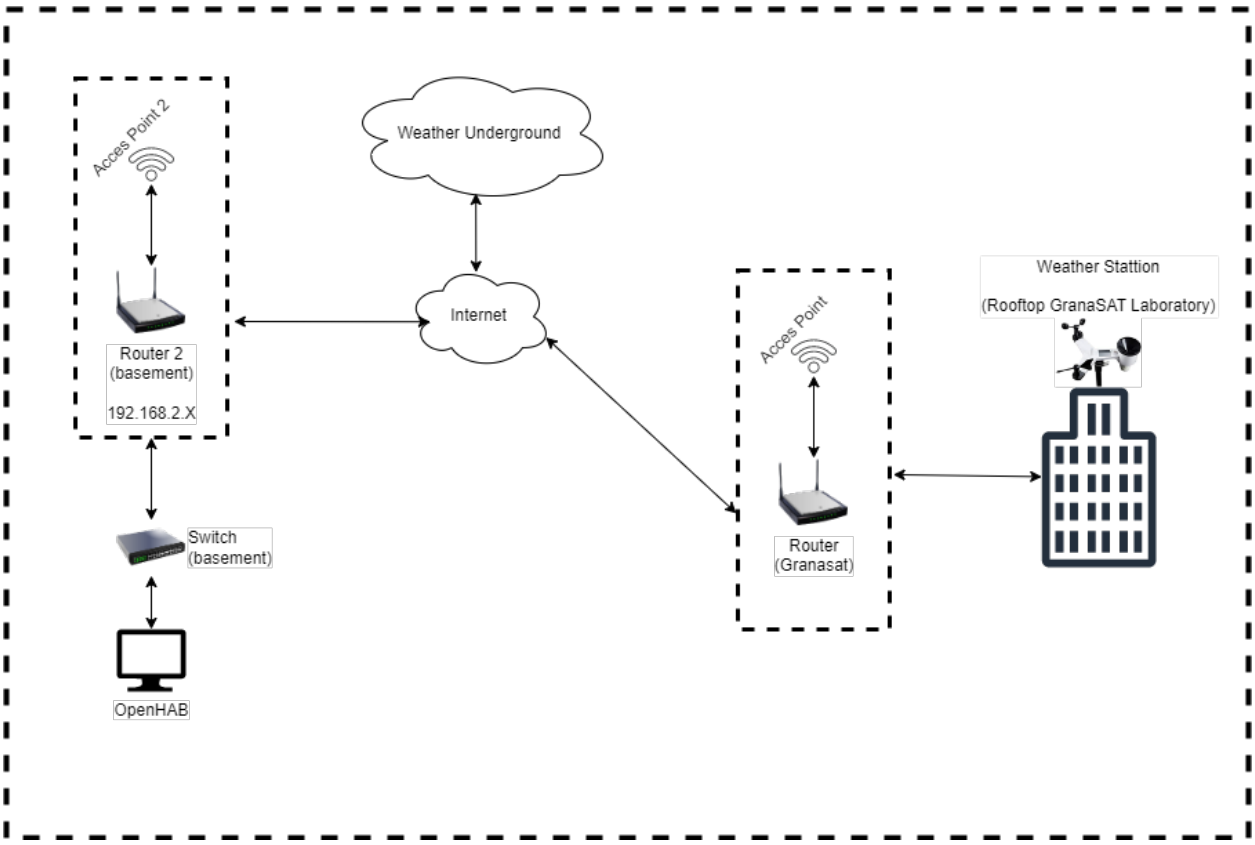


Figure 4.2 – *Weather station design*

The **weather station** is installed on the roof of the GranaSAT laboratory building. As shown in the image, it connects to the GranaSAT router via WiFi. To integrate its data into the OpenHAB system, the station must first **publish the information on a web platform**. The data is then accessed through its API via WeatherUnderground, allowing OpenHAB to **retrieve and process the weather information**.

4.1.6 Solar Inverter

The **solar inverter** is responsible for **converting the energy generated by the solar panels into alternating current (AC)**, making it useful for household devices. It is installed in the **basement**, where it is directly connected to the solar panels. **SMA provides a detailed diagram** that illustrates its functionality and various connections.

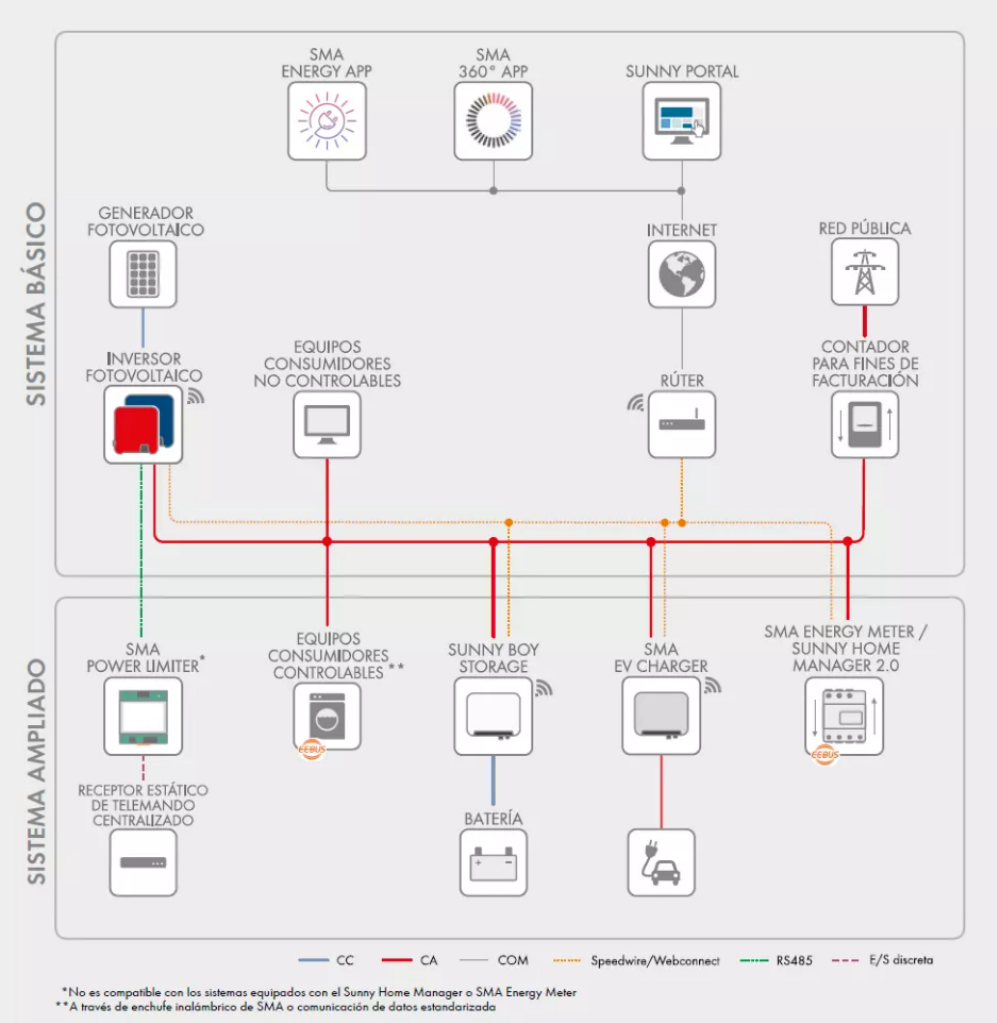


Figure 4.3 – *How the solar system works [45]*

The system is divided into two sections: the **basic part** and the **optional part**. In the basic part, the **essential electrical connections** are established. The electricity in the home is mainly derived from the **energy generated by the solar panels**. However, it can also draw power from the **public grid** when necessary. This is why the system includes an **electrical meter** to monitor and measure the energy consumption of both sources.

In addition to the basic installation, the customer has an SMA Energy Meter to **monitor and regulate energy injection**. Both the inverter and the energy meter are connected to **Router 2** (Figure 4.1). **SMA offers access to several online portals** where data from installed devices, including the inverter and the energy meter, can be accessed. These portals provide critical metrics such as **solar panel generation**, **estimated solar radiation**, and **household energy consumption**. To integrate these data into the OpenHAB system, **Python scripts** are used to automate the retrieval and processing of this information.

4.2 Key technologies & home integration

As mentioned above, this section details the main technologies and software components of the home automation system, explaining their functionality, integration, and suitability for this project.

4.2.1 Raspberry Pi

Previously, a **standard computer** running **Ubuntu** was used as the host server, but now a Raspberry Pi will be used in the project. The objective is to **migrate the system** to the Raspberry Pi after installing a **headless operating system**, specifically chosen to **optimize storage usage** and eliminate the need for a graphical interface. This selection is well-founded, as the Raspberry Pi is widely recognized for its suitability in **home automation systems** due to the following key advantages [40]:

1. **Cost-effectiveness**: Raspberry Pi boards are relatively inexpensive, providing an affordable entry point for building home automation systems.
2. **Connectivity**: Raspberry Pi boards are equipped with multiple connectivity options, including Wi-Fi, Ethernet, and USB ports.
3. **Low power consumption**: Designed for energy efficiency, Raspberry Pi boards are well suited for applications that require continuous operation. Their low energy consumption contributes to reduced electricity costs and a smaller environmental footprint.
4. **Community support**: The Raspberry Pi benefits from an extensive and active community of users and developers. This ensures the availability of abundant online resources.

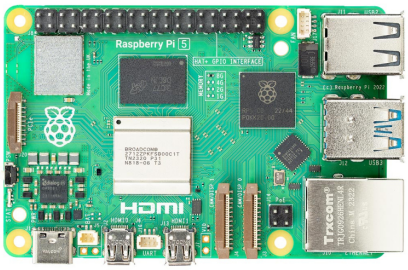


Figure 4.4 – *Raspberry Pi 5*

4.2.2 Docker

Docker is an **open source** set of platforms as a service **Platform as a service** (PaaS) products that use **OS-level virtualization** to deliver software in packages called **containers (virtualization)**. The service has **free and premium levels**. The software that hosts the containers is called **Docker Engine**. In essence, Docker can be described as a **platform that provides and manages containers for application deployment** [3].

Before explaining what a container is, the key architecture used to run the home automation system, it is important to first understand what a **Docker image** is and how to design it for **optimal performance** [2].

A **Docker image** serves as a **base template** for creating containers that can be used to run applications or generate new images. A Docker image is a **snapshot or blueprint** of the libraries and dependencies required inside a **container** for an application to run. These images can range from executable to operating system images, applications, and more.

Docker Images can be downloaded from registries or repositories, such as **Docker Hub** or Docker Store, or created manually using a Dockerfile. In our home automation system, we use both approaches. When creating a custom image, it is advisable to keep it as **simple as possible** to optimize performance [2].

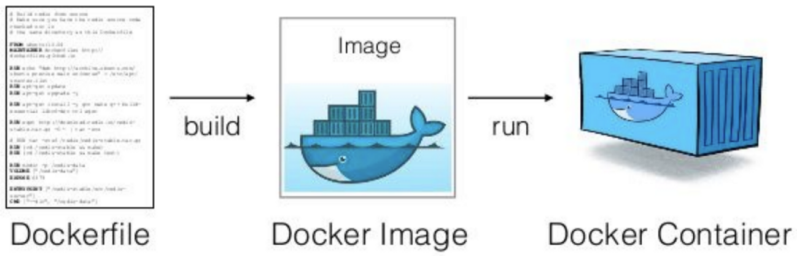


Figure 4.5 – *Docker file, image and container*

The Dockerfile is essentially a **set of instructions** that is used to build a Docker image. A Dockerfile can be either an official version or one created by the user. In our case, we created our own Dockerfile. This file is used by the **docker build** command to generate the image [4].

In a Dockerfile, each command creates a **separate layer**, even if the same command is used multiple times. To optimize performance, it is essential to **minimize the number of layers** [8].

Docker utilizes a **caching mechanism** to accelerate the image-building process. However, this caching can sometimes lead to issues where updates are not installed as expected. This occurs because Docker may **reuse cached layers** instead of fetching the latest versions of packages or dependencies [7].

Once the concept of Docker image is understood, it becomes possible to define a container. A **container** can be described as a **Docker image in execution**. It represents a lightweight and portable software package that includes all the necessary dependencies to run a specific application efficiently and consistently across different environments.

A container enables to **package an application** along with its required libraries and dependencies, ensuring that it runs consistently across different environments. Once a container is running, the application cannot access resources that were not explicitly provided. In addition, each container has **allocated resources**, such as CPU, memory, and storage, which are restricted and managed to ensure efficient performance [2].

4.2.3 Docker Compose

Docker Compose is a powerful tool for defining and running **multi-container** Docker applications. It simplifies the management of complex environments that involve multiple interconnected services, such as databases, web servers, and Internet of Things applications [6].

Using a single YAML configuration file, **Docker Compose** allows developers to **define and launch** multiple containers with a single command. This file specifies the dependencies of services, networking, and resource allocation, ensuring seamless **orchestration of the system**. Any modifications to the setup can be made directly within the **configuration file**, making it easy to update and maintain the application [10].

In this context, **orchestration** refers to the process of **managing multiple containers** as a single cohesive system, ensuring they function together efficiently and reliably.

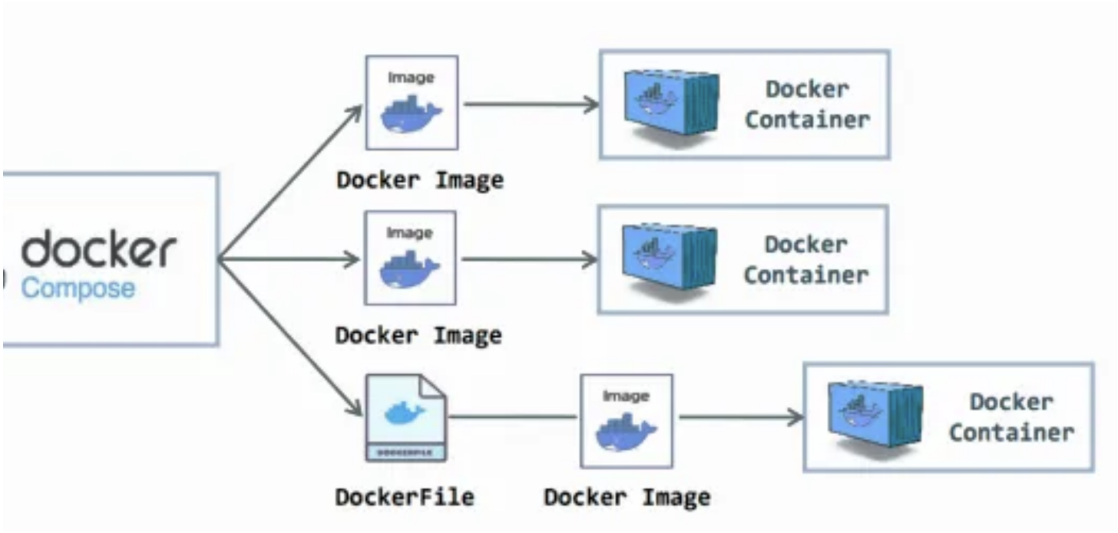


Figure 4.6 – *Docker Compose*

4.2.4 OpenHAB

OpenHAB is the **central element** of this project, serving as the platform where all information and control functions from various devices and systems are unified. It enables centralized management and integration of the entire home automation environment.

Figure 4.7 – *Openhab logo*

As a starting point, this section introduces the fundamental principles of how OpenHAB operates [1, 34, 28].

- **Things**: Entities that can be added to the system; they can be **physical devices** or sensors, as they can also represent a **Web service** or any other manageable source of information and functionality.
- **Bindings**: **Software adapters**, which enable communication between Things and your home automation system.
- **Channels**: **Interfaces exposed by Things**. A single **Thing** can have multiple **Channels**, each representing a particular function or data point. The functionalities are defined by the binding.
- **Items**: Logical representations of individual **data points** or **control elements** in the system. **Items** are linked to **Channels** to visualize or manipulate the state of the device.
- **Links**: **Associations** between **Items** and **Channels**. They allow data flow between the logical (Items) and physical (Things) layers.
- **Rules**: **Automation** that defines system behavior based on specific conditions or triggers, for example, when an item changes state or at a specific time expressed by a cron expression.
- **Persistence**: **Mechanism to store historical data** from **Items**, enabling charting, analysis, or restoring previous states.
- **Sitemaps**: Define how **Items** are displayed in the user interface. They offer a customizable structure for users to interact with their smart home setup. Not used in this project.
- **Pages**: **New user interface** released in OpenHAB version 3, to interact with **Items**.

Things have an associated **status** that indicates their current state and helps detect potential issues:

Status	Description
UNINITIALIZED	This is the **initial status** of a Thing when it is added or the framework is being started. This status is also assigned if the initializing process failed or the binding is not available. Commands sent to Channels will not be processed.
INITIALIZING	This status is assigned while the **binding initializes the Thing**. It depends on the binding how long the initializing process takes. Commands sent to Channels will not be processed.
UNKNOWN	The handler is **fully initialized** but due to the nature of the represented device/service it cannot really tell yet whether the Thing is ONLINE or OFFLINE. Therefore the Thing potentially might **be working correctly** already and may or may not process commands.
ONLINE	The device/service represented by a Thing is assumed to be **working correctly** and can **process commands**.
OFFLINE	The device/service represented by a Thing is assumed to be **not working** correctly and may not process commands. But the framework is allowed to send commands, because some radio-based devices may go back to ONLINE if a command is sent to them.
REMOVING	The device/service represented by a Thing **should be removed**, but the binding has not confirmed the deletion yet. Some bindings need to communicate with the device to unpair it from the system.
REMOVED	This status indicates that the device/service represented by a Thing **was removed** from the external system after the REMOVING was initiated by the framework.

Table 4.1 – *States of things [31]*

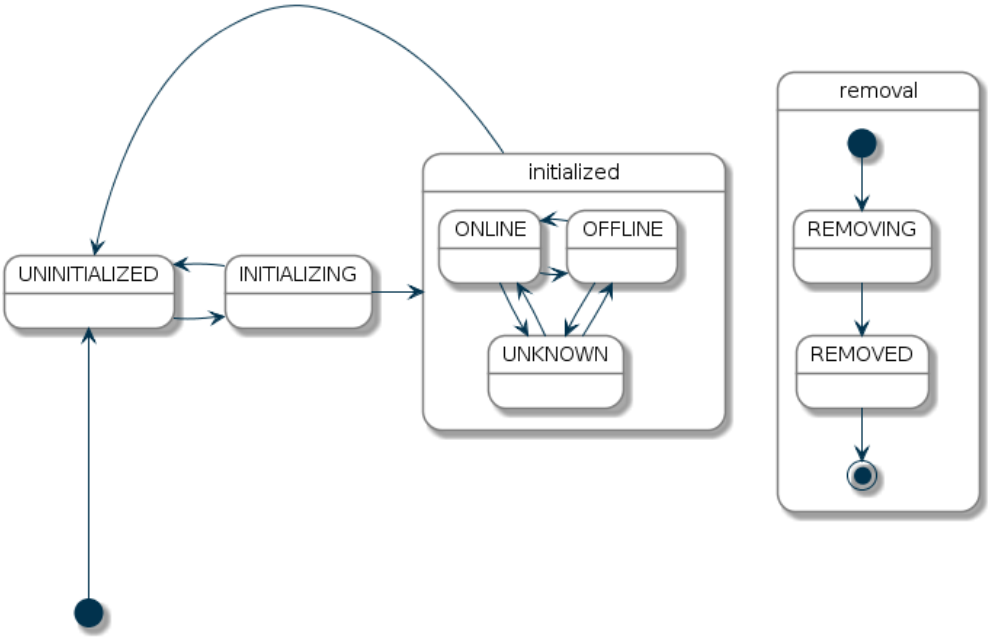


Figure 4.8 – *State diagram of the things [35]*

As mentioned in section 2.2, we use the OpenHAB REST API through MyOpenHAB.org, a service provided by OpenHAB Cloud. This solution enables **secure remote access** to local OpenHAB instances without exposing ports to the Internet or configure a complex VPN, ensuring a safe and reliable connection.

4.2.4.1 OpenHAB Cloud Configuration and Mobile App Integration

OpenHAB Cloud enables users to **link their local OpenHAB installation to a remote cloud service**, such as `myopenHAB.org`, which is a publicly hosted instance maintained by the **OpenHAB Foundation**. This setup allows for **remote access to the OpenHAB server** without requiring a connection to the local network. As a result, users can interact with their smart home system through the **OpenHAB mobile app** or the **MainUI interface from any external device** [32, 14].

The configuration of OpenHAB Cloud is managed through a specific **binding**, which is thoroughly documented in **Section 5.2** of the previous project [28]. Since no modifications were required from the development team and only a basic understanding of the setup was needed, this section will focus solely on the user registration process and the steps necessary to enable remote access via MyOpenHAB.org [14].

The first step involves creating an account on MyOpenHAB.org. Before doing so, it is necessary to retrieve two identification values from the local server: the **UUID**, located in `openhab_userdata/uuid`, and the **Secret**, found in `openhab_userdata/openhabcloud/secret`. These values allow OpenHAB Cloud to recognize the server during account creation [32].

```
/openHAB$ cd openhab_userdata/
/openHAB/openhab_userdata$ cat uuid
```

Figure 4.9 – *UUID location*

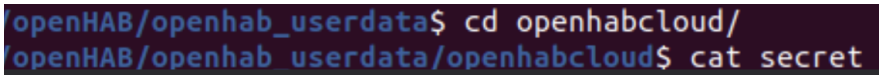


Figure 4.10 – *Secret location*

Once both values are obtained, the user can proceed to MyOpenHAB.org to complete the registration process:

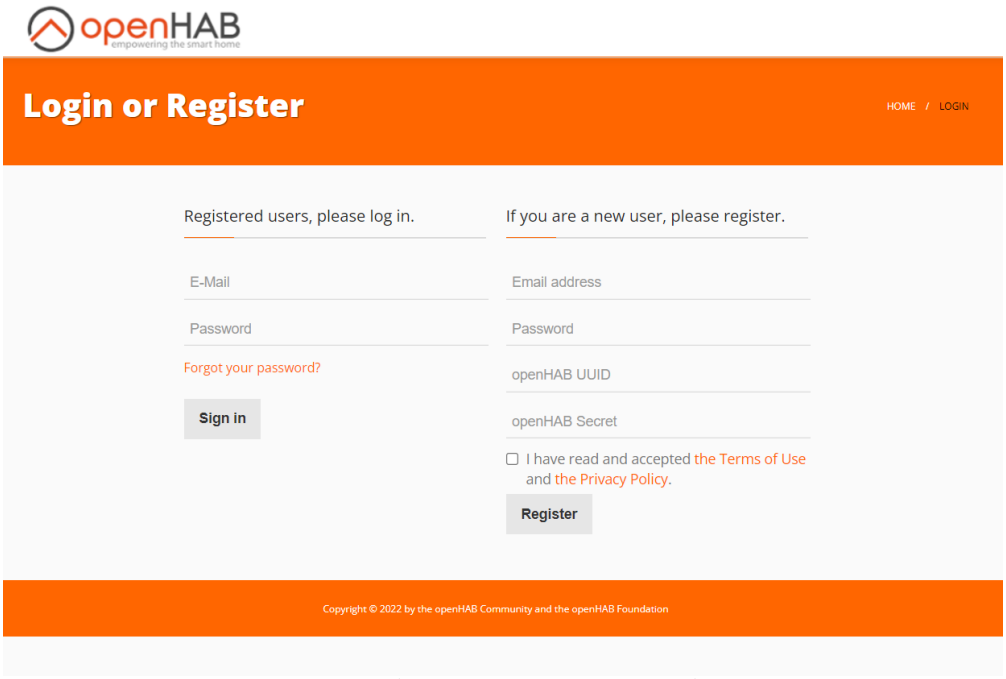


Figure 4.11 – *Account creation on OpenHAB Cloud*

Upon successful registration, the system **redirects to a dashboard** indicating whether the server is currently online. If so, a **link to the server's MainUI interface** is also provided:

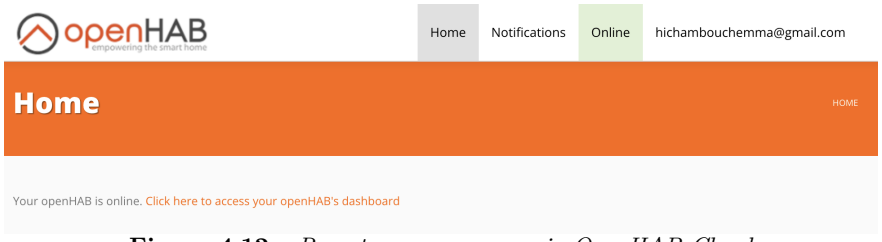


Figure 4.12 – *Remote server access via OpenHAB Cloud*

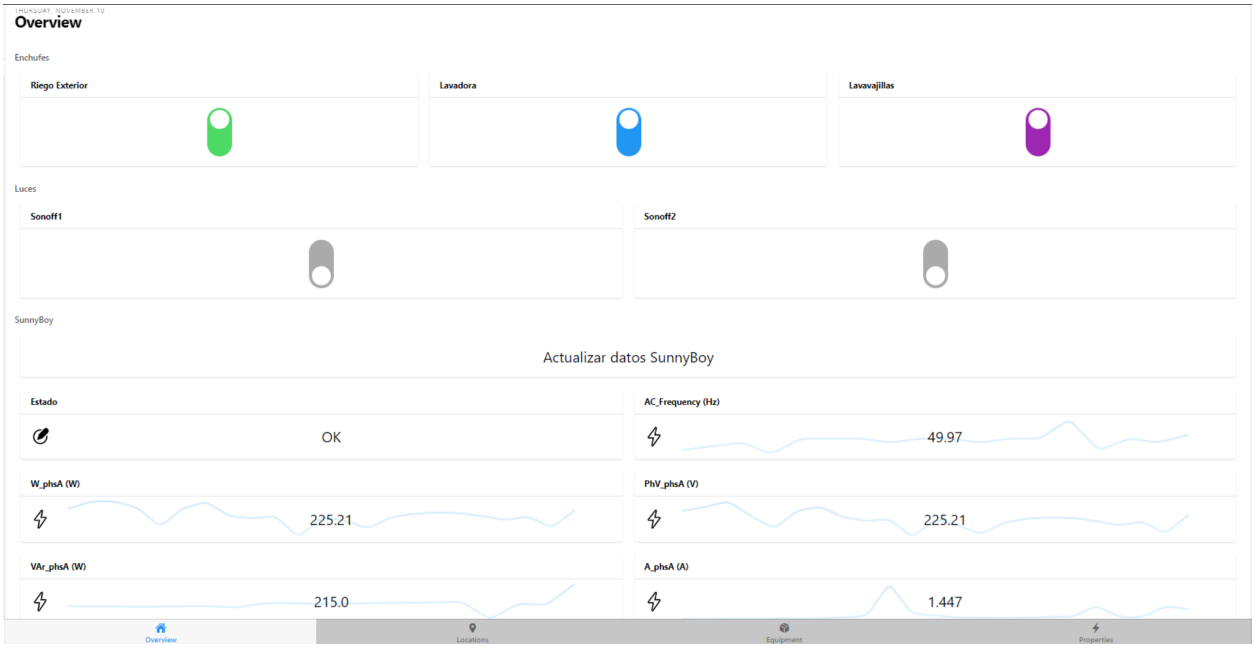


Figure 4.13 – *MainUI dashboard of the remote server*

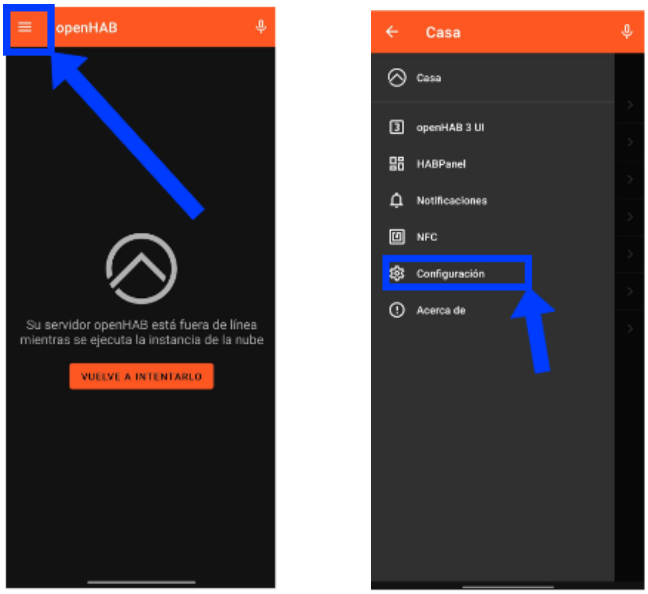
4

Mobile Application Setup

OpenHAB also provides a mobile application for both Android and iOS platforms. This application connects to the OpenHAB server through MyOpenHAB.org.

The configuration steps are as follows:

1. Ensure that an **account has been created** and properly configured in OpenHAB Cloud.
2. Download the official **OpenHAB app** from the Google Play Store or Apple App Store.
3. After installing the application, configure it by linking it to the previously created OpenHAB Cloud account.

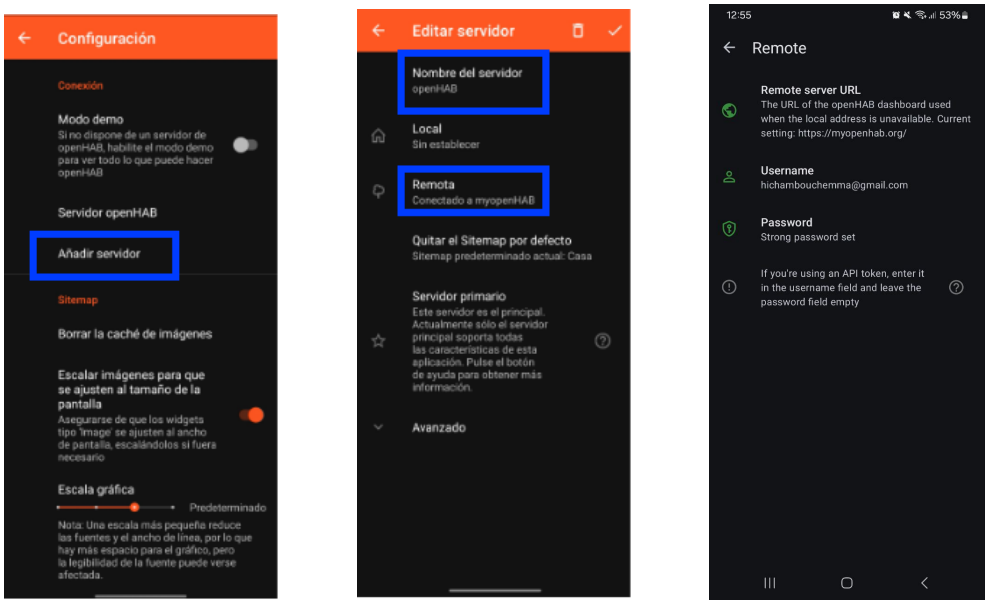


(a) *App launch screen* **(b)** *Accessing configuration*

Figure 4.14 – *Initial setup of the OpenHAB mobile app*

4. Navigate through the app using the following path: `Menu → Settings → Connection → Add Server`. Enter a custom name and select **Remote** connection. Fill in the following fields:

   - **URL address**: `https://myopenhab.org/`
   - **User**: Email address associated with the OpenHAB Cloud account
   - **Password**: Password for the OpenHAB Cloud account



(a) *Connection settings* **(b)** *Server details* **(c)** *Verification*

Figure 4.15 – *Server configuration in the mobile app*

4.2.5 InfluxDB

InfluxDB is a Time series database TSDB developed by the company InfluxData. It is used for the storage and retrieval of **time series data** in fields such as operations monitoring, application metrics, Internet of Things sensor data and real-time analytics [19].

Figure 4.16 – *InfluxDB logo*

The use of a Time series database is particularly well suited for this project, as the **home automation** system generates **time-dependent data** from various devices. Examples include weather conditions, smart plug status, solar energy production, and household energy consumption. This **information is valuable** not only for real-time monitoring, but also for long-term storage and analysis to observe trends and improve energy management [53].

InfluxDB has been selected for this purpose because its specialization in time-series data was mentioned in the software study in 3.2. In addition, the client expressed a preference for retaining InfluxDB. Once the data are collected and stored, it will be visualized using the built-in graphing features of OpenHAB. The seamless integration between OpenHAB and InfluxDB allows for a straightforward configuration process and efficient data visualization.

Some of the key features that make InfluxDB an excellent choice for time series data include:

- A custom **high-performance data store optimized specifically for time series utilizing the TSM** engine for fast ingestion and efficient data compression.
- **High-performance HTTP APIs** for writing and querying data.
- **Support for tags**, which **enable index series and efficient query execution**.
- **Retention policies that automatically expire outdated data**.
- **Continuous queries to precalculate aggregated values**, improving the performance of frequent queries.

4.2.6 Server Architecture

The **Server Architecture** is based on a Raspberry Pi, which utilizes Docker to host both OpenHAB and InfluxDB, each running in its own isolated container, along with additional containers that enhance functionality, such as logging and backup services. InfluxDB will be accessible over the local network for

configuration purposes, while OpenHAB will be available both locally and remotely through the MyOpenHAB.org cloud service, as described in Section 4.2.4.1. Communication will be established between OpenHAB and InfluxDB for data storage and retrieval, as well as between OpenHAB, the Raspberry Pi, and the additional containers to monitor system status and extend capabilities.

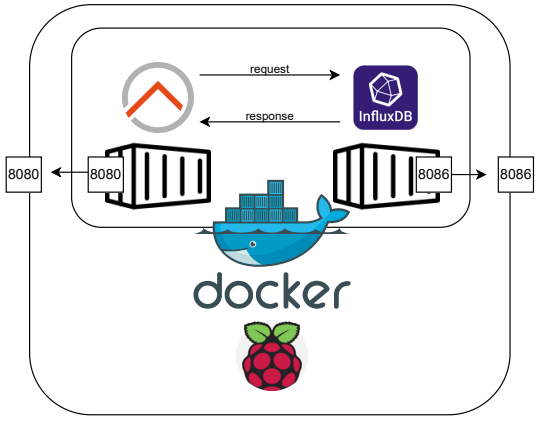


Figure 4.17 – *Server Architecture*

The remote access is achieved through a persistent outbound **WebSocket connection** established by the `openhabcloud` binding installed on the local server [54].

Upon configuration 4.2.4.1, the local instance authenticates using two files: the `uuid` located at `userdata/uuid`, and the `secret` at `userdata/openhabcloud/secret`. These identifiers **link the local instance to the user's OpenHAB Cloud account**.

Once connected, the cloud service acts as a **reverse proxy**. Any requests made via the cloud platform (either through a browser or the OpenHAB mobile app) are **securely relayed** to the local server through the WebSocket tunnel. This method avoids exposing the local network directly and ensures encrypted communication over HTTPS [36].

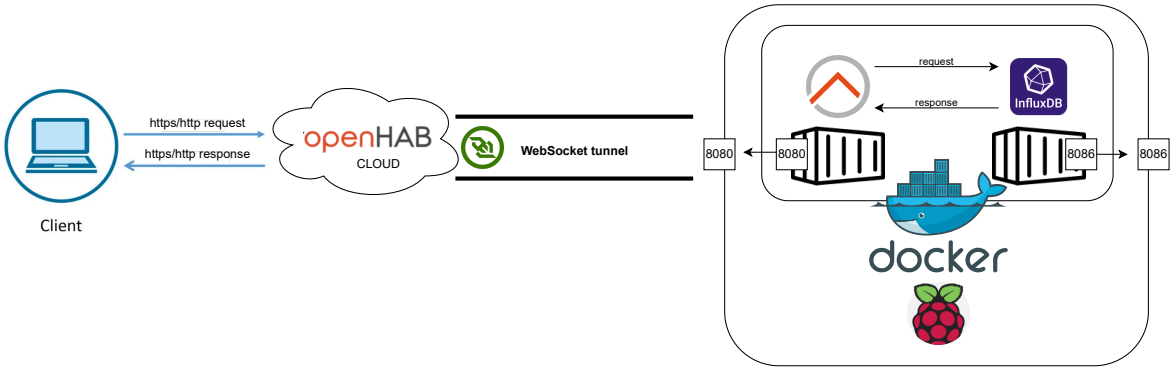


Figure 4.18 – *Remote access*

It is important to highlight the reasons **why the selected server architecture and technologies**—OpenHAB, InfluxDB, and Docker on a Raspberry Pi—are ideal for this project [11, 6, 9, 10]:

- **Simplified Setup and Maintenance:** Docker containers **encapsulate applications along with their dependencies**, streamlining installation and minimizing potential configuration conflicts. This ensures a consistent runtime environment, making it easier to manage and update services over time.
- **Isolation and Resource Management:** Each **service runs in its own isolated container**, allowing multiple applications to operate concurrently on the same device without interfering with one another. This enhances system stability and prevents cross-service issues.
- **Portability and Scalability: Docker enables seamless migration of services** across different hardware or platforms. This portability simplifies scaling or re-deploying the system, which is particularly valuable during tasks such as freeing up storage or reinstalling the operating system. It also plays a critical role in database migration processes.
- **Efficient Resource Utilization: Containers are lightweight and introduce minimal overhead**, making them **well-suited for** resource-constrained devices like the Raspberry Pi. This allows for the concurrent operation of services such as OpenHAB and InfluxDB without significant performance degradation.
- **Simplified Backup and Recovery: Docker facilitates easy backup and restoration through the use of volumes**. This ensures that configurations and data can be preserved and quickly restored in the event of a failure, enhancing system reliability.
- **Optimized Time-Series Data Handling with InfluxDB: InfluxDB is specifically designed for time-series data**, making it an excellent choice for **storing event and telemetry data generated by the home automation system**. Running InfluxDB in a Docker container alongside OpenHAB enables fast querying and efficient long-term storage.
- **Lightweight and Efficient: InfluxDB has a low CPU and memory footprint**, which makes it highly suitable for the Raspberry Pi. It can handle high-frequency data ingestion with minimal performance impact.
- **Seamless Integration with OpenHAB: OpenHAB natively supports InfluxDB**, allowing for straightforward data logging and visualization. Furthermore, compatibility with tools like Grafana enables the creation of real-time dashboards for monitoring and analytics.

Chapter 5

Implementation and configuration

This chapter outlines the steps taken after the project planning and requirements analysis phases. Focuses on the practical aspects of implementing the system in order to achieve the defined objectives. Unlike the previous chapter, which provided the theoretical foundation of the technologies used, this section emphasizes the application of that theory. It also discusses the challenges encountered during the execution phase and details the strategies and solutions used to overcome them.

5.1 Database migration

Before proceeding with database migration, it is essential to clarify some key concepts related to InfluxDB. Understanding the fundamentals of Time series database is crucial to effectively performing the migration process and ensuring proper integration with the rest of the system.

5.1.1 InfluxDB Query Languages

The first step to understand the **query languages** supported by InfluxDB. This process is relatively straightforward, as InfluxDB provides support for two main languages: **InfluxQL** and **Flux** [22].

Flux is a powerful **scripting language** designed specifically to query, analyze, and act on time series data. On the other hand, **InfluxQL** is an **SQL-like query language** designed to work with time series data in InfluxDB. Although it is intuitive for users familiar with traditional SQL, it lacks some of the advanced capabilities found in Flux. Despite these limitations, InfluxQL remains a convenient option for simpler queries and data manipulations.

InfluxQL was chosen for this project due to the **simplicity of the database schema**. InfluxQL provides a clear and efficient way to perform basic queries to view data and interact with tables. Its SQL-like syntax also makes it accessible and well suited for the project's requirements.

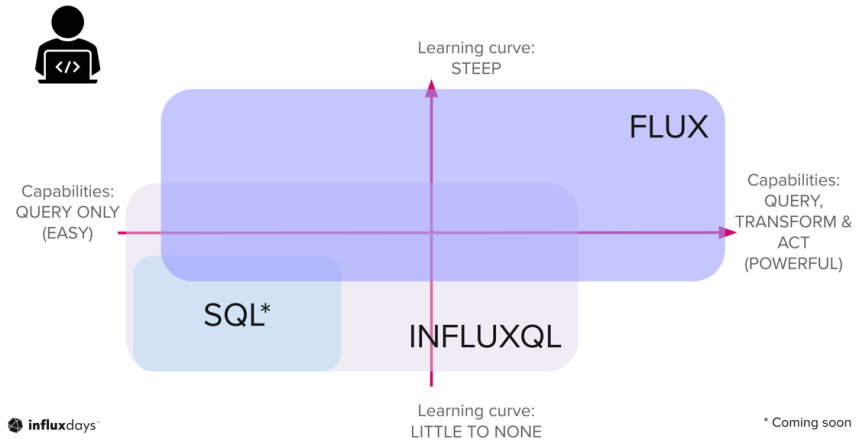


Figure 5.1 – *Flux vs InfluxQL*

5.1.2 InfluxDB Instance configuration

The next step involves understanding **how the InfluxDB instance is configured** and identifying the location of its configuration file. On Linux-based systems, the default configuration file for InfluxDB can typically be found in the following path [25]: `/etc/influxdb/influxdb.conf`

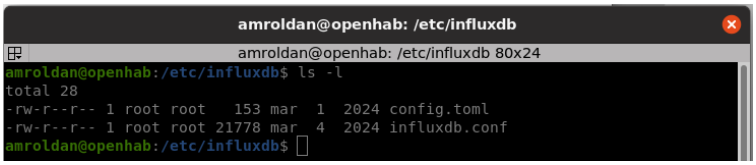


Figure 5.2 – *InfluxDB config file*

InfluxDB instances are **configured** using the **configuration file** (`/etc/influxdb/influxdb.conf`) **or environment variables**. The **environment variable overrides the equivalent option in the configuration file**. If a **configuration option is not explicitly defined**, the **system uses its default setting**. Unless otherwise specified, the configuration settings discussed in this section reflect the default values used by the system.

The **default configuration settings were used**, with the **exception of the HTTP connection parameters**. That section was overridden **to enable HTTP access** [24].

```
[http]
  # Determines whether HTTP endpoint is enabled.
  enabled = true

  # Determines whether the Flux query endpoint is enabled.
  # flux-enabled = false

  # Determines whether the Flux query logging is enabled.
  # flux-log-enabled = false

  # The bind address used by the HTTP service.
  bind-address = ":8086"

  # Determines whether user authentication is enabled over HTTP/HTTPS.
   auth-enabled = true
```

Figure 5.3 – *Http config of data base*

5.1.3 Data Storage in InfluxDB

The third step was to know **where and how InfluxDB stores its data**. The storage structure and terminology presented here apply to InfluxDB v1 and v2.

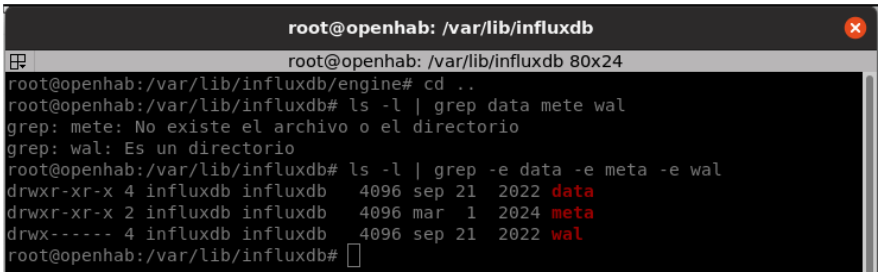


Figure 5.4 – *Data, Metadata, and WAL structure in InfluxDB*

5.1.3.1 Persistent Data

InfluxDB stores persistent data in the directory `/var/lib/influxdb/data` using a specialized file format known as **TSM (Time-Structured Merge tree)**. A database called `openhab_db`, which is **used by OpenHAB**, has been observed.

TSM is a purpose-built data storage format designed **specifically for handling time series data**. It **enables high write and read throughput**, making it highly efficient for large volumes of time-ordered data. This format **also allows for advanced data compaction techniques**, which significantly reduce storage overhead while maintaining access speed [27].

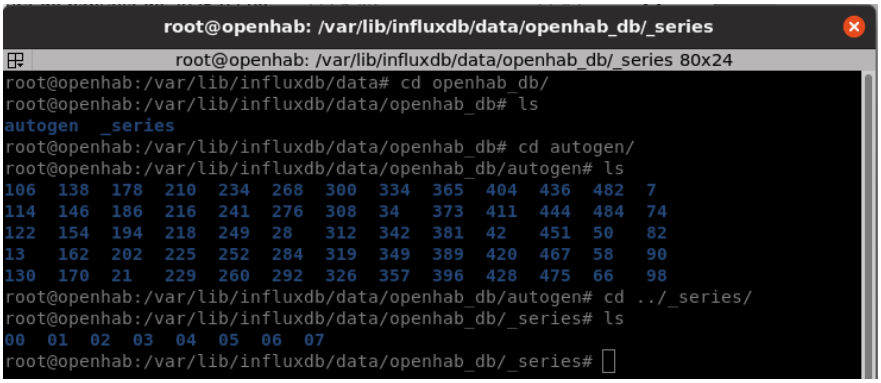


Figure 5.5 – *Persistent Data*

5.1.3.2 Write Ahead Log Data

The **Write-Ahead Log (WAL)** in InfluxDB acts as a **temporary cache** for recently written Point. To reduce the frequency with which permanent storage files are accessed, InfluxDB caches new Point in the WAL until their total size or age **triggers a flush to more permanent storage**. This allows for efficient batching of the writes into the TSM [27]. Additionally, the WAL contains entries from the `openhab_db` databse, as mentioned the database used by OpenHAB .

Point in the WAL can be queried and persist through a system reboot. On process start, all points in

the WAL must be flushed before the system accepts new writes.

Figure 5.6 – *Write Ahead Log Data*

The **data** settings (in the configuration file or through environment variables) control where the actual shard data for InfluxDB are stored and how they are flushed from the WAL. The defaults should work for the project.

WAL Operation Workflow [21]:

1. **Data Ingestion (Write Process)**
   - Data are sent to InfluxDB via HTTP, UDP, or the Line Protocol.
   - New Point are stored in an in-memory cache for fast access.
   - Simultaneously, data are added to the WAL directory (`/var/lib/influxdb/wal/`) to ensure durability in case of crashes.
2. **Data Persistence (Flushing & Compaction)**
   - When the cache reaches a size or time threshold, it flushes to permanent TSM storage (`/var/lib/influxdb/data/`).
   - Corresponding WAL entries are then deleted to free space.
   - Periodic compaction merges small TSM files into larger ones, optimizing disk usage and performance.
   - Deletes and updates are managed using *tombstones*.
3. **Query Execution (Read Process)**
   - When a query is run, InfluxDB retrieves results by merging:
     - Recent data from the memory cache
     - Unflushed data from the WAL
     - Historical data from the TSM files
4. **Maintenance & Cleanup**
   - Retention policies define when old data is automatically deleted.
   - Continuous compaction ensures storage remains optimized over time.

5.1.3.3 Metastore

The Metastore contains essential **internal information required to manage and maintain the InfluxDB instance**. It **stores metadata** such as user accounts, databases, retention policies, shard metadata, continuous queries, and subscriptions. This metadata is crucial for the system to operate correctly and consistently.

All metadata is stored in the file `meta.db`, located in the `meta` directory.

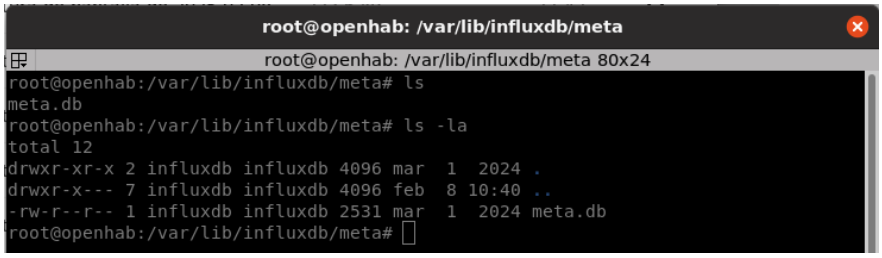


Figure 5.7 – *InfluxDB Metastore File Structure*

In the current configuration, **two key options are enabled**. The first is `retention-autocreate = true`, which allows the system to automatically create a **default retention policy** named `autogen` whenever a new database is created. This policy has **infinite duration** and is used as the default for write and query operations. Disabling this setting would prevent the automatic creation of this default retention policy.

The second setting is `logging-enabled = true`, which activates **log for the meta service**. This feature is useful for debugging and monitoring metadata-related operations.

5.1.3.4 Accessing and Exploring the Existing InfluxDB Data

Now that the **configuration of the database has been reviewed and the data storage structure is understood**, the **next step is to start the InfluxDB instance and investigate its current contents**. This exploration serves to understand the existing data and later verify that the migration process has preserved all relevant information correctly.

- The **first step** involves **starting the InfluxDB v1 service on the host machine**, as shown in Figure 5.8.

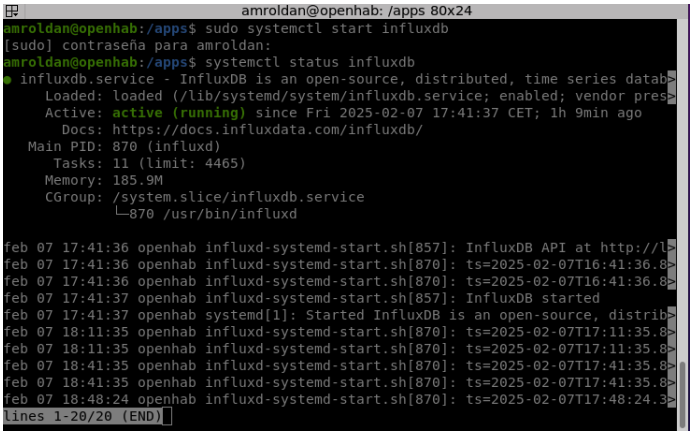


Figure 5.8 – *Starting InfluxDB v1 on the host system*

- Once the service is running, the **next step** is to **enter the InfluxDB shell to begin querying and inspecting the data stored** in the system, to do that it is nesessary to run `use openhab_db` to access the database.

```
amroldan@openhab:/apps/openHAB$ influx -username admin -password SuperSecretPassword123+ -host localhost
Connected to http://localhost:8086 version 1.8.10
InfluxDB shell version: 1.8.10
>
```

Figure 5.9 – *Accessing the InfluxDB shell*

```
> select * from WeatherCompanyObservations_Temperature limit 5
name: WeatherCompanyObservations_Temperature
time                item                                    value
----                ----                                    -----
1663752313199000000 WeatherCompanyObservations_Temperature 22.72222222222222
1663752613415000000 WeatherCompanyObservations_Temperature 23.22222222222222
1663752913613000000 WeatherCompanyObservations_Temperature 23.88888888888889
1663753213856000000 WeatherCompanyObservations_Temperature 23.72222222222222
1663753514321000000 WeatherCompanyObservations_Temperature 23.77777777777778
```

Figure 5.10 – *Example of a basic query in InfluxDB*

It is important to note that although the queried results may appear tabular, **InfluxDB does not store data in traditional tables**. Instead, it uses a **structure based on measurements, Series, tags, and fields**.

- Measurements:

```
Interactive Table View (press q to exit mode, shift+up/down to navigate tables):
Name: measurements
```

index	name
1	AC_Frequency
2	AlexaEchoDotAMRoldan_MediaProgress
3	AlexaEchoDotAMRoldan_Player
4	AlexaEchoDotAMRoldan_Title
5	AlexaEchoDotAMRoldan_Treble
6	AlexaEchoDotAMRoldan_Volume
7	AlexaPlugonPrimerEnchufe_PowerState
8	AstroMoonData_FullMoon
9	AstroMoonData_MoonIllumination
10	AstroMoonData_MoonPhaseName
11	AstroMoonData_NewMoon
12	AstroMoonData_PartialMoonEclipse
13	AstroMoonData_TotalMoonEclipse
14	AstroSunData_DiffuseRadiation
15	AstroSunData_DirectRadiation
16	AstroSunData_PartialEclipse
17	AstroSunData_StartTime
18	AstroSunData_SunPhaseName
19	AstroSunData_TotalEclipse
20	AstroSunData_TotalRadiation
21	AstroSunRiseData_Duration
22	AstroSunRiseData_EndTime
23	AstroSunSetData_Duration
24	AstroSunSetData_EndTime
25	Astro_SunSetStartTime
26	CalentadorAguaTapoP100SmartPlug_OutputSwitch
27	Calentador_Agua_Power_State
28	CurrentL1
29	CurrentL2
30	CurrentL3
31	Echo_Living_Room_ImageUrl
32	Echo_Living_Room_MediaLength
33	Echo_Living_Room_MediaProgressTime
34	EnchufeCirculadoraguacaliente_PowerState
35	EnchufeRiegoExterior_PowerState
36	EnchufeTelePlaystationP100Nuevo_OutputSwitch
37	EnergyForward
38	EnergyForwardL1
39	EnergyForwardL2
40	EnergyForwardL3

```
2 Columns, 151 Rows, Page 1/4
Table 1/1, Statement 1/1
```

Figure 5.11 – *Measurements available in the database*

- Fields:

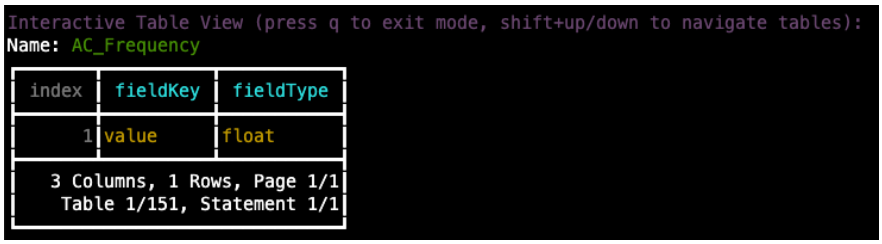


Figure 5.12 – *Field keys in InfluxDB*

- Tags:

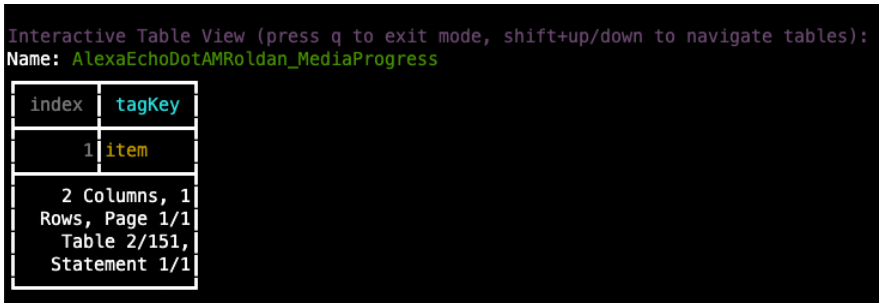


Figure 5.13 – *Tag keys in InfluxDB*

- Series:

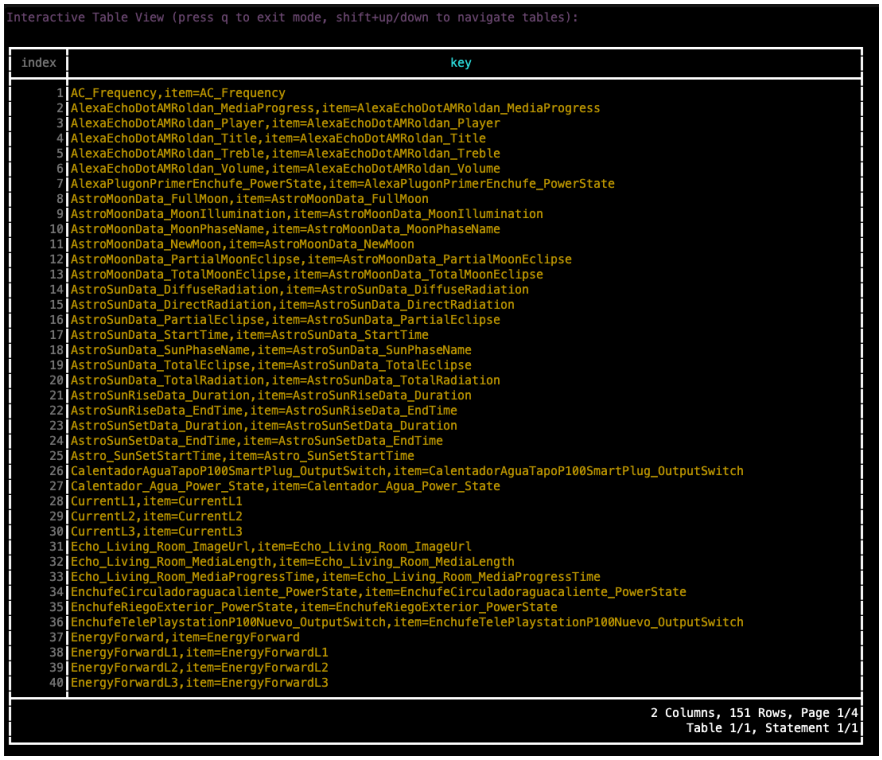


Figure 5.14 – *Series in InfluxDB*

5.1.3.5 Upgrade to InfluxDB v2

The upgrade process from InfluxDB v1 to v2 was relatively straightforward due to the availability of an **automatic upgrade tool**. The upgrade was performed by executing the `influxd upgrade` command, which **automates the conversion of configuration and data** [23].

This process performs **several key steps**:

- It **reads the existing InfluxDB 1.x configuration file** and **generates an equivalent InfluxDB 2.7 configuration file** at `~/.influxdbv2/config.toml`, or at a custom location specified using the `-v2-config-path` flag.
- It **updates metadata and storage engine** paths to `~/.influxdbv2/meta` and `~/.influxdbv2/engine`, respectively, unless other paths are explicitly defined.
- **Migration of existing data and Write-Ahead Log (WAL)** files to **InfluxDB 2.7 Bucket**.
- **Creates database and retention policy (DBRP)** mappings that are required to query data using InfluxQL. The `autogen` Retention Policy is converted to an infinite Retention Period. **This is done to enable the use of the InfluxDB v1 shell**.
- It reads existing **metadata** and migrates nonadmin users, passwords, and permissions into a 1.x authorization compatible store located at `~/.influxdbv2/influxdb.bolt`.

After executing the upgrade, when InfluxDB 2.7 is started for the first time, it must build a new **Time Series Index (TSI)**. Depending on the volume of existing data, this indexing process may take a significant amount of time to complete.

InfluxDB v2 introduces **new terminology and structural changes** compared to the v1.x version. Among these, three concepts are particularly important to understand:

- **Organization**: A **workspace for a group of users**. All elements such as dashboards, tasks, buckets, and user roles belong to a specific organization.
- **Bucket**: A bucket is a named **storage location for time series data**. Buckets have associated retention period. Each bucket is tied to a single organization.
- **Retention Period**: The **duration of time that a bucket retains data**. InfluxDB automatically removes data points with timestamps older than the defined retention period. The minimum duration allowed for retention is one hour. This concept replaces the *retention policy* (RP) used in InfluxDB v1.

Once the InfluxDB is upgraded, the next process is to migrate the required bucket, which is the `openhab_db` bucket, to a Docker container. Due to the upgrade **the database was converted to a Bucket**.

5.1.3.6 Migration to Docker-based InfluxDB v2

Once the upgrade to InfluxDB v2 was completed, the **migration process** became significantly easier, as both backup and restore operations were performed **within the same version of the database engine**. The main goal of this migration was to **move the InfluxDB openhab_db bucket** from a host-based setup **to a Docker container**, which allows better portability, isolation, and ease of deployment.

The first step was to inspect the **current running containers**:

```
amroldan@openhab:/apps/openHAB$ cat docker-compose.yml
version: '2.2'

services:
  openhab:
    build: .
    restart: always
    ports:
      - "8101:8101"
    network_mode: host
    volumes:
      - "/etc/localtime:/etc/localtime:ro"
      - "/etc/timezone:/etc/timezone:ro"
      - "./openhab_addons:/openhab/addons"
      - "./openhab_conf:/openhab/conf"
      - "./openhab_userdata:/openhab/userdata"
    environment:
      OPENHAB_HTTP_PORT: "8080"
      OPENHAB_HTTPS_PORT: "8443"
      EXTRA_JAVA_OPTS: "-Duser.timezone=Europe/Madrid"

#volumes:
#  openhab_addons:
#    driver: local
#  openhab_conf:
#    driver: local
#  openhab_userdata:
#    driver: local
```

Figure 5.15 – *Existing containers on the system*

Next, a **provisional Docker container** was created for the InfluxDB v2 instance:

```
  influxdb2:
    image: influxdb:latest
    container_name: influxdb2
    restart: always
    volumes:
      - "./data_influxdb2:/var/lib/influxdb2"
      - "./data_influxdb2_config:/etc/influxdb2"
    ports:
      - "8086:8086"
```

Figure 5.16 – *Provisional InfluxDB v2 Docker container*

The **setup and access of the containers are explained in more detail in Section 5.3**. It is recommended to review that section for a clearer understanding of the following steps.

After deploying the container, the InfluxDB v2 **setup process was completed within the container**. This involved configuring **essential parameters** such as the Organization name, initial user credentials, and the first bucket. **Completing this setup is crucial as it ensures that the restore process can be performed successfully** [26]. These concepts are explained in Section 5.1.3.5.

```
root@92af85a95fe3:/# influx setup
> Welcome to InfluxDB 2.0!
? Please type your primary username admin
? Please type your password ***************
? Please type your password again ***************
? Please type your primary organization name adminOrg
? Please type your primary bucket name openhab_db
? Please type your retention period in hours, or 0 for infinite 0
? Setup with these parameters?
  Username:          admin
  Organization:      adminOrg
  Bucket:            openhab_db
  Retention Period:  infinite
 Yes
User    Organization    Bucket
admin   adminOrg        openhab_db
root@92af85a95fe3:/#
```

Figure 5.17 – *InfluxDB v2 initial setup in the Docker container*

Backup and Restore Process

InfluxDB v2 provides a streamlined mechanism for backing up and restoring data. Thanks to Docker, managing these operations becomes straightforward and reproducible [29].

1. Creating a Backup on the host

The **process started by performing a backup of the relevant Bucket** using the InfluxDB CLI:

```
influx backup -openhab_db /home/granasat/openHAB/influx2_backup
```

Figure 5.18 – *Creating a backup on the host system*

The output of this operation provided confirmation of the backup files and metadata.

```
/02/07 18:41:37 INFO: Downloading metadata snapshot
/02/07 18:41:37 INFO: Backing up TSM for shard 1
/02/07 18:41:37 INFO: Backing up TSM for shard 2
/02/07 18:41:37 INFO: Backing up TSM for shard 3
/02/07 18:41:37 INFO: Backing up TSM for shard 4
/02/07 18:41:37 INFO: Backing up TSM for shard 5
/02/07 18:41:37 INFO: Backing up TSM for shard 6
/02/07 18:41:38 INFO: Backing up TSM for shard 7
/02/07 18:41:38 INFO: Backing up TSM for shard 8
/02/07 18:41:38 INFO: Backing up TSM for shard 9
/02/07 18:41:38 INFO: Backing up TSM for shard 10
/02/07 18:41:38 INFO: Backing up TSM for shard 11
/02/07 18:41:39 INFO: Backing up TSM for shard 12
/02/07 18:41:39 INFO: Backing up TSM for shard 13
/02/07 18:41:39 INFO: Backing up TSM for shard 14
/02/07 18:41:39 INFO: Backing up TSM for shard 15
/02/07 18:41:40 INFO: Backing up TSM for shard 16
/02/07 18:41:40 INFO: Backing up TSM for shard 17
/02/07 18:41:40 INFO: Backing up TSM for shard 18
/02/07 18:41:40 INFO: Backing up TSM for shard 19
/02/07 18:41:41 INFO: Backing up TSM for shard 20
/02/07 18:41:41 INFO: Backing up TSM for shard 21
/02/07 18:41:41 INFO: Backing up TSM for shard 22
/02/07 18:41:41 INFO: Backing up TSM for shard 23
/02/07 18:41:42 INFO: Backing up TSM for shard 24
/02/07 18:41:42 INFO: Backing up TSM for shard 25
/02/07 18:41:42 INFO: Backing up TSM for shard 26
/02/07 18:41:42 INFO: Backing up TSM for shard 27
```

Figure 5.19 – *Backup output*

2. *Transfer of the backup to the Docker Container*

The name or ID of the target Docker container was recovered using `docker ps`:

```
granasat@granasat:~/openHAB $ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED       STATUS                 PORTS                                              NAMES
bff580a2c27b   welteki/frontail-openhab:latest   "docker-entrypoint.s…"   12 days ago   Up 12 days             0.0.0.0:9001->9001/tcp, :::9001->9001/tcp          frontail
1473fd190655   openhab_openhab                "/entrypoint gosu op…"   12 days ago   Up 12 days (healthy)                                                      openhab
69f676e0a5ac   influxdb:latest                "/entrypoint.sh infl…"   12 days ago   Up 12 days             0.0.0.0:8086->8086/tcp, :::8086->8086/tcp          influxdb2
granasat@granasat:~/openHAB $
```

Figure 5.20 – *Identifying the InfluxDB Docker container*

The **backup directory was then copied into the Docker container**:

```
granasat@granasat:~/openHAB/influx2_backup $ docker cp ./ influxdb2:/home/influx2_backup
granasat@granasat:~/openHAB/influx2_backup $ docker exec -it influxdb2 /bin/bash
root@69f676e0a5ac:/# ls
LICENSE     bin   dev                      entrypoint.sh  home    influxdb2-2.7.6  media  opt   root  sbin  sys  usr
README.md   boot  docker-entrypoint-initdb.d  etc            influx  lib              mnt    proc  run   srv   tmp  var
root@69f676e0a5ac:/# cd home/
root@69f676e0a5ac:/home# ls influx2_backup/
20250207T184137Z.1.tar.gz    20250207T184137Z.20.tar.gz   20250207T184137Z.326.tar.gz  20250207T184137Z.49.tar.gz  20250207T184137Z.65.tar.gz
20250207T184137Z.10.tar.gz   20250207T184137Z.21.tar.gz   20250207T184137Z.33.tar.gz   20250207T184137Z.5.tar.gz   20250207T184137Z.66.tar.gz
20250207T184137Z.103.tar.gz  20250207T184137Z.22.tar.gz   20250207T184137Z.34.tar.gz   20250207T184137Z.50.tar.gz  20250207T184137Z.67.tar.gz
20250207T184137Z.11.tar.gz   20250207T184137Z.23.tar.gz   20250207T184137Z.35.tar.gz   20250207T184137Z.51.tar.gz  20250207T184137Z.7.tar.gz
20250207T184137Z.111.tar.gz  20250207T184137Z.24.tar.gz   20250207T184137Z.36.tar.gz   20250207T184137Z.52.tar.gz  20250207T184137Z.71.tar.gz
20250207T184137Z.119.tar.gz  20250207T184137Z.25.tar.gz   20250207T184137Z.37.tar.gz   20250207T184137Z.53.tar.gz  20250207T184137Z.74.tar.gz
20250207T184137Z.12.tar.gz   20250207T184137Z.26.tar.gz   20250207T184137Z.38.tar.gz   20250207T184137Z.54.tar.gz  20250207T184137Z.75.tar.gz
20250207T184137Z.127.tar.gz  20250207T184137Z.27.tar.gz   20250207T184137Z.39.tar.gz   20250207T184137Z.55.tar.gz  20250207T184137Z.76.tar.gz
20250207T184137Z.13.tar.gz   20250207T184137Z.28.tar.gz   20250207T184137Z.4.tar.gz    20250207T184137Z.56.tar.gz  20250207T184137Z.77.tar.gz
20250207T184137Z.135.tar.gz  20250207T184137Z.29.tar.gz   20250207T184137Z.40.tar.gz   20250207T184137Z.57.tar.gz  20250207T184137Z.78.tar.gz
20250207T184137Z.14.tar.gz   20250207T184137Z.293.tar.gz  20250207T184137Z.41.tar.gz   20250207T184137Z.58.tar.gz  20250207T184137Z.8.tar.gz
20250207T184137Z.15.tar.gz   20250207T184137Z.3.tar.gz    20250207T184137Z.42.tar.gz   20250207T184137Z.59.tar.gz  20250207T184137Z.87.tar.gz
20250207T184137Z.159.tar.gz  20250207T184137Z.30.tar.gz   20250207T184137Z.43.tar.gz   20250207T184137Z.6.tar.gz   20250207T184137Z.9.tar.gz
20250207T184137Z.16.tar.gz   20250207T184137Z.302.tar.gz  20250207T184137Z.44.tar.gz   20250207T184137Z.60.tar.gz  20250207T184137Z.95.tar.gz
20250207T184137Z.17.tar.gz   20250207T184137Z.31.tar.gz   20250207T184137Z.45.tar.gz   20250207T184137Z.61.tar.gz  20250207T184137Z.bolt.gz
20250207T184137Z.18.tar.gz   20250207T184137Z.310.tar.gz  20250207T184137Z.46.tar.gz   20250207T184137Z.62.tar.gz  20250207T184137Z.manifest
20250207T184137Z.19.tar.gz   20250207T184137Z.318.tar.gz  20250207T184137Z.47.tar.gz   20250207T184137Z.63.tar.gz  20250207T184137Z.sqlite.gz
20250207T184137Z.2.tar.gz    20250207T184137Z.32.tar.gz   20250207T184137Z.48.tar.gz   20250207T184137Z.64.tar.gz
root@69f676e0a5ac:/home#
```

Figure 5.21 – *Copying the backup to the InfluxDB container*

3. *Restoring the Bucket Inside the Container*

With the **backup available inside the container**, the restore command was executed to **import the data into the appropriate bucket**.

```
granasat@granasat:~/openHAB $ docker exec -it influxdb2 /bin/bash
root@69f676e0a5ac:/# influx restore -b openhab_db /home/influx2_backup
```

Figure 5.22 – *Restoring the backup in the Docker container*

4. *Verification of the data*

After the restoration process, **access to the InfluxDB UI confirmed that the data had been successfully migrated** and were available in the new Docker-based environment:

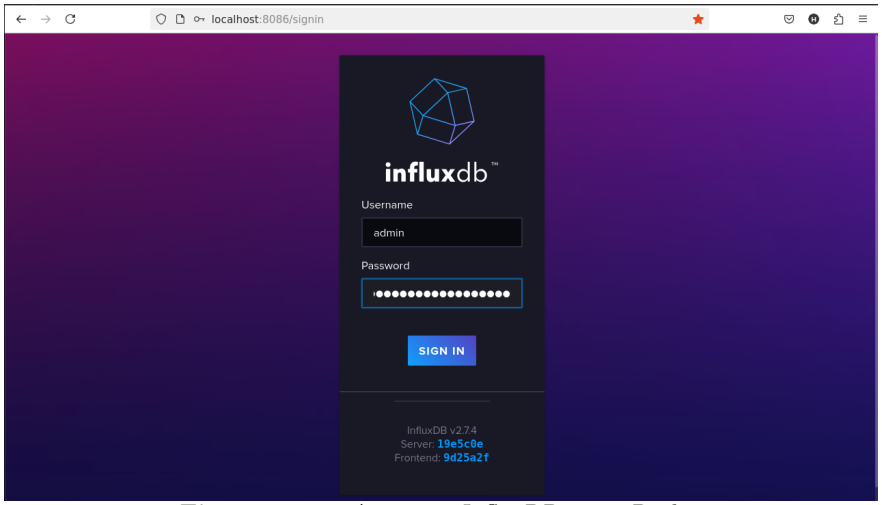


Figure 5.23 – *Accessing InfluxDB v2 in Docker*

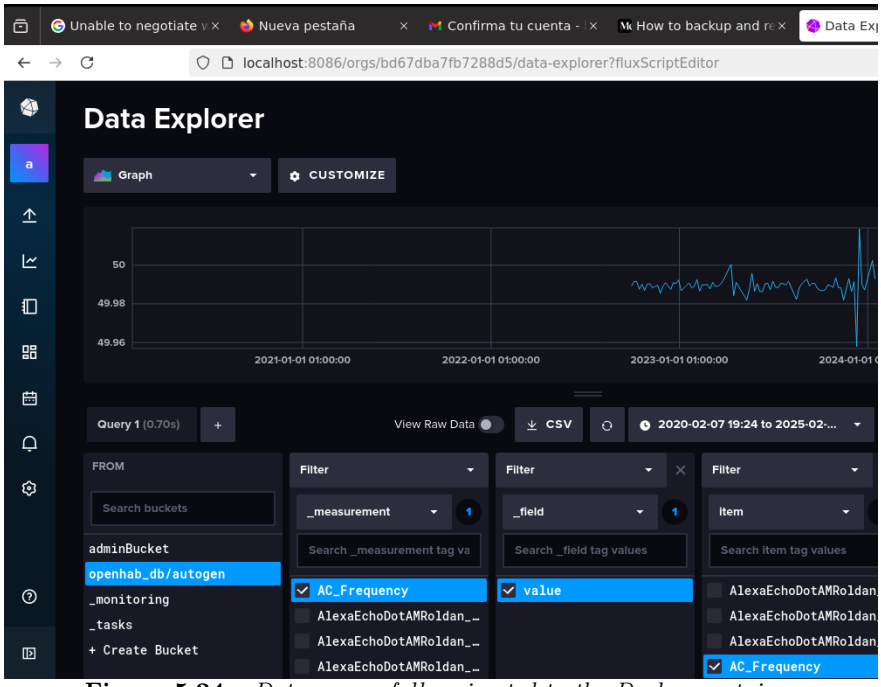


Figure 5.24 – *Data successfully migrated to the Docker container*

This migration demonstrated how seamless it is to use Docker for InfluxDB management. With the use of **volumes and containerized environments**, it becomes easy to **maintain the entire home automation system** in a single folder and move it effortlessly between different machines or environments.

5.2 Raspberry Pi Setup and System Migration

The tasks related to the Raspberry Pi involved **installing a headless operating system**. Furthermore, usfing a new server provided a clean environment to migrate the home automation system [40, 41].

First, a headless version of the Raspberry Pi 5 OS was installed. The installation process involved flashing the OS image onto an SD card and configuring essential settings. After preparing the SD card, the system was **set up to allow SSH access** without needing a graphical interface [58].

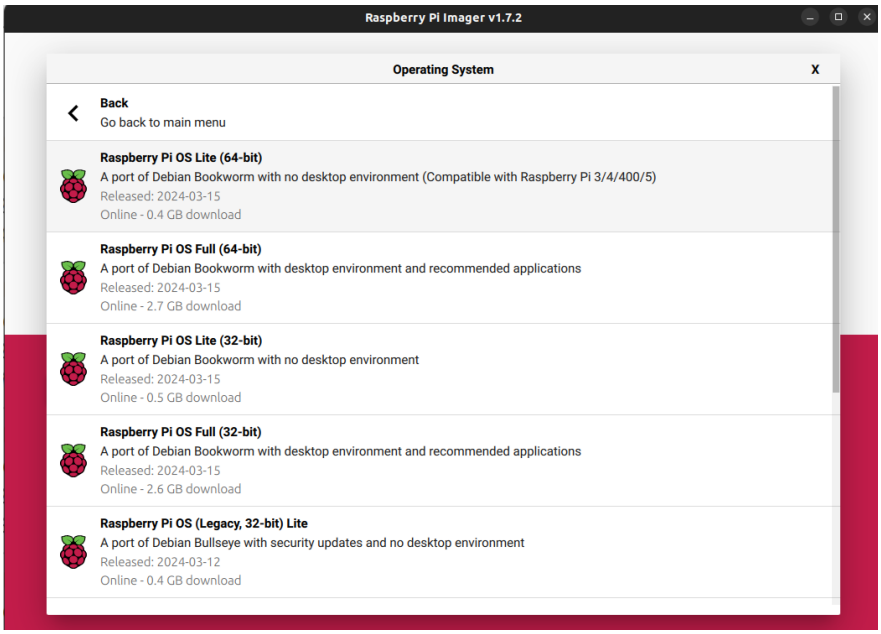


Figure 5.25 – *Installing Raspberry Pi OS (headless version)*

Once the OS was installed, the **basic configuration was completed**.

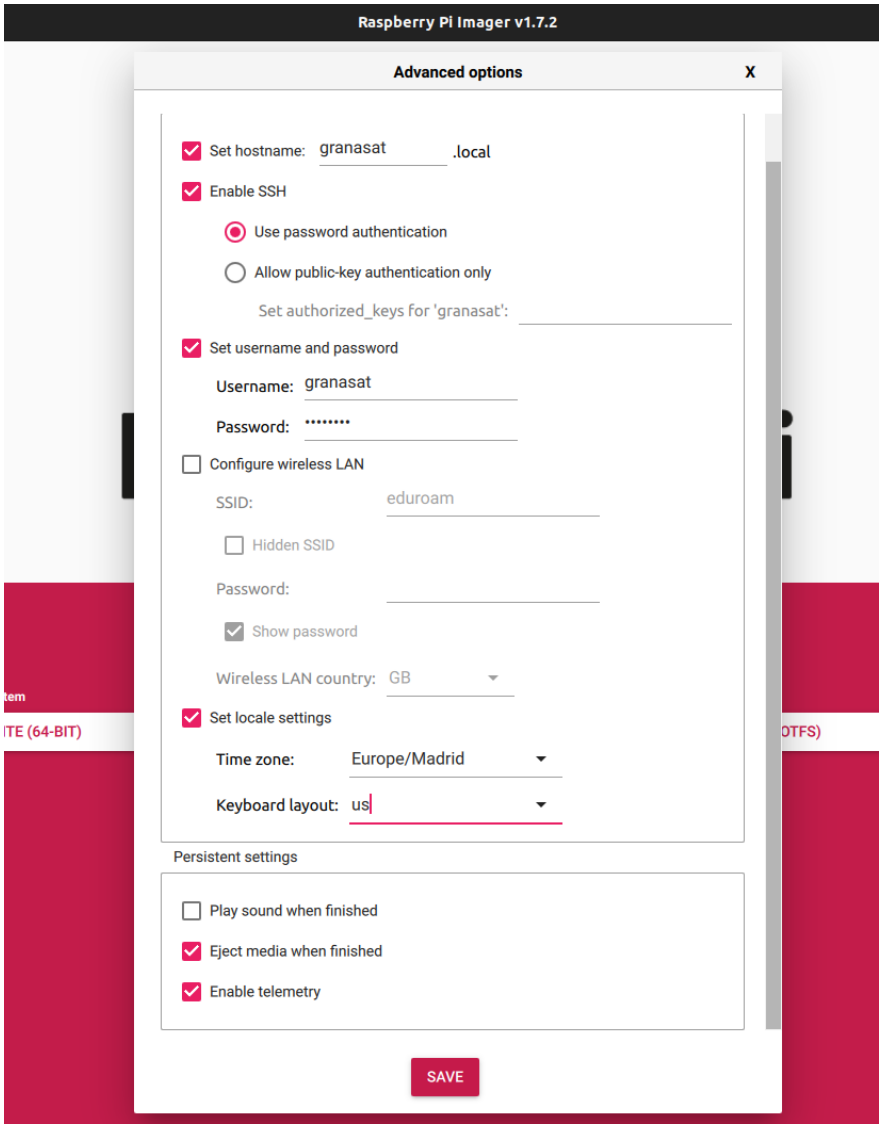


Figure 5.26 – *Configuring Raspberry Pi OS*

To **establish a connection**, the local network was scanned to **find the IP address assigned to the Raspberry Pi**.

```
12:07:40 cpeep@cpeep ~ → sudo arp-scan --localnet
Interface: eno1, type: EN10MB, MAC: e4:e7:49:42:94:b6, IPv4: 192.168.1.130
Starting arp-scan 1.9.7 with 256 hosts (https://github.com/royhills/arp-scan)
192.168.1.1     f8:8e:85:00:f0:47       Comtrend Corporation
192.168.1.140   dc:a6:32:66:6f:09       Raspberry Pi Trading Ltd

2 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9.7: 256 hosts scanned in 1.960 seconds (130.61 hosts/sec). 2 responded
12:07:47 cpeep@cpeep ~ →
```

Figure 5.27 – *Identifying the Raspberry Pi's IP address*

After finding the device on the network, **SSH keys** were generated, and a **secure connection was established** to the Raspberry Pi.

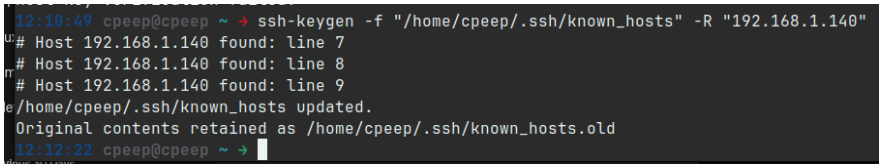


Figure 5.28 – *Generating SSH keys*

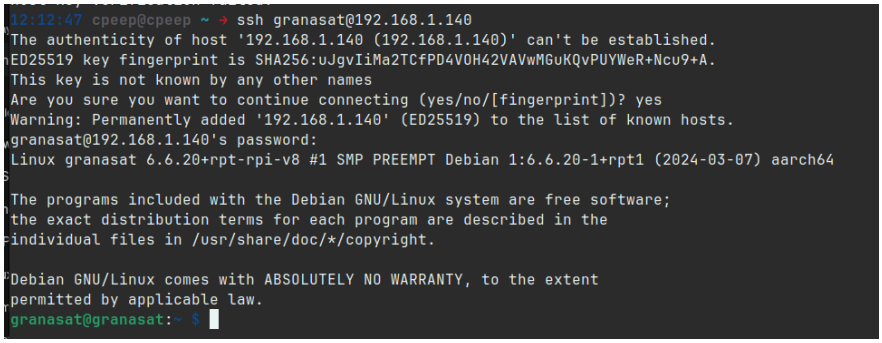


Figure 5.29 – *Connecting to the Raspberry Pi via SSH*

After establishing access, a **backup** was prepared by **copying the home directory** of previous server host to a **USB device**. The USB also **contained the OpenHAB project folder**, intended to be migrated to the Raspberry Pi via its USB port.

Thanks to Docker, the entire **home automation system**, including data, configurations, and hardware integrations, was **unified and portable**. The portability and scalability of Docker significantly simplified the migration process, as described in Section 4.2.2.

The first step in the backup process was to list the available USB devices connected to the Raspberry Pi.

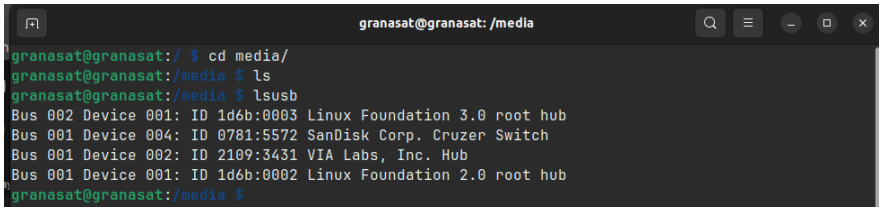


Figure 5.30 – *Listing USB devices*

The USB storage connected was then verified.

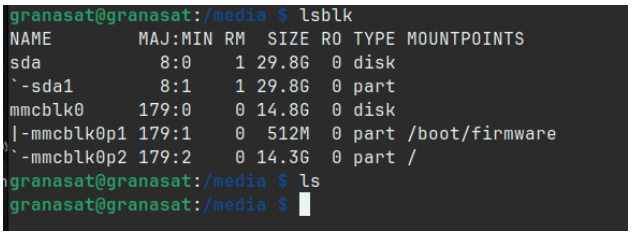


Figure 5.31 – *Viewing USB disk*

The **USB disk was mounted in the `/media`** directory to allow file access and transfer. Finally, the project archive was **decompressed to restore its contents**.

```
granasat@granasat:/ $ cd media/
granasat@granasat:/media $ ls
'Brigth minds'
'Los Amos Del Aire [HDTV][Cap.101](wolfmax4k.com).avi'
'Los Amos Del Aire [HDTV][Cap.102](wolfmax4k.com).avi'
'Los Amos Del Aire [HDTV][Cap.103](wolfmax4k.com).avi'
'Los Amos Del Aire [HDTV][Cap.104](wolfmax4k.com).avi'
'Los Amos Del Aire [HDTV][Cap.105](wolfmax4k.com).avi'
'Los Amos Del Aire [HDTV][Cap.107](wolfmax4k.com).avi'
'Los Amos Del Aire [HDTV][Cap.108](wolfmax4k.com).avi'
'Los amos del aire [HDTV][Cap.106](wolfmax4k.com).avi'
 home_rasp.tar
granasat@granasat:/media $ sudo tar -xvf home_rasp.tar -C /home/
home/
home/granasat/
home/granasat/ver_espacio_disco.sh
home/granasat/.selected_editor
home/granasat/.sudo_as_admin_successful
home/granasat/.pp_backup/
home/granasat/.pp_backup/.config/
home/granasat/.pp_backup/.config/wf-panel-pi.ini
home/granasat/.pp_backup/.config/libfm/
home/granasat/.pp_backup/.config/libfm/libfm.conf
home/granasat/.pp_backup/.config/lxterminal/
```

Figure 5.32 – *Decompressing the project archive*

With the Raspberry Pi environment now ready and the **project migrated**, the next logical step was to set up GitHub to manage the configuration and version control of the home automation system. Since this process is widely known, it will not be documented.

5.3 Containerized System Setup with Docker

This section outlines the process of **installing and configuring Docker** on the target operating system, building and managing Docker Images for the home automation system, and leveraging Docker Compose for easier **orchestration**. The goal is to **unify all relevant services**, including OpenHAB and InfluxDB, within a consistent and manageable containerized environment.

5.3.1 Installing Docker and Docker Compose

To begin with, **Docker is installed** on the operating system. Figures 5.33 through 5.35 illustrate the process of installing Docker and configuring **user permissions**.

```
granasat@granasat:~ $ curl -sSL https://get.docker.com | sh
# Executing docker install script, commit: 6d9743e9656cc56f699a64800b098d5ea5a60020
+ sudo -E sh -c apt-get update -qq >/dev/null
+ sudo -E sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq apt-transport-https ca-cer
tificates curl >/dev/null
apt-listchanges: Can't set locale; make sure $LC_* and $LANG are correct!
perl: warning: Setting locale failed.
perl: warning: Please check that your locale settings:
        LANGUAGE = (unset),
        LC_ALL = (unset),
        LC_TIME = "es_ES.UTF-8",
        LC_MONETARY = "es_ES.UTF-8",
        LC_CTYPE = "en_GB.UTF-8",
        LC_ADDRESS = "es_ES.UTF-8",
        LC_TELEPHONE = "es_ES.UTF-8",
        LC_NAME = "es_ES.UTF-8",
        LC_MEASUREMENT = "es_ES.UTF-8",
        LC_IDENTIFICATION = "es_ES.UTF-8",
        LC_NUMERIC = "es_ES.UTF-8",
        LC_PAPER = "es_ES.UTF-8",
        LANG = "en_GB.UTF-8"
    are supported and installed on your system.
perl: warning: Falling back to a fallback locale ("en_GB.UTF-8").
locale: Cannot set LC_ALL to default locale: No such file or directory
+ sudo -E sh -c install -m 0755 -d /etc/apt/keyrings
```

Figure 5.33 – *Installing Docker*

```
granasat@granasat:~ $ docker ps
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.
sock: Get "http://%2Fvar%2Frun%2Fdocker.sock/v1.45/containers/json": dial unix /var/run/docker.s
ock: connect: permission denied
granasat@granasat:~ $ sudo usermod -aG docker ${USER}
```

Figure 5.34 – *Adding Docker to Sudoers*

```
granasat@granasat:~ $ su - ${USER}
Password:
granasat@granasat:~ $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
granasat@granasat:~ $
```

Figure 5.35 – *Docker Permissions Confirmed*

Next, **Docker Compose is installed**. This tool simplifies the definition and management of multicontainer applications using a YAML configuration file.

```
granasat@granasat: ~                                    You can paste the image from the clipboar
granasat@granasat:~ $ sudo apt install docker-compose
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libltdl7 libslirp0 pigz slirp4netns
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  cgroupfs-mount containerd criu docker.io libintl-perl libintl-xs-perl libmodule-find-perl
  libmodule-scandeps-perl libnet1 libproc-processtable-perl libprotobuf-c1 libprotobuf32
  libsort-naturally-perl libterm-readkey-perl needrestart python3-attr python3-docker
  python3-dockerpty python3-docopt python3-dotenv python3-json-pointer python3-jsonschema
  python3-protobuf python3-pyrsistent python3-rfc3987 python3-texttable python3-uritemplate
  python3-webcolors python3-websocket python3-yaml runc tini
Suggested packages:
```

Figure 5.36 – *Installing Docker Compose*

Once Docker and Docker Compose are set up, the environment is ready to **host the home automation system** with the latest containerized versions of OpenHAB and InfluxDB. Understanding the Docker ecosystem, including how containers interconnect and communicate, is essential before deployment. **For a theoretical background, refer to Sections 4.2.2 and 4.2.3.**

5.3.2 Building Docker Images

5.3.2.1 OpenHAB Docker Image

The OpenHAB service requires a custom Docker Image, built using a Dockerfile. This file contains a **sequence of instructions** for the assembly of the image, as illustrated in Figure 5.37.

```
FROM openhab/openhab:latest

RUN  apt-get update && apt-get install -y python3 \
    && apt-get install -y python3-pip \
    && apt-get -y install python3.11-venv

ENV VIRTUAL_ENV=/opt/venv
RUN python3 -m venv $VIRTUAL_ENV
ENV PATH="$VIRTUAL_ENV/bin:$PATH"

RUN python3 -m pip install setuptools wheel && python3 -m pip install pymodbus==2.4.0
```

Figure 5.37 – *OpenHAB Dockerfile*

As explained in Section 4.2.2, Docker Images are built in layers. The OpenHAB image begins with an **official base image** pulled from Docker Hub, which ensures stability and community support (Figure 5.38).

```
<missing>      6 days ago     CMD ["gosu" "openhab" "tini" "-s" "./start.s…   0B        buildkit.dockerfile.v0
<missing>      6 days ago     ENTRYPOINT ["/entrypoint"]                       0B        buildkit.dockerfile.v0
<missing>      6 days ago     RUN |4 BUILD_DATE=2025-02-20T16:40:12Z VCS_R…    0B        buildkit.dockerfile.v0
<missing>      6 days ago     COPY entrypoint /entrypoint # buildkit           4.31kB    buildkit.dockerfile.v0
<missing>      6 days ago     WORKDIR /openhab                                 0B        buildkit.dockerfile.v0
<missing>      6 days ago     HEALTHCHECK &{["CMD-SHELL" "curl -f http://l…    0B        buildkit.dockerfile.v0
<missing>      6 days ago     EXPOSE map[5007/tcp:{} 8080/tcp:{} 8101/tcp:…    0B        buildkit.dockerfile.v0
<missing>      6 days ago     VOLUME [/openhab/conf /openhab/userdata /ope…    0B        buildkit.dockerfile.v0
<missing>      6 days ago     RUN |4 BUILD_DATE=2025-02-20T16:40:12Z VCS_R…    0B        buildkit.dockerfile.v0
<missing>      6 days ago     COPY update /openhab/runtime/bin/update # bu…    7.25kB    buildkit.dockerfile.v0
<missing>      6 days ago     RUN |4 BUILD_DATE=2025-02-20T16:40:12Z VCS_R…    124MB     buildkit.dockerfile.v0
<missing>      6 days ago     RUN |4 BUILD_DATE=2025-02-20T16:40:12Z VCS_R…    58.1kB    buildkit.dockerfile.v0
<missing>      6 days ago     RUN |4 BUILD_DATE=2025-02-20T16:40:12Z VCS_R…    480MB     buildkit.dockerfile.v0
<missing>      6 days ago     SHELL [/bin/bash -o pipefail -c]                 0B        buildkit.dockerfile.v0
<missing>      6 days ago     LABEL org.opencontainers.image.created=2025-…    0B        buildkit.dockerfile.v0
<missing>      6 days ago     ENV CRYPTO_POLICY=limited EXTRA_JAVA_OPTS= E…    0B        buildkit.dockerfile.v0
<missing>      6 days ago     ARG OPENHAB_VERSION=4.3.3                        0B        buildkit.dockerfile.v0
<missing>      6 days ago     ARG JAVA_VERSION=17                              0B        buildkit.dockerfile.v0
<missing>      6 days ago     ARG VCS_REF=4957653df9cba06d72f1e495916df037…    0B        buildkit.dockerfile.v0
<missing>      6 days ago     ARG BUILD_DATE=2025-02-20T16:40:12Z              0B        buildkit.dockerfile.v0
<missing>      2 months ago   # debian.sh --arch 'arm64' out/ 'bookworm' '…   97.2MB    debuerreotype 0.15
```

Figure 5.38 – *OpenHAB Image Layers*

Each **additional command** in the Dockerfile adds a **new layer**, customizing the base image for the automation system's requirements (Figure 5.39).

```
hichamb@SP00354 openHAB % docker image history openhab
IMAGE          CREATED        CREATED BY                                      SIZE      COMMENT
be7b44c750fb   24 hours ago   RUN /bin/bash -o pipefail -c python3 -m pip …   3.04MB    buildkit.dockerfile.v0
<missing>      24 hours ago   ENV PATH=/opt/venv/bin:/usr/local/sbin:/usr/…   0B        buildkit.dockerfile.v0
<missing>      24 hours ago   RUN /bin/bash -o pipefail -c python3 -m venv…   21.6MB    buildkit.dockerfile.v0
<missing>      24 hours ago   ENV VIRTUAL_ENV=/opt/venv                       0B        buildkit.dockerfile.v0
<missing>      24 hours ago   RUN /bin/bash -o pipefail -c apt-get update …   420MB     buildkit.dockerfile.v0
<missing>      6 days ago     CMD ["gosu" "openhab" "tini" "-s" "./start.s…   0B        buildkit.dockerfile.v0
```

Figure 5.39 – *New Layers Added to OpenHAB Image*

The **command perform the following tasks**:

- `FROM`
  Specifies the base image for the Docker build.
- `RUN`
  Executes a shell command during the image build process. Each `RUN` command creates a new layer.
- `ENV`
  Sets environment variables in the image. These variables persist and are available to the container at runtime.

The **layers perform the following tasks**:

- `FROM openhab/openhab:latest`
  Uses the latest official OpenHAB Docker Image as the base.
- `RUN apt-get update && apt-get install -y python3 \`
  `&& apt-get install -y python3-pip \`
  `&& apt-get -y install python3.11-venv`
  Installs Python 3, pip, and the `venv` module for creating virtual environments.
- `ENV VIRTUAL_ENV=/opt/venv`
  Defines the path for the Python virtual environment.
- `RUN python3 -m venv $VIRTUAL_ENV`
  Creates the virtual environment at `/opt/venv`.
- `ENV PATH="$VIRTUAL_ENV/bin:$PATH"`
  Ensures the virtual environment's `bin` directory is in the shell's path.
- `RUN python3 -m pip install setuptools wheel && python3 -m pip install pymodbus==2.4.0`
  Installs Python packaging tools and the specific version 2.4.0 of the `pymodbus` library, used for Modbus communication.

The reason for this Docker Image configuration will be explained in Section 5.8.

Each of these commands is a Docker layer, and their order and structure directly affect **image efficiency**. Redundant layers can increase the image size and rebuild time, so **combining related commands** into a single layer where possible is considered **best practice**.

To analyze the structure and **history of the built image**, the `docker history <image-name>` command can be used. This provides a detailed overview of how the image was built and helps identify optimization opportunities.

Understanding how Docker layers work, analyzing build history, and writing efficient Dockerfile instructions are essential for **building maintainable and performant** containerized applications.

5.3.2.2 InfluxDB Image

For InfluxDB, **no customization** is necessary. The official image from Docker Hub is sufficient. Figure 5.40 shows the command used to **pull the image**.

```
hichamb@SP00354 openHAB % docker pull influxdb
Using default tag: latest
latest: Pulling from library/influxdb
d9b636547744: Pull complete
1ab2539f5934: Pull complete
8cd1b82bf9bd: Pull complete
0fefb2cb2fd3: Pull complete
30334ae92818: Pull complete
bb86366f0c9b: Pull complete
f0c63fdd53f0: Pull complete
d854b2c94de0: Pull complete
1960380a2e8b: Pull complete
7a005551db93: Pull complete
Digest: sha256:e20505e98b485b5d764937ded954ef12d7f0888e5c36c4955747ef850c2b9f8b
Status: Downloaded newer image for influxdb:latest
```

Figure 5.40 – *Pulling the InfluxDB Image*

Once both images are available locally, they can be listed using the `docker images` command (Figure 5.41).

```
hichamb@SP00354 openHAB % docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
openhab      latest    be7b44c750fb   5 weeks ago    1.15GB
influxdb     latest    e6f3c3669b03   2 months ago   419MB
hichamb@SP00354 openHAB %
```

Figure 5.41 – *Listing Docker Images*

5.3.2.3 Running the Containers

With the **images built and pulled**, continers can be launched using standard Docker commands.

```
hichamb@SP00354 openHAB % docker run -d \
  --name openhab \
  --restart always \
  --privileged \
  --network host \
openhab
dfc28dea8d8f3831b7447f772571b90ca450d2bd190b40e74c9a6b9abafdf85b
hichamb@SP00354 openHAB % docker run -d \
  --name influxdb \
  --restart always \
  -p 8086:8086 \
  influxdb
48e5c45c34538b429546c8cc508e67fccf9108661048a6a92d9dff61936444e9
hichamb@SP00354 openHAB %
```

Figure 5.42 – *Running OpenHAB and InfluxDB Containers*

To **verify running containers**, the `docker ps` command is used (Figure 5.46).

```
hichamb@SP00354 openHAB % docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS                            PORTS                    NAMES
48e5c45c3453   influxdb   "/entrypoint.sh infl…"   20 seconds ago   Up 19 seconds                     0.0.0.0:8086->8086/tcp   influxdb
dfc28dea8d8f   openhab    "/entrypoint gosu op…"   3 minutes ago    Up 3 minutes (health: starting)                            openhab
hichamb@SP00354 openHAB %
```

Figure 5.43 – *Listing Running Containers*

5.3.3 Simplifying Setup with Docker Compose

As discussed in Section 4.2.3, all the steps previously **carried out manually** using individual Docker commands can be **greatly simplified** using Docker Compose. This is especially useful for managing multi-container applications such as our home automation system with OpenHAB and InfluxDB.

Docker Compose allows us to define and **run multi-container** Docker applications **using a single YAML configuration file**. Instead of executing multiple commands, we **define our system** in a file called `docker-compose.yml`, which **handles everything**: building images, setting up networks, volumes, ports, environment variables, and service dependencies.

```yaml
services:
  openhab:
    build: .
    container_name: openhab
    restart: always
    privileged: true
    depends_on:
      - influxdb2
    volumes:
      - "/etc/localtime:/etc/localtime:ro"
      - "/etc/timezone:/etc/timezone:ro"
      - "./openhab_addons:/openhab/addons"
      - "./openhab_conf:/openhab/conf"
      - "./openhab_userdata:/openhab/userdata"
    environment:
      OPENHAB_HTTP_PORT: "8080"
      OPENHAB_HTTPS_PORT: "8443"
      EXTRA_JAVA_OPTS: "-Duser.timezone=Europe/Madrid"
    network_mode: "host"

  influxdb2:
    image: influxdb:latest
    container_name: influxdb2
    restart: always
    volumes:
      - "./data_influxdb2:/var/lib/influxdb2"
      - "./data_influxdb2_config:/etc/influxdb2"
    ports:
      - "8086:8086"
    networks:
      backend-nw:
        ipv4_address: 10.5.0.6


networks:
  backend-nw:
    driver: bridge
    ipam:
     config:
       - subnet: 10.5.0.0/16
         gateway: 10.5.0.1
```

Figure 5.44 – *Structure of the docker-compose.yml file*

Compose File Overview

The `docker-compose.yml` **file includes**:

- ***Services:*** **Containers to run** (e.g., OpenHAB, InfluxDB)
- ***Images or Build Contexts:*** **Image sources or Dockerfile paths**
- ***Volumes and Ports:*** **Mappings between host and containers**
- ***Networks:*** **Container interconnections**
- ***Environment Variables:*** **Configuration for services**

Services

- **OpenHAB**
  - `build: .` — **Specifies the directory** with the Dockerfile.
  - `container_name: openhab`
  - `restart: always` — Ensures the container **restarts automatically on failure**.
  - `privileged: true` — Grants **extended privileges**, such as access to host devices.
  - `depends_on:` — **Waits for** `influxdb2` to be ready before starting.
  - `volumes:` — **Mounts directories for configuration, userdata, and Add-on**.
  - `environment:` — Sets **HTTP/HTTPS ports and Java timezone**.
  - `network_mode: "host"` — **Shares the host network** stack.
- **InfluxDB**
  - `image: influxdb:latest`
  - `container_name: influxdb2`
  - `restart: always`
  - `volumes:` — **Maps persistent storage for InfluxDB data and configuration**.
  - `ports:` — **Exposes port 8086 for the web UI and API**.
  - `networks: backend-nw` — Uses a **custom bridge network with a fixed IP**.

Networks

- **backend-nw**
  - `driver: bridge`
  - `ipam:` — **Manually assigns IP range**:
    - Subnet: `10.5.0.0/16`
    - Gateway: `10.5.0.1`

Networks will be detailed with more attention in Section 5.7, which was the reason for their creation.

Running the System

To start the containers defined in the compose file, simply run:

```
docker compose up
```

Docker Compose will:

1. **Parse the YAML file**
2. **Build or pull any required images**
3. **Create networks and volumes**
4. **Start all containers in the correct order**

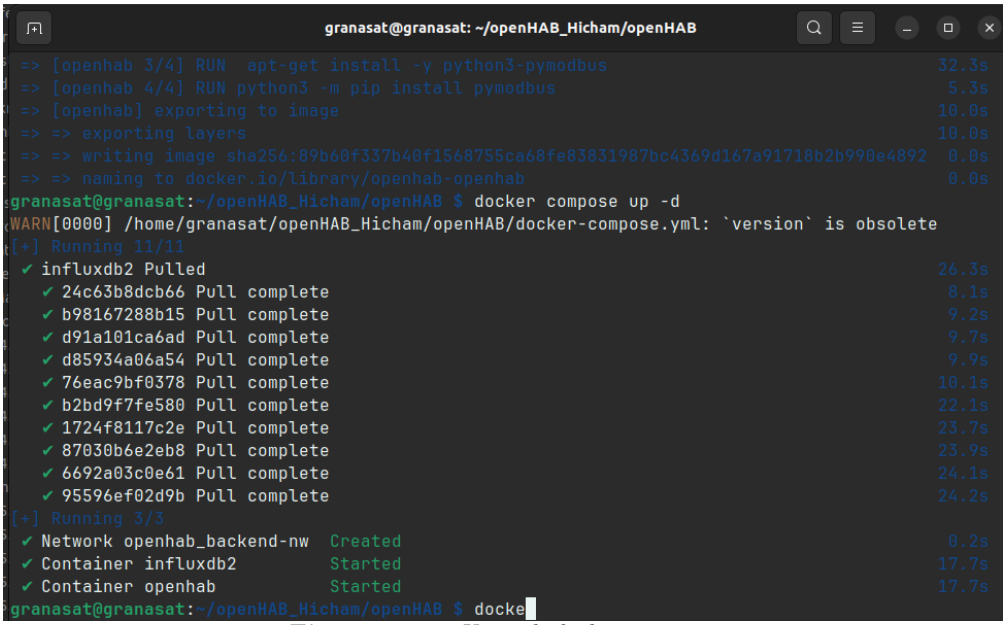


Figure 5.45 – *Up with docker compose*

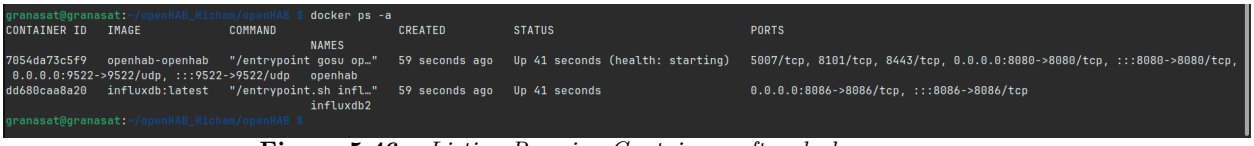


Figure 5.46 – *Listing Running Containers after docker compose up*

It is important to note that if the Docker Images are updated or the Dockerfile is modified, **it is necessary** to run the command **`docker compose build -no-cache`** before executing `docker compose up -d`. This ensures that the **updates are properly incorporated** into the images. Otherwise, Docker Compose may use **cached layers** and skip the new changes.

Figure 5.47 – *Building without cache*

In this specific case, an **update** to OpenHAB was performed. After executing **`docker compose build -no-cache`** followed by **`docker compose up -d`**, the container was **successfully rebuilt** with the latest version of OpenHAB, as specified by the **`latest`** tag in the Dockerfile.

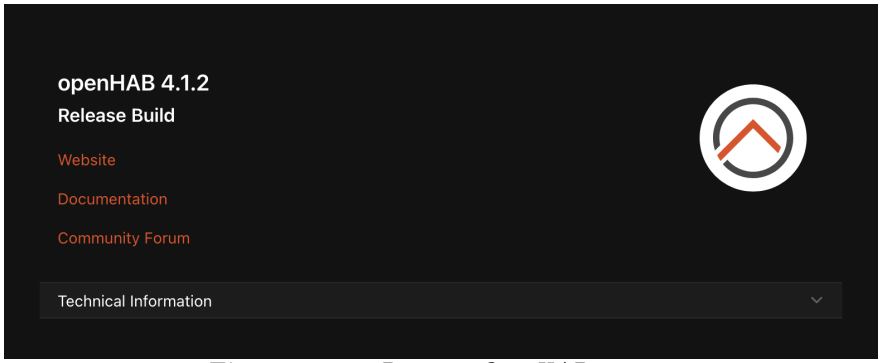


Figure 5.48 – *Previous OpenHAB version*

Figure 5.49 – *Updated OpenHAB version*

It is also recommended to **stop and remove** any running continers before **rebuilding the images**. This can be done using **`docker compose down`**, followed by **removing the outdated images** to ensure a **clean and error-free build process**.

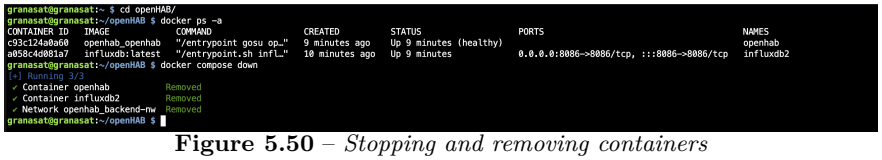


Figure 5.50 – *Stopping and removing containers*

```
granasat@granasat:~/openHAB $ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
granasat@granasat:~/openHAB $ docker images
REPOSITORY         TAG       IMAGE ID       CREATED         SIZE
influxdb           latest    c8acfacfc831   12 months ago   389MB
openhab/openhab    latest    e33c759923b9   13 months ago   657MB
granasat@granasat:~/openHAB $ docker rmi $(docker images -a -q)
Untagged: influxdb:latest
Untagged: influxdb@sha256:166d2c5d89dd4a726709a9cf661236ab5d3efb6f9b6aa7765ae405e1d5b5c0e2
Deleted: sha256:c8acfacfc83128f2e92d41bea9c1bb8e15e8ff7027b39dcef185e22ab6fbe197
Deleted: sha256:91de67e3c7f007747fa22dc5f85e09c9a5a381e4ced08b63bb70706b2797545d
Deleted: sha256:96c424902a14c306b72655869ed1867c8d8594ac14b160700932439e5c260f22
Deleted: sha256:1cffdce263b146f2139f8591d8859549dfa8ec255181bab702ac49e484e195a8
Deleted: sha256:f4e1aff98518b1f0c6e12c4bdb0acf7da4e8c7ea20721300a32e988e3f135d05
Deleted: sha256:1665db08dbc5e1187dbbe13d7cf8c22f968fcb9265b9f5bd269f53843ba7cc35
Deleted: sha256:249d7b4b818aff1a7e8bb2e2aad037806cc85779d957f4f13400a9576e4dc68e
Deleted: sha256:916bf21246ffebaf5cde4e33dd70b4be99d49e040b502bf9f2d452b3b4fdebd3
Deleted: sha256:48665eaa0548ec6ea267001dc32a393fef2080c415afb9859c4938cb90032de5
Deleted: sha256:19e982e22c4b01f037662c827eb09f626522dfca2533d3a2f1a0370c9c841c86
Deleted: sha256:2bd1a2222589b50b52ff960c3d004829633df61532e7a670a91618cd775f2d47
Untagged: openhab/openhab:latest
Untagged: openhab/openhab@sha256:73b9ba282b79faccd0f036f7d857ea0ace2834a1105fd0c6c87d58979c4d72bb
Deleted: sha256:e33c759923b9e7e576f835511cd16615a222b496be65d32a4c1c96ff042cb78b
granasat@granasat:~/openHAB $ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
granasat@granasat:~/openHAB $
```

Figure 5.51 – *Deleting Docker images*

Basic Docker Compose Commands

Compose provides several useful commands for managing the system will be used:

- **`docker compose up -d`** — Starts all services in background
- **`docker compose down`** — Stops and removes containers, networks, and volumes
- **`docker compose logs`** — Displays logs from all services
- **`docker exec -it <image name> <service>`** — Executes a command inside a running container in interactive mode
- **`docker compose build`** — Builds or rebuilds images

Why Use Docker Compose?

- **Simplified Configuration:** All service definitions in a single readable YAML file
- **Reproducibility:** Ensures consistent setups across development, testing, and production
- **Portability:** Easily shared in version control (e.g., Git)
- **Efficiency:** Reduces manual effort and chances of error

In summary, Docker Compose transforms a multi-command, manual setup into a highly manageable, repeatable, and scalable workflow with just a few lines of configuration.

5.4 Establishing a `tmate` Connection

After verifying that the system is up and running via `docker compose`, and confirming that both OpenHAB and InfluxDB are accessible, the next step is to **establish a remote connection** using `tmate` to enable online collaboration and debugging [48].

Figure 5.52 – *`tmate` running*

`Tmate` is a terminal sharing tool that enables **secure remote access to a terminal session**. It is particularly useful because it:

- **Works seamlessly through** NAT.
- **Tolerates dynamic IP changes**.
- **Requires no manual SSH key configuration**.

Clients connect transparently via `tmate.io` servers, which act as a proxy. This makes it extremely simple to access a remote machine securely.

5

Standard Remote Access with `tmate`

The usual way to initiate a remote session is by running the `tmate` command. Once started, `tmate` will output an **SSH connection string** such as:

```
ssh PMhmes4XeKQyBR2JtvnQt6BJw@nyc1.tmate.io
```

Anyone with this link can join the **shared terminal session**, and all users will see the same terminal content in real-time.

Creating Named Sessions with API Key

By default, `tmate` generates **random session strings**, which change upon each restart. To avoid this and maintain a **consistent connection string**, **named sessions** are used. This requires an **API key** and allows deterministic URLs like:

```
ssh username/session-name@nyc1.tmate.io
```

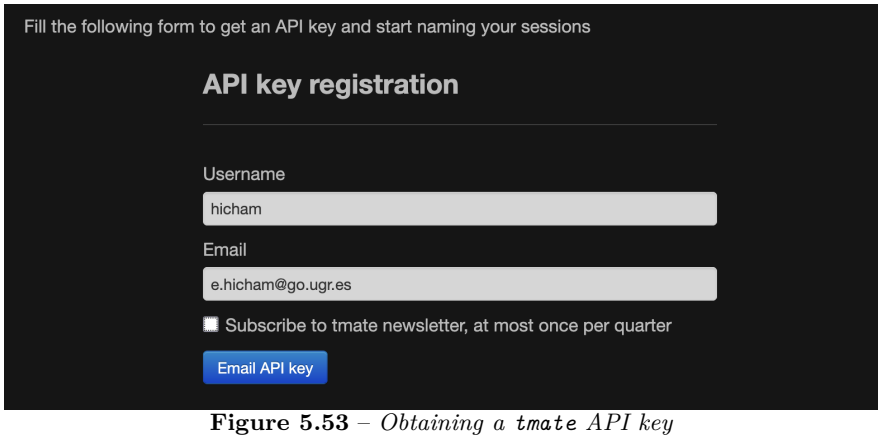


Figure 5.53 – *Obtaining a* `tmate` *API key*

The `username` is defined during **Application Programming Interface () key registration**, and the **session name** is provided when starting `tmate`.

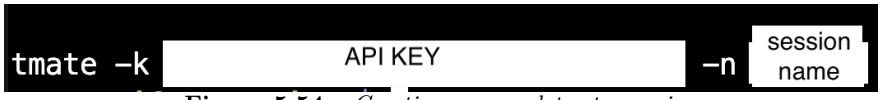


Figure 5.54 – *Creating a named* `tmate` *session*

Security Note: Named sessions must use **hard-to-guess names** to prevent unauthorized access, as these session names act like passwords.

Connecting to a Session

Once a session is active, it can be accessed via:

- **SSH** using the named or generated URL.
- **Web interface** provided in the terminal output.

An example of the output provided by `tmate` when a session is launched is shown below, with `abcdef1234567890` as the name of the session:

```
ssh session:         ssh abcdef1234567890@ny.tmate.io
ssh session read-only: ssh ro-abcdef1234567890@ny.tmate.io
web session:         https://tmate.io/t/abcdef1234567890
web session read-only: https://tmate.io/t/ro-abcdef1234567890
```

Figure 5.55 – *Session connection output from* `tmate`

5.5 Fixing Thing Connections

With remote access established and the **system up and running**, OpenHAB was initialized and the **troubleshooting process** began. Before adding new devices, it was necessary to first **resolve connectivity issues** with the existing ones. Most Things were correctly recognized after updating OpenHAB. The first step was to ensure that all bindings were up to date:

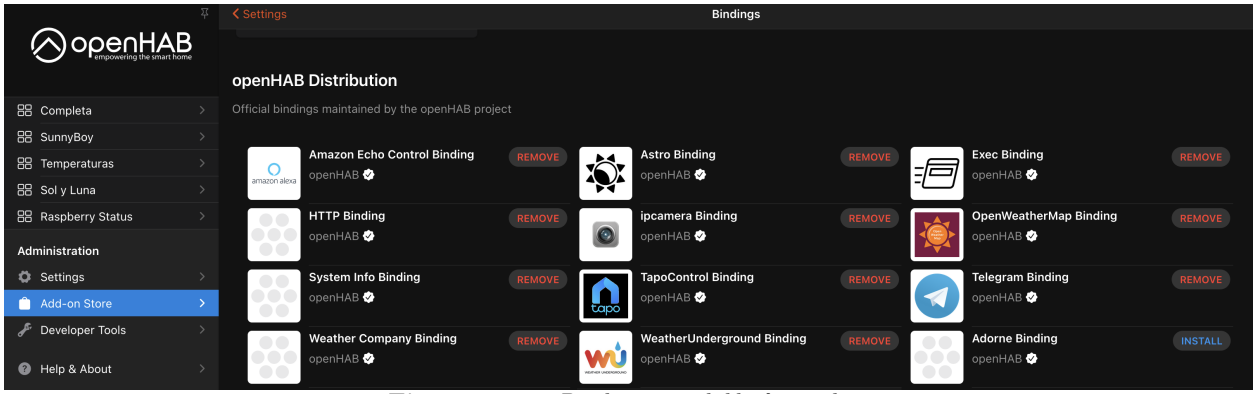


Figure 5.56 – *Bindings available for update*

Once the **bindings were updated**, the process continued in the **Things tab**, where devices were grouped by binding. This allowed for easier detection and resolution of issues:

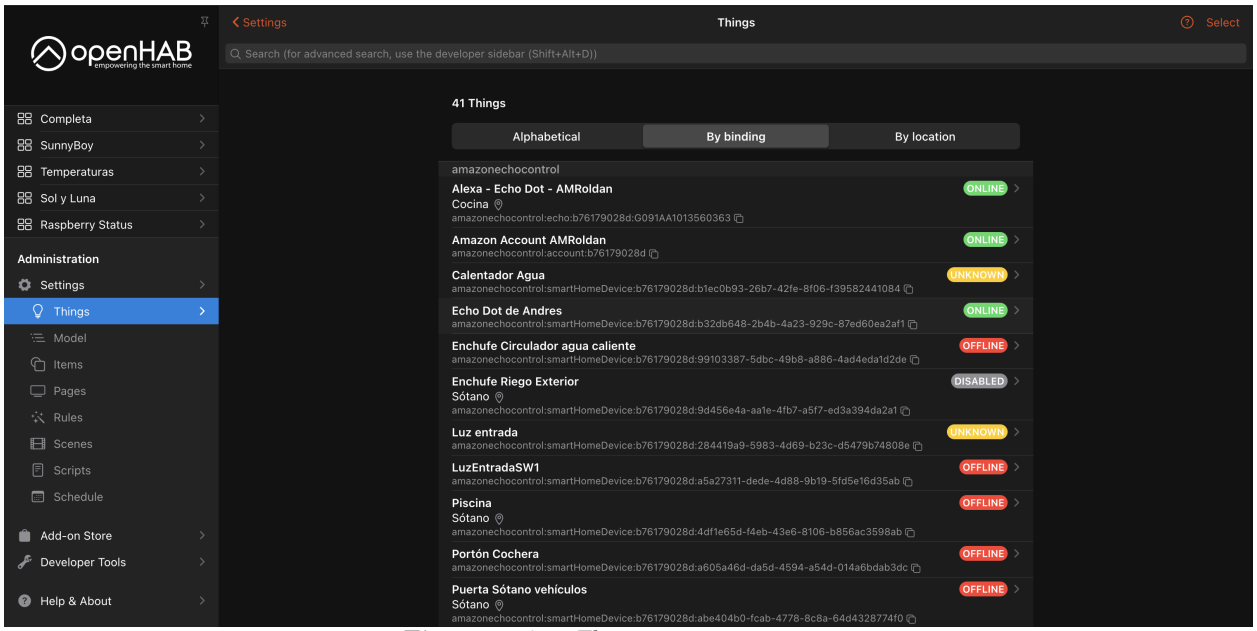


Figure 5.57 – *Things requiring attention*

The most common issues identified were:

1. **Device offline:** The sensor or Thing was disconnected or unavailable.
2. **Disabled Thing:** The device was manually disabled in the interface.
3. **Bridge errors:** Some bindings require a special type of Thing called a *bridge* to facilitate communication.

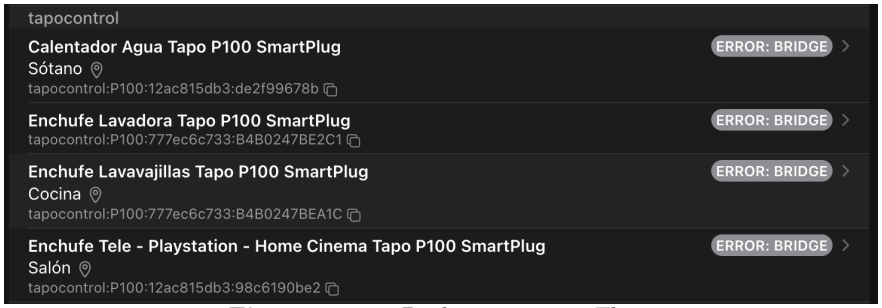


Figure 5.58 – *Bridge error in a Thing*

A **bridge** acts as a gateway between OpenHAB and external devices. If the bridge is not configured correctly, its dependent Things will not function. The **necessary bridges** were added and properly configured for each required Binding, **the most relevant ones will be mentioned**.

5.5.1 OpenWeather Binding Setup

This case involved multiple steps:

1. **Log in to the OpenWeather** platform with a new account.
2. **Choose a subscription plan** (the free plan was selected).

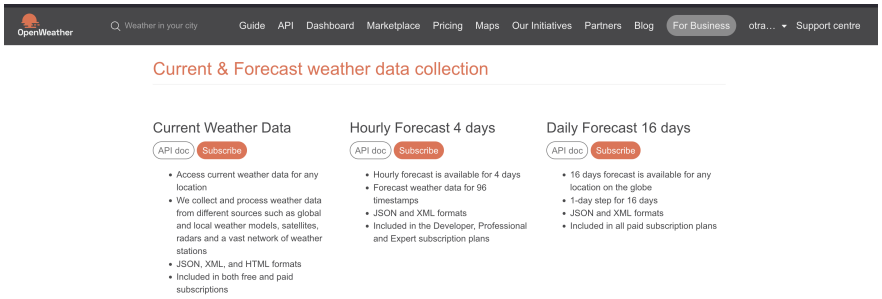


Figure 5.59 – *API subscription options*

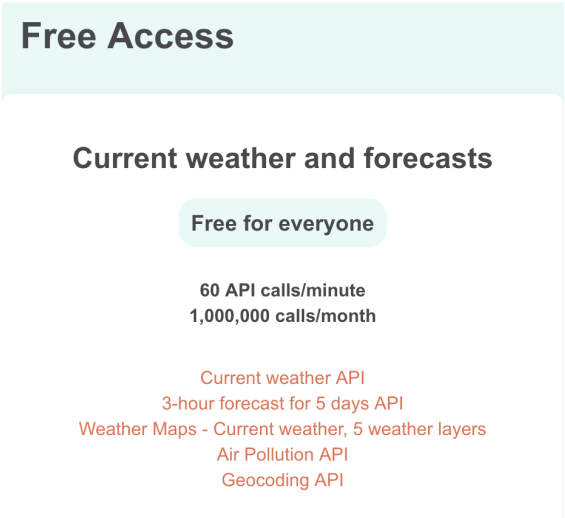


Figure 5.60 – *Free API subscription*

3. **Generate a new API key.**

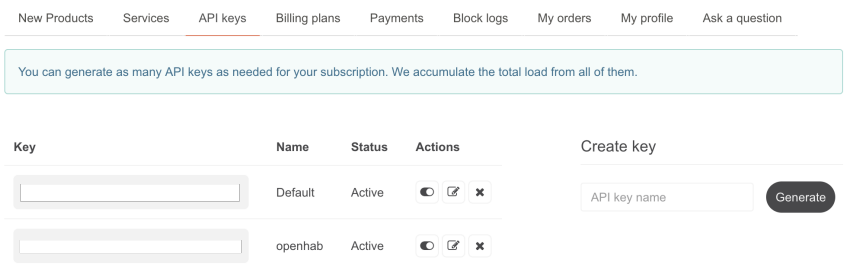


Figure 5.61 – *Generating the OpenWeather API key*

4. In OpenHAB, go to the **Things tab, click the + button, select the OpenWeather binding**, and **create a bridge Thing**.

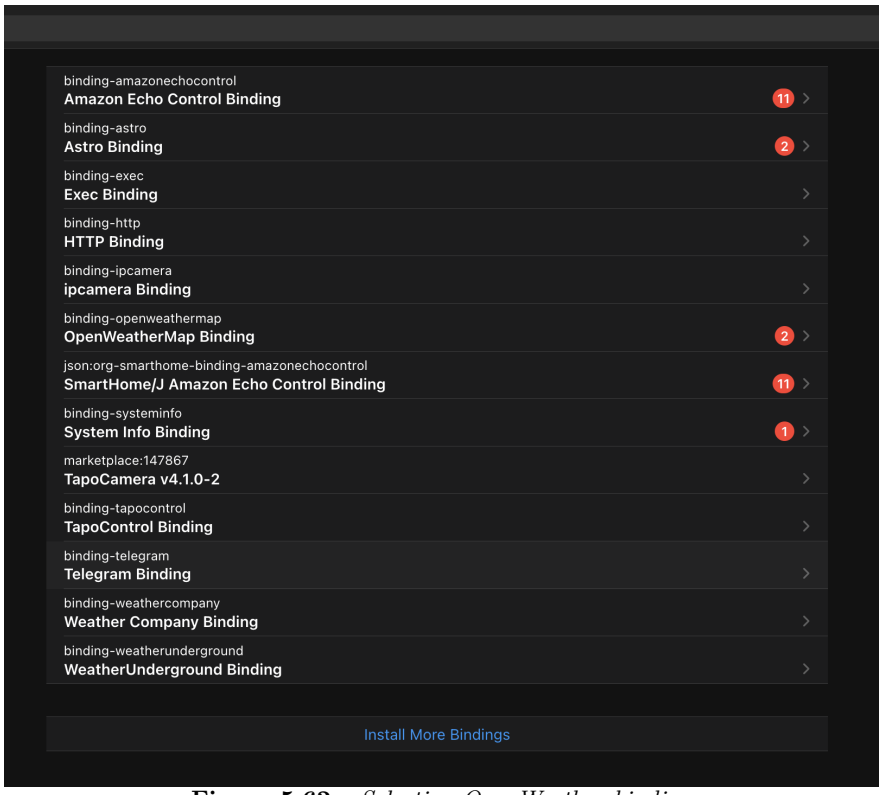


Figure 5.62 – *Selecting OpenWeather binding*

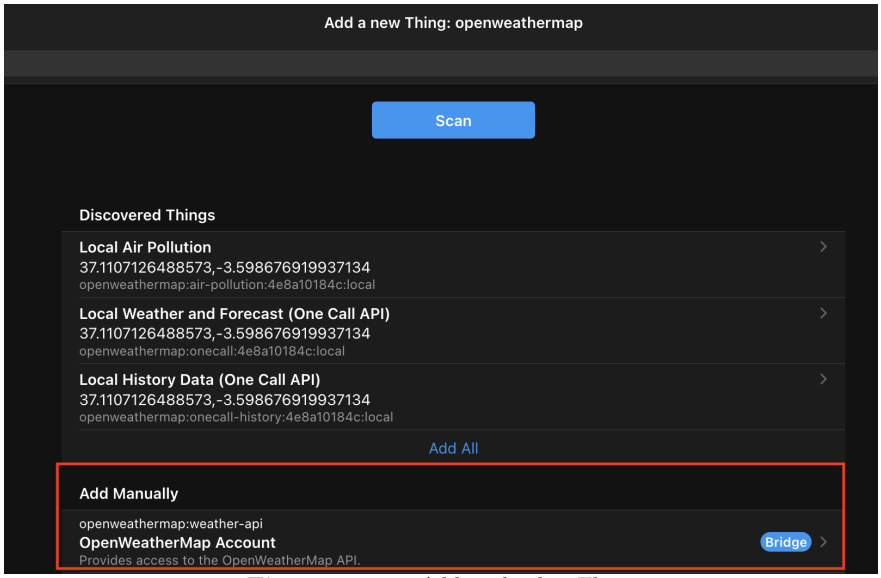


Figure 5.63 – *Adding bridge Thing*

5. **Enter the necessary fields and the API key in Thing coniguration** to finalize bridge creation.

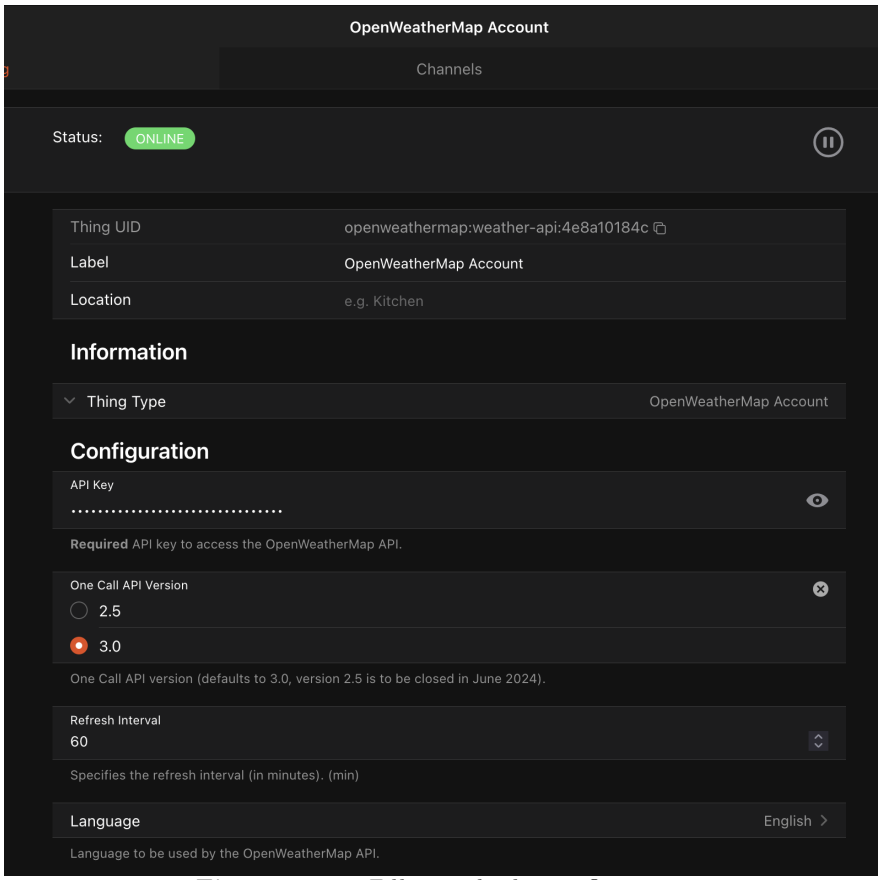


Figure 5.64 – *Filling in bridge configuration*

6. **Assign the newly created bridge to all other OpenWeather Things.**

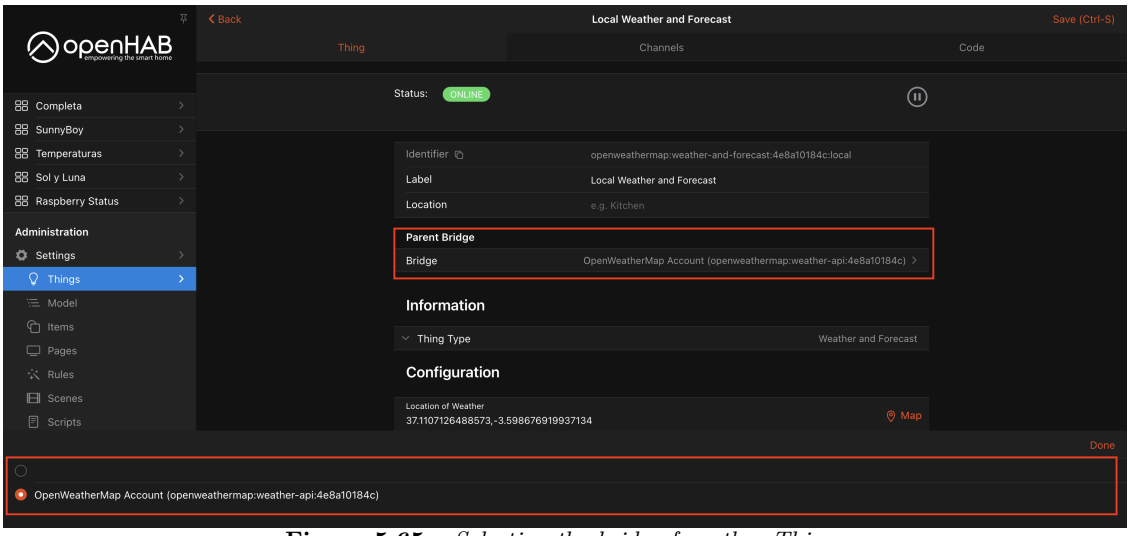


Figure 5.65 – *Selecting the bridge for other Things*

With that configuration, **all Things related to the OpenWeather binding were fixed**.

5.5.2 TP-Link Tapo Control Setup

Setting up the **Tapo bridge** was more straightforward:

1. Add **Thing from Tapo binding** and **select the bridge Thing**.

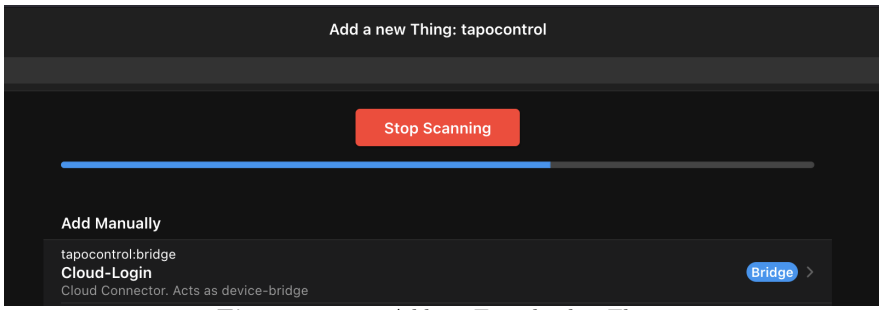


Figure 5.66 – *Adding Tapo bridge Thing*

2. **Fill in the necessary fields in the Thing configuration** (e.g., cloud account credentials to connect to Tapo cloud).

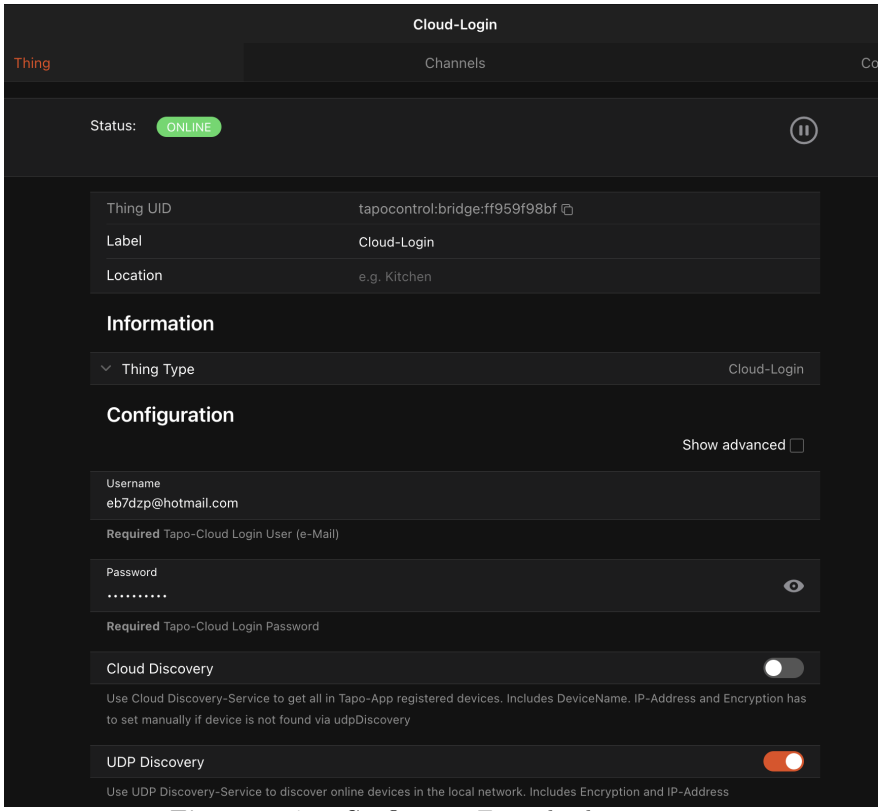


Figure 5.67 – *Configuring Tapo cloud connection*

3. **Assign this bridge to the rest of the Tapo Things.**

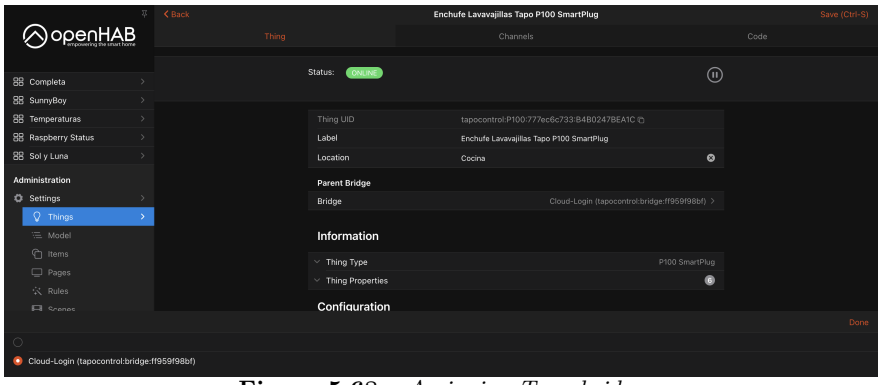


Figure 5.68 – *Assigning Tapo bridge*

Note: The *Tapo integration* was also mentioned in a previous project in section 5.5 [28].

5.5.3 Testing Other Things

After configuring the bridges of these two bindings, other bindings were tested and found to work correctly. Additionally, Items previously not functioning started to generate data as expected.

After **configuring the bridges for these two bindings**, the **remaining bindings were tested** and found to be **working correctly**. Additionally, **Items** that were previously not functioning **began to generate data as expected**.

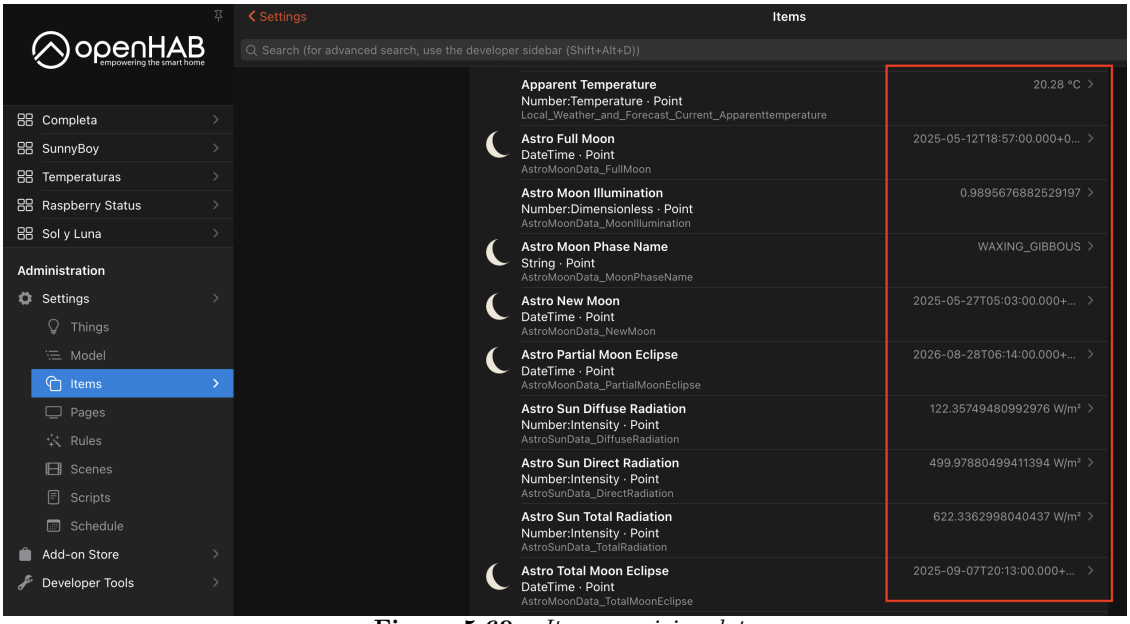


Figure 5.69 – *Items receiving data*

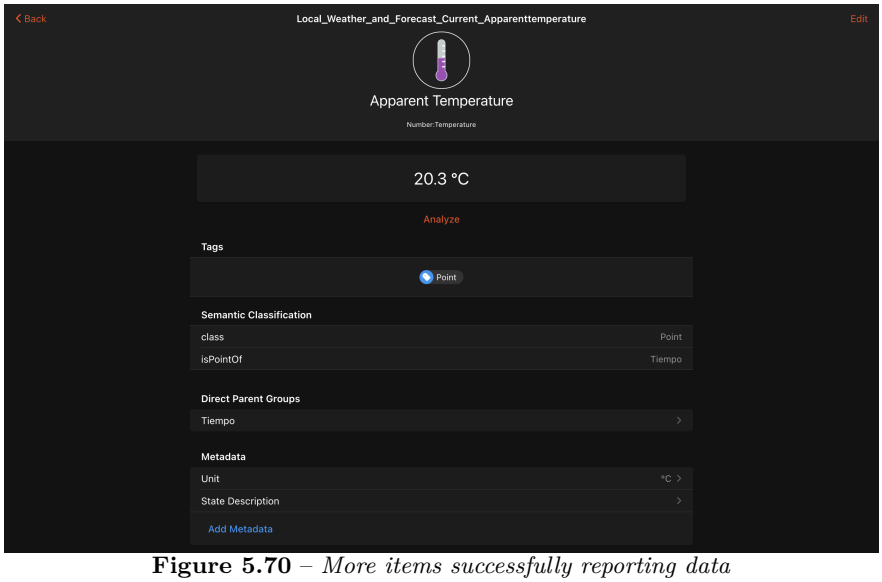


Figure 5.70 – *More items successfully reporting data*

5.6 New Devices

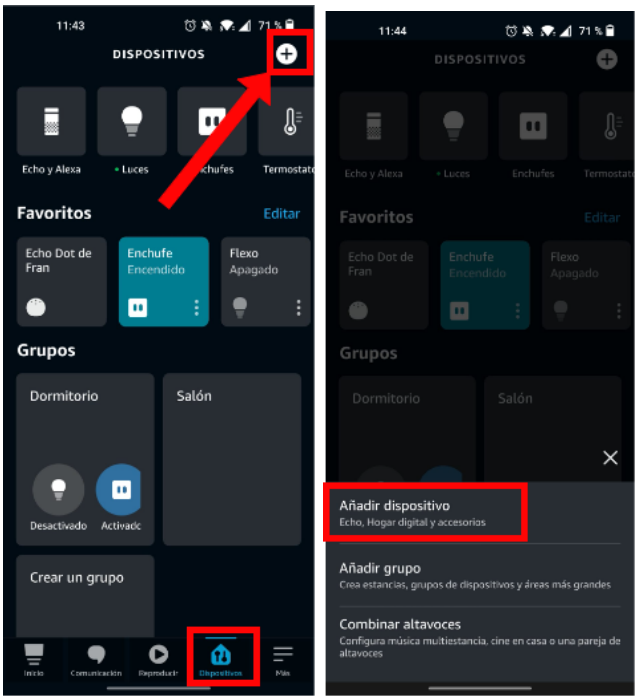
To use the **new smart plugs**, we first had to add them to the **corresponding mobile applications** — Amazon Alexa and TP-Link Tapo — as these plugs are **managed through those apps**, as shown in house design 4.1.

Most of the new devices added were **smart plugs** from either Amazon or Tapo. Since both applications had already been installed and **configured in previous work** (section 5.5 [28]), it was **only necessary to add the new plugs** to the respective apps. No further configuration was required on the application side. All **remaining setup was carried out in OpenHAB**.

5.6.1 Adding Amazon Smart Plugs

Before adding the respective Things, it is necessary to add the new Amazon smart plug in the Amazon Alexa app.

1. Open the **Amazon Alexa** app.
2. Go to the **Devices** tab.
3. Tap the **+ icon** in the top right corner and select ***Add Device***.
4. Choose the ***Plug* device** type.
5. Select the **brand (Amazon)** and follow the on-screen instructions.



(a) **(b)**

Figure 5.71 – *Adding a new device in the Alexa app*

Once the plug is added to Alexa, it will be **automatically discovered** by OpenHAB using the **Amazon Echo Control** binding.

1. In the **Things** tab, click on the **+ icon**.
2. Select the ***Amazon Echo Control* binding**.
3. Click the ***Scan* button** to discover new devices.
4. Once the device appears, click on it and select ***Add as Thing***.

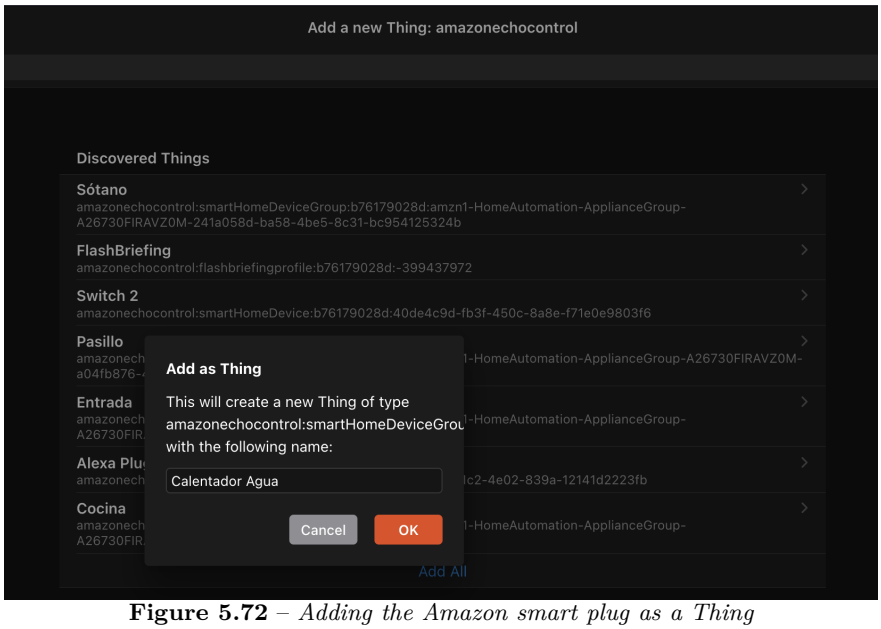


Figure 5.72 – *Adding the Amazon smart plug as a Thing*

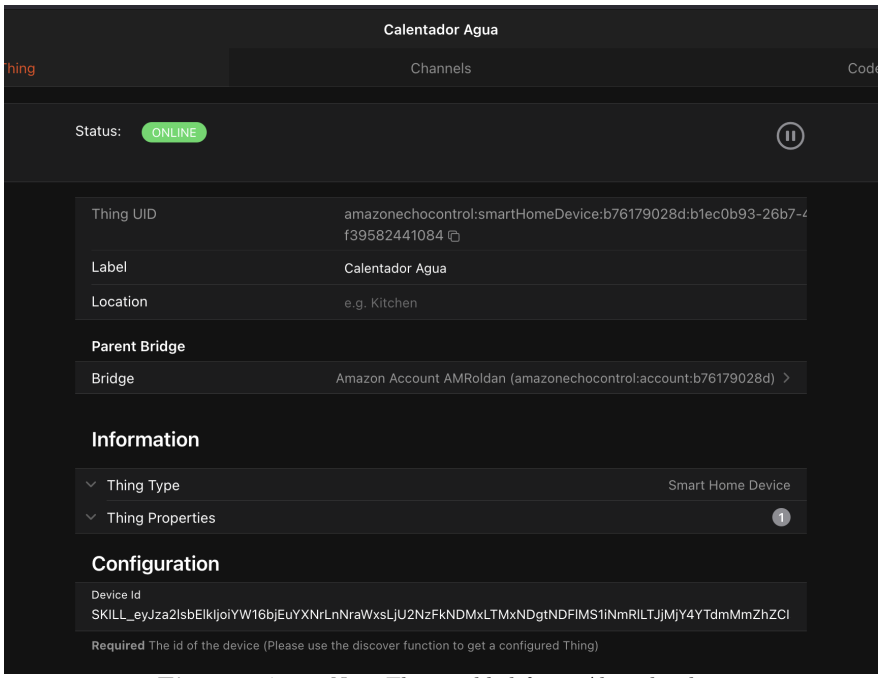


Figure 5.73 – *New Thing added from Alexa binding*

5.6.2 Creating the Corresponding Item

Only Items—not Things—can be controlled. Once the Thing is discovered and added, we create an **Item linked** to its **Switch channel to control the plug** (on/off).

1. **Click on the Thing** in Things tab.
2. Go to the ***Channels* section**.
3. Click ***Add Link to item...***, and then select ***Create new Item***.
4. Use semantic type **Point** and item type **Switch**.

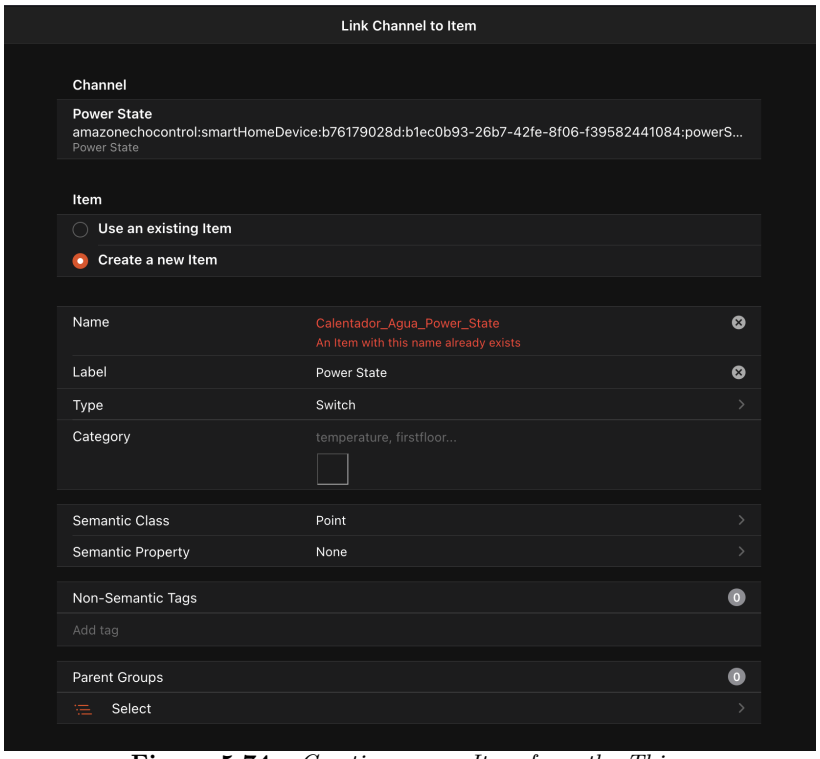


Figure 5.74 – *Creating a new Item from the Thing*

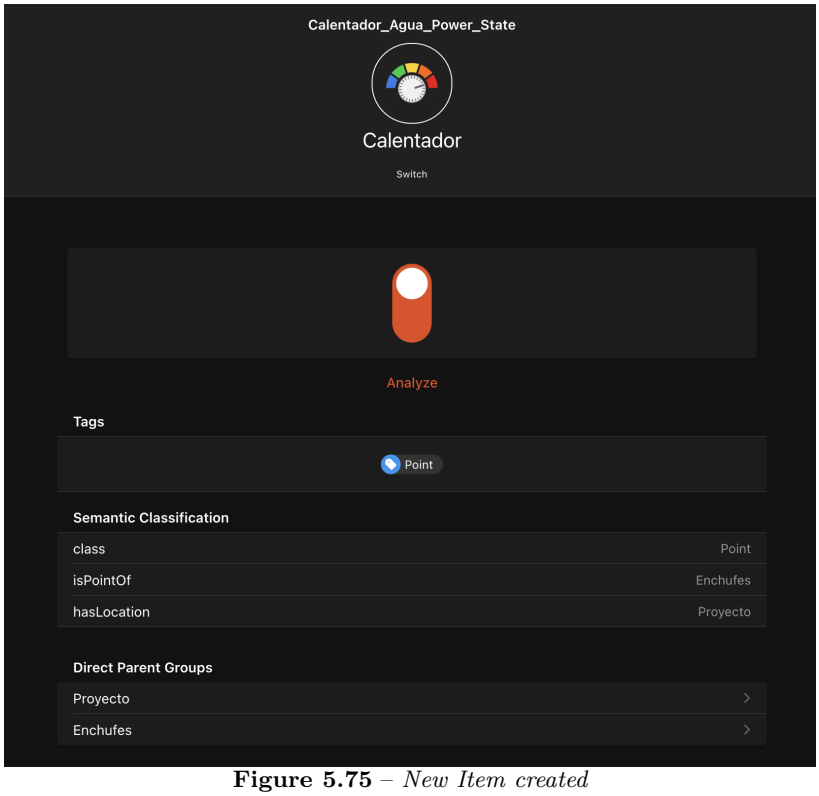


Figure 5.75 – *New Item created*

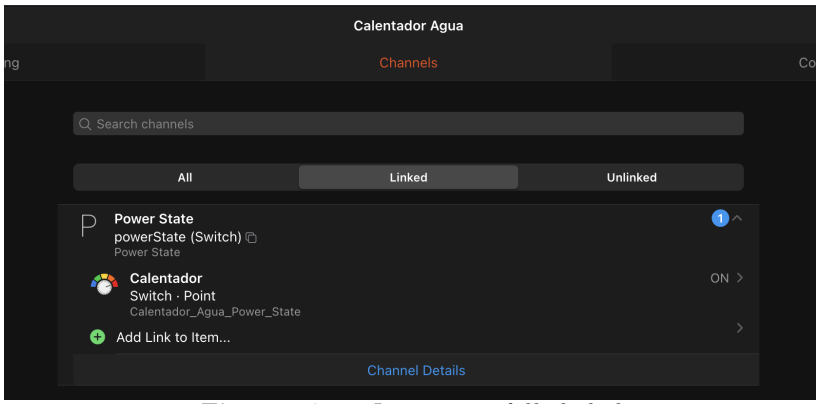


Figure 5.76 – *Item successfully linked*

This process is the same for all Amazon smart plugs.

5.6.3 Adding Tapo Smart Plugs

Similar to the Amazon process, we began by adding the Tapo plugs to the **TP-Link Tapo app**, following the instructions already detailed in Section 5.5.2 of the previous work [28].

Once the Tapo binding and bridge were correctly configured (as explained in Section 5.5.2), the new devices were:

- **Automatically discovered** via a scan, following the same process as adding Amazon plugs.
- **Manually added** if not discovered.

Manual addition steps:

1. In OpenHAB, add a new Thing selecting the Tapo binding.
2. Select the device to add manually.

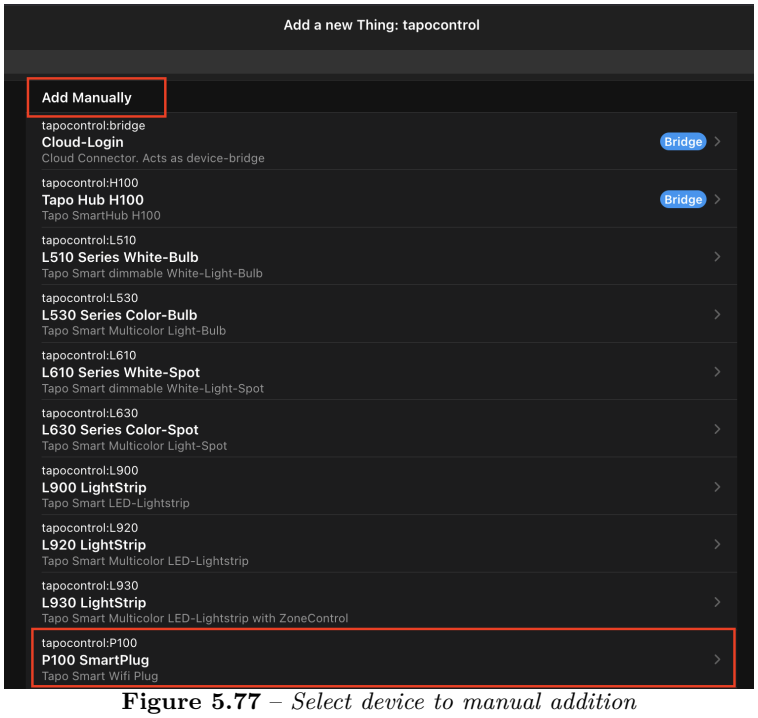


Figure 5.77 – *Select device to manual addition*

3. Select the appropriate **Cloud-Login bridge**.
4. Enter the IP address of the plug.

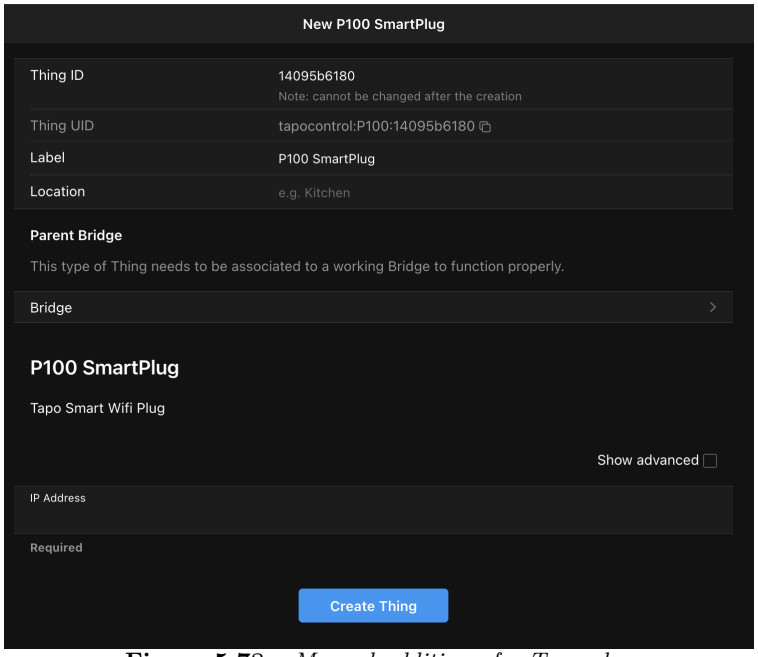


Figure 5.78 – *Manual addition of a Tapo plug*

To find the IP address:

1. Open the Tapo app.
2. Select the device and tap the gear icon in the top-right corner.
3. Go to *Device Information*.
4. The IP and MAC addresses are listed there.

To avoid issues after a power or internet outage, it is recommended to assign a **static IP address** to each plug based on its MAC address as mentioned in section 5.5.2 in the previous project [28].

1. Open a browser and go to `192.168.1.1`.
2. Log in using the default password found on the back of the router.
3. Navigate to: `Menu → Advanced Settings → LAN → Static IP Lease List`.
4. Add the plug's MAC address and the desired IP address.

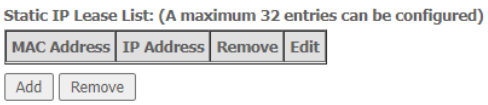


Figure 5.79 – *Assigning a static IP to the plug*

As with the Amazon devices, once the Tapo plug is added as a Thing, an **Item** linked to the **Switch** channel to control the plug's state was created. The process is identical:

- Click on the Thing.
- Go to the *Channels* tab.
- Link to a new **Item** of type **Switch**.

5.7 InfluxDB Persistence Configuration

Due to a system update and a change in the deployment approach, it became necessary to re-establish the InfluxDB **persistence**. Previously, InfluxDB was hosted outside of Docker on the physical host machine. In the current configuration, however, it is deployed inside a Docker container. This change significantly alters how persistence functions and requires additional considerations for **container communication** — particularly between OpenHAB and InfluxDB.

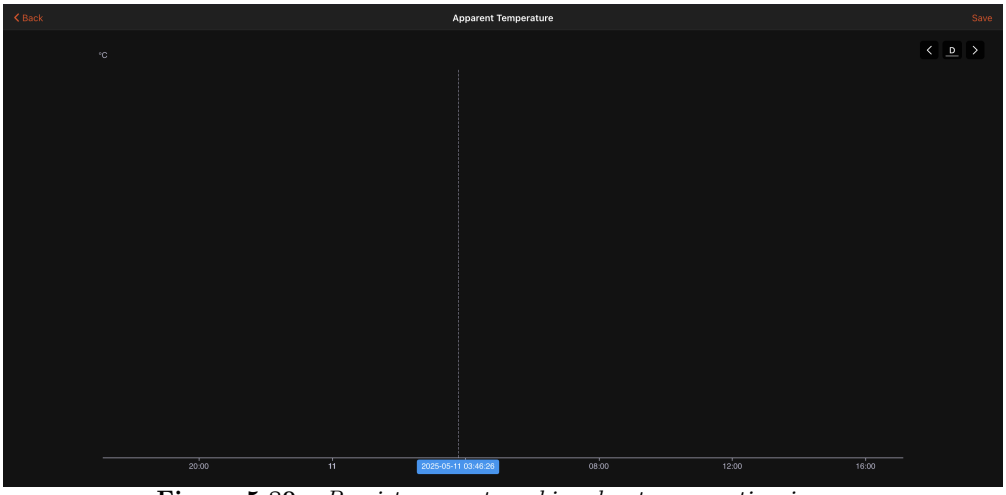


Figure 5.80 – *Persistence not working due to connection issues*

5.7.1 InfluxDB Container

5.7.1.1 Overview of Docker Networks

Docker networks are **virtual layers** that enable **communication between continers** and external systems. They are crucial for **orchestrating** complex applications that consist of multiple interconnected services. Various network drivers are available to cater to different use cases [49]:

- **Bridge**: The default network driver for containers on a single Docker host.
- **Host**: Shares the host's networking stack with the container, removing isolation.
- **Overlay**: Allows networking between continers on different Docker hosts.
- **Macvlan**: Assigns a MAC address to each container, making it behave like a physical device.
- **None**: Disables networking for the container.

By default, Docker Compose creates a single bridge network for all services in a 'docker-compose.yml' file. However, defining custom networks provides **greater control**, particularly for **security** and inter-service communication.

5.7.1.2 Why Define a Custom Network?

In this project, a user-defined **bridge network** named `backend-nw` was created. The main reasons for this decision include:

- **Improved Isolation and Security:** By explicitly placing continers on a private network, only those intentionally attached can communicate. This reduces the risk of unintended exposure and enhances security.
- **Stable and Predictable Networking:** Assigning static IPs ensures that services like OpenHAB can always reach InfluxDB at a known address, avoiding issues caused by dynamic IP allocation.
- **Avoidance of Name Resolution Issues:** Custom networks handle DNS resolution more reliably than Docker's default bridge network, ensuring smoother inter-container communication.
- **Scalability and Modularity:** Custom networks allow logical segmentation of services, making the **architecture** more **maintainable and scalable** as complexity grows.

```
networks:
  backend-nw:
    driver: bridge
    ipam:
     config:
       - subnet: 10.5.0.0/16
         gateway: 10.5.0.1
```

Figure 5.81 – *Custom Docker network defined in* `docker-compose.yml`

The custom network is defined as follows:

- **backend-nw**: The name of the custom Docker network.
- **driver: bridge**: Specifies the use of Docker's bridge networking.
- **ipam**: IP Address Management settings.
- **subnet: 10.5.0.0/16**: Provides a large private IP range.
- **gateway: 10.5.0.1**: Defines the network gateway.

5

5.7.1.3 InfluxDB Container Configuration

The new InfluxDB container was configured with the custom network and a static IP address. This ensures **predictable connectivity** from OpenHAB.

```
influxdb2:
  image: influxdb:latest
  container_name: influxdb2
  restart: always
  volumes:
    - "./data_influxdb2:/var/lib/influxdb2"
    - "./data_influxdb2_config:/etc/influxdb2"
  ports:
    - "8086:8086"
  networks:
    backend-nw:
      ipv4_address: 10.5.0.6
      aliases:
        - db_influxdb2
```

Figure 5.82 – *InfluxDB container configuration with network settings*

Key aspects of the configuration include:

- **Network: backend-nw**: Specifies the previously defined custom Docker network.
- **IPv4 Address: 10.5.0.6**: Sets a static IP address within the `backend-nw` subnet.
- **Aliases: [db_influxdb2]**: Assigns DNS aliases that simplify inter-container communication.

With these configurations, OpenHAB can reliably connect to InfluxDB using a static IP or the hostname `db_influxdb2`. This setup will help resolve the issues shown in Figure 5.80 and ensure that the **persistence layer** remains **stable after updates or restarts**.

5.7.2 Configuring Persistence

Once the InfluxDB container is set up, it is necessary to **configure persistence** in OpenHAB.

Since both OpenHAB and InfluxDB were updated, the **InfluxDB binding** —used to connect OpenHAB to InfluxDB—was reinstalled.

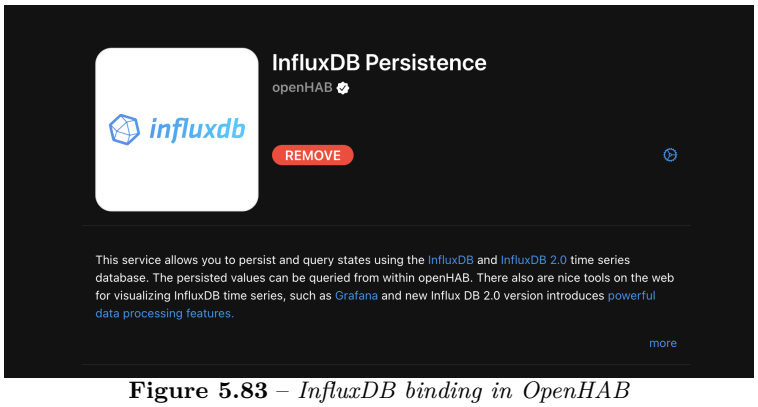


Figure 5.83 – *InfluxDB binding in OpenHAB*

To proceed, the user must navigate to the binding settings and access the **InfluxDB binding configuration** page. Go to **Settings** and **Add-on Settings**.

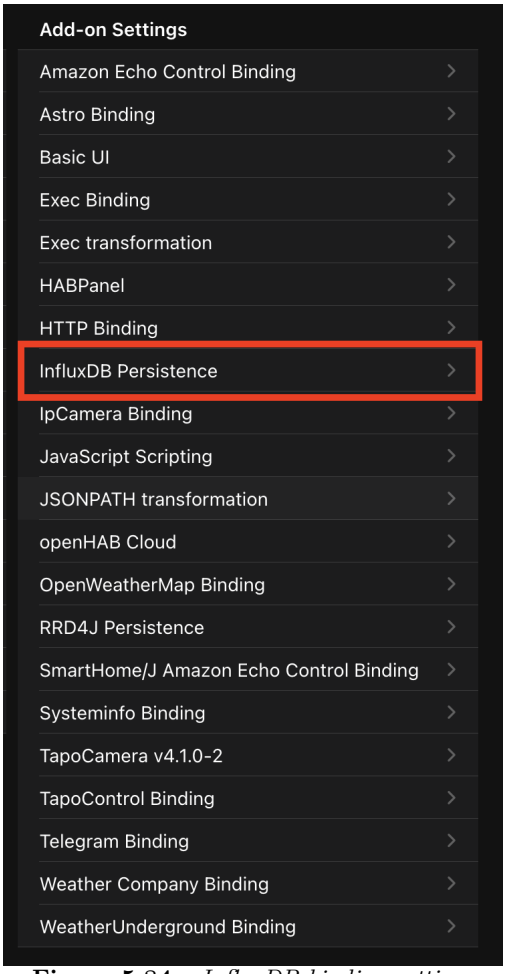


Figure 5.84 – *InfluxDB binding settings*

The following parameters must be configured to establish the **connection** between OpenHAB and InfluxDB:

- **Database URL:** The IP address assigned to the InfluxDB container.
- **InfluxDB Version:** 2
- **Username:** `openhab`
- **Authentication Token:** A valid token generated for access.
- **Organization:** `adminOrg`
- **Bucket:** `openhab_db`

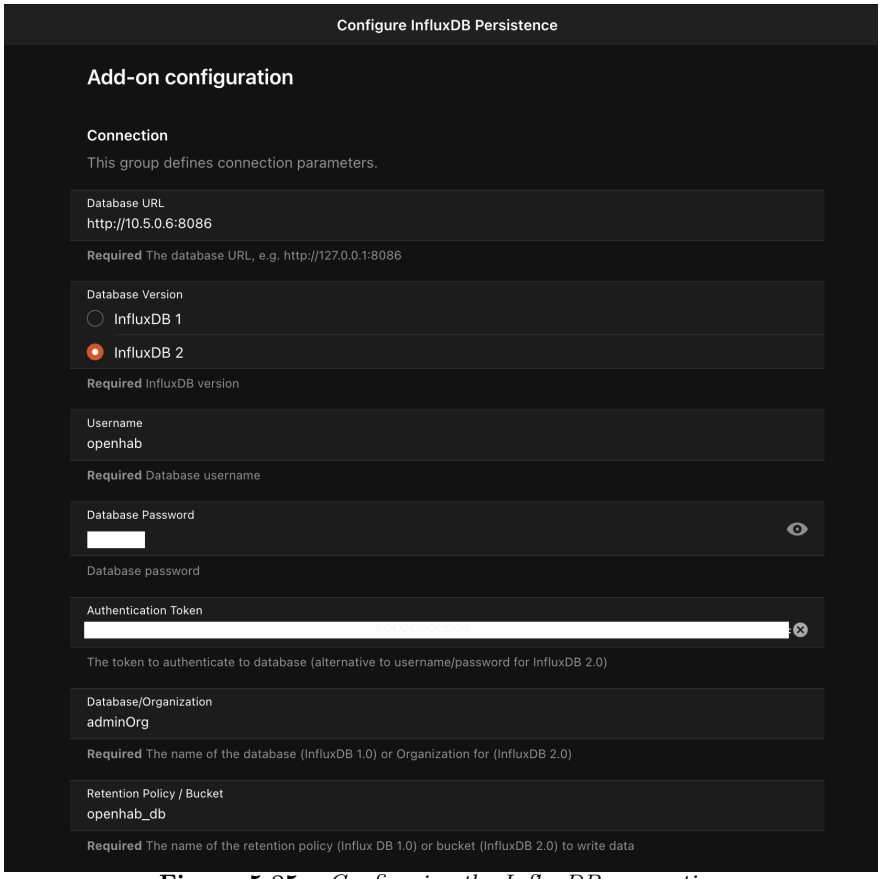


Figure 5.85 – *Configuring the InfluxDB connection*

A **new user** for OpenHAB was created in the InfluxDB database, as shown below. It is important to **associate the user** with the correct **organization**, the one that **contains the `openhab_db` bucket**. This organization is identified by its organization ID:

```
root@92af85a95fe3:/# influx user create -n openhab -p 12345678 -o adminOrg
ID                  Name
0cf588ee40bb7000    openhab
root@92af85a95fe3:/# influx org ls
ID                  Name
09bda1c098e79028    adminOrg
root@92af85a95fe3:/# influx bucket ls
ID                  Name            Retention       Shard group duration    Organization ID         Schema Type
7101eb1d4d15fd9d    _monitoring     168h0m0s        24h0m0s                 09bda1c098e79028        implicit
d191753216ec901c    _tasks          72h0m0s         24h0m0s                 09bda1c098e79028        implicit
e3e1396af83776bf    openhab_db      infinite        168h0m0s                09bda1c098e79028        implicit
root@92af85a95fe3:/#
```

Figure 5.86 – *New user associated with existing bucket*

5.7.2.1 Creating an Authentication Token

The **token used for authentication** was created using the `influx auth create` command. This command specifies the **organization** (`adminOrg`), the **username** (`openhab`), and grants the **necessary permissions**.

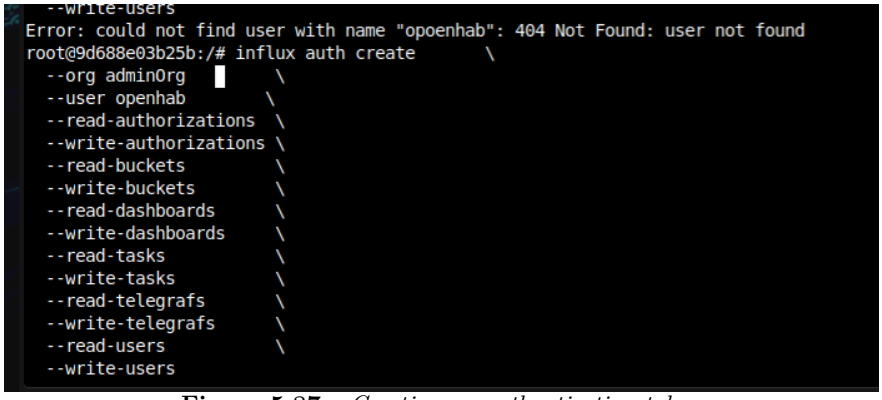


Figure 5.87 – *Creating an authentication token*

5.7.2.2 Creating a Remote Connection

To enable **remote interaction** with the **token**, a remote connection was created using the `influx remote` command. A name was assigned to the remote, along with its URL. Since the token resides in the current InfluxDB instance, the URL is set to `localhost`. Additionally, the ID of the organization in which the token resides must be provided.

Figure 5.88 – *Creating a remote connection to InfluxDB*

5.7.2.3 Defining Persistence Policies

After successfully configuring the connection, **persistence policies** must be defined. These policies determine which **items** should be persisted and **how frequently** their data is stored.

In this case, **all items** are selected for persistence. This includes both existing and newly added items. Consequently, their historical data will be stored in InfluxDB and made visible through OpenHAB.

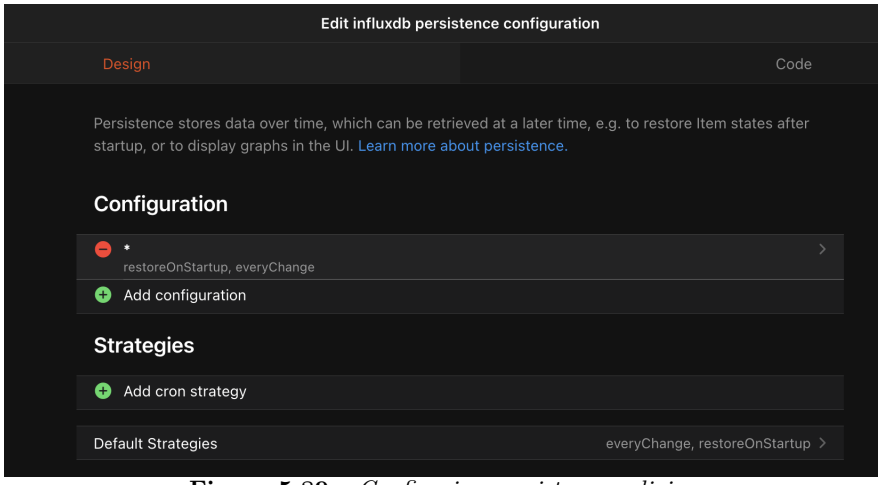


Figure 5.89 – *Configuring persistence policies*

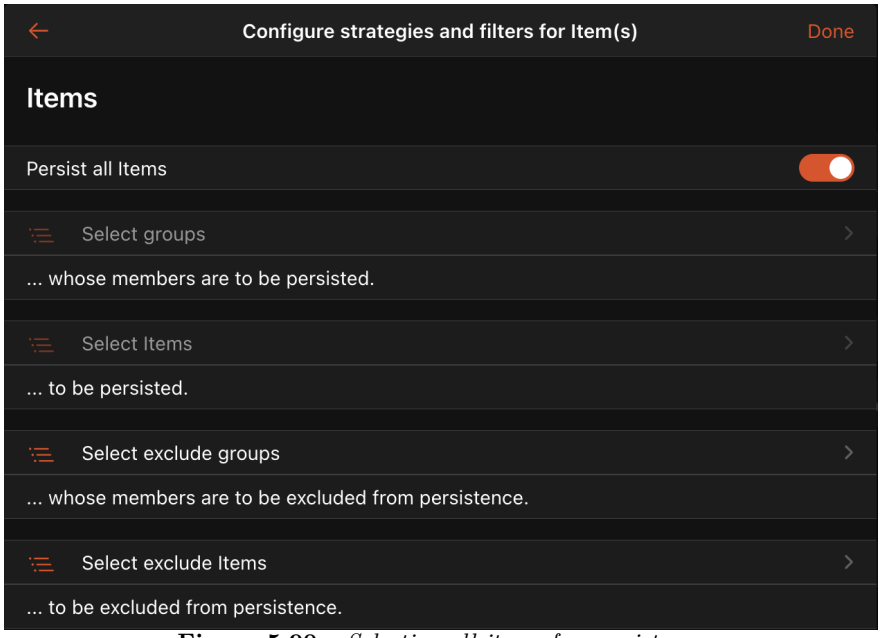


Figure 5.90 – *Selecting all items for persistence*

Finally, once everything is properly configured, **persistence works** as expected, and item data is successfully stored in InfluxDB.

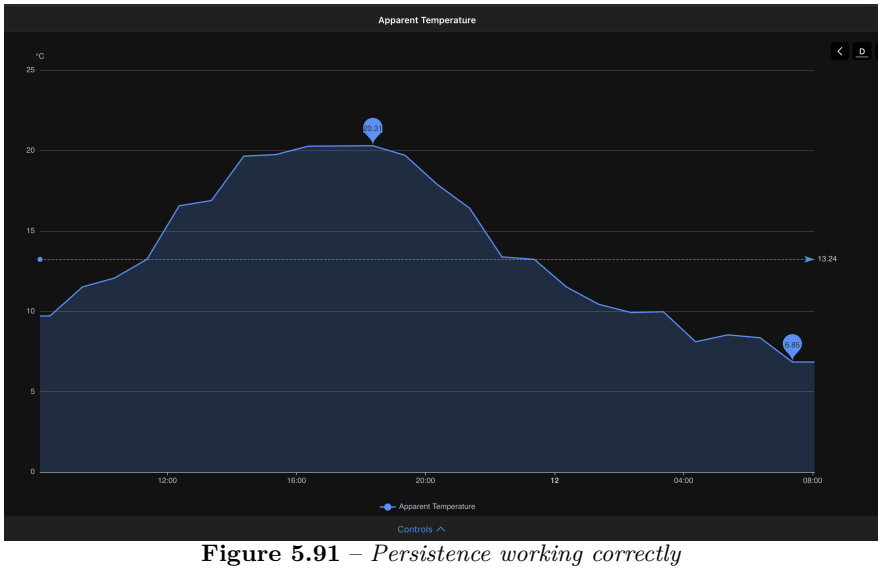


Figure 5.91 – *Persistence working correctly*

5.8 Fix and Understand Connectivity with Solar Inverter and Home Manager

This section builds upon the explanation provided in **Section 5.9 of the previous work** [28]. The data is retrieved from two key sources: the SMA Sunny Boy 3.0 solar inverter and the SMA Energy Meter.

In order to assign and represent this retrieved data appropriately, **corresponding items** were defined using textual configuration, as described in Section 5.9.3 of the previous project.

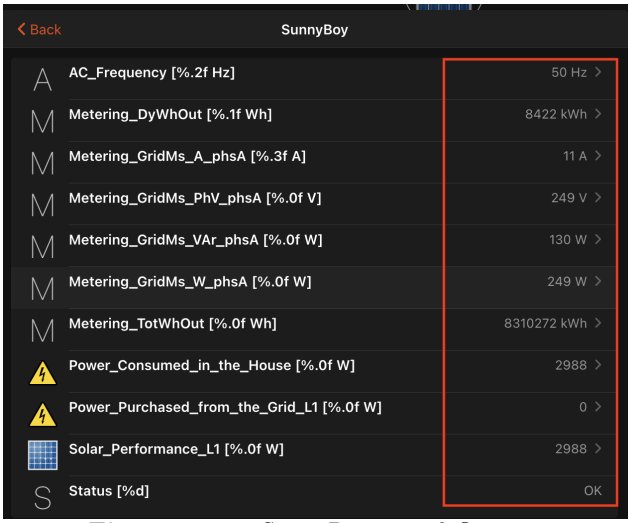


Figure 5.92 – *SunnyBoy item definitions*

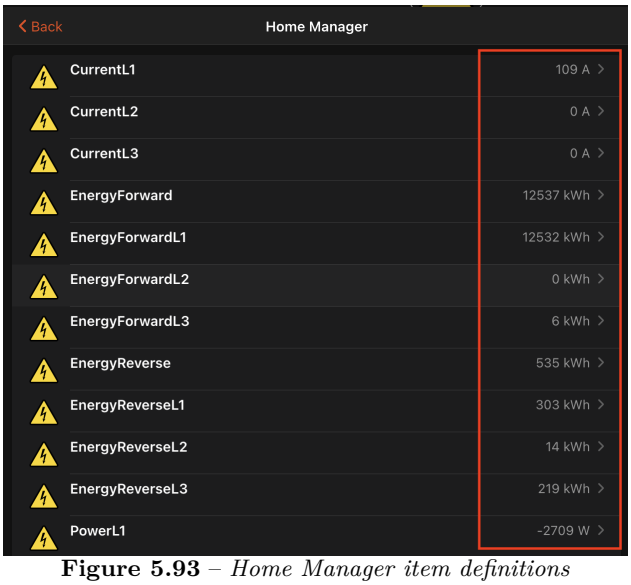


Figure 5.93 – *Home Manager item definitions*

The data stored in these **Items** is retrieved using **two Python scripts**. To execute these scripts, the Dockerfile was configured as previously explained in Section 5.3.2.1. The main purpose of the additional Dockerfile layers is to ensure the correct execution of these scripts by installing the **necessary libraries** and configuring the **environment variables** required to host them.

Figure 5.94 – *Running SunnyBoy and Home Manager scripts in Docker*

A Python **virtual environment** (venv) was created to install the required dependencies. This resolved environment-related errors, shown below, by isolating the dependencies from the host system environment.

```
 => ERROR [3/3] RUN python3 -m pip install setuptools wheel
------
 > [3/3] RUN python3 -m pip install setuptools wheel:
0.245 error: externally-managed-environment
0.245
0.245 × This environment is externally managed
0.245 ╰─> To install Python packages system-wide, try apt install
0.245     python3-xyz, where xyz is the package you are trying to
0.245     install.
0.245
0.245     If you wish to install a non-Debian-packaged Python package,
0.245     create a virtual environment using python3 -m venv path/to/venv.
0.245     Then use path/to/venv/bin/python and path/to/venv/bin/pip. Make
0.245     sure you have python3-full installed.
0.245
0.245     If you wish to install a non-Debian packaged Python application,
0.245     it may be easiest to use pipx install xyz, which will manage a
0.245     virtual environment for you. Make sure you have pipx installed.
0.245
0.245     See /usr/share/doc/python3.11/README.venv for more information.
0.245
```

Figure 5.95 – *Environment error resolved using virtual environment*

Although the scripts successfully retrieve data from the Sunny system devices, Rules are used to publish this data into the defined OpenHAB items. The rules are further detailed in the following section.

The scripts are triggered automatically via OpenHAB rules—one for the SunnyBoy inverter and another for the Home Manager. Each rule executes its respective script and **updates the items** using bindings that parse the JSON output of the scripts in to the items. The rules are configured to run every minute.

```
#!/bin/bash

RESULT=$(python3 /openhab/conf/scripts/HomeManager20.py)

echo "$RESULT"
```

Figure 5.96 – *Shell script used to retrieve Home Manager data*

```
rule "ejecutar homemanager"

when Time cron "0 * * * * ? *"

then
    var HM_JSON_Out = executeCommandLine(Duration.ofSeconds(40), "/openhab/conf/scripts/home_manager.sh")
    logInfo("HM_JSON_Out",HM_JSON_Out)
    PowerTotal.postUpdate(Float::parseFloat(transform("JSONPATH", "$.PowerTotal", HM_JSON_Out)))
    PowerL1.postUpdate(Float::parseFloat(transform("JSONPATH", "$.PowerL1", HM_JSON_Out)))
    PowerL2.postUpdate(Float::parseFloat(transform("JSONPATH", "$.PowerL2", HM_JSON_Out)))
    PowerL3.postUpdate(Float::parseFloat(transform("JSONPATH", "$.PowerL3", HM_JSON_Out)))
    CurrentL1.postUpdate(Float::parseFloat(transform("JSONPATH", "$.CurrentL1", HM_JSON_Out)))
    CurrentL2.postUpdate(Float::parseFloat(transform("JSONPATH", "$.CurrentL2", HM_JSON_Out)))
    CurrentL3.postUpdate(Float::parseFloat(transform("JSONPATH", "$.CurrentL3", HM_JSON_Out)))
    VoltageL1.postUpdate(Float::parseFloat(transform("JSONPATH", "$.VoltageL1", HM_JSON_Out)))
    VoltageL2.postUpdate(Float::parseFloat(transform("JSONPATH", "$.VoltageL2", HM_JSON_Out)))
    VoltageL3.postUpdate(Float::parseFloat(transform("JSONPATH", "$.VoltageL3", HM_JSON_Out)))
    EnergyForward.postUpdate(Float::parseFloat(transform("JSONPATH", "$.EnergyForward", HM_JSON_Out)))
    EnergyForwardL1.postUpdate(Float::parseFloat(transform("JSONPATH", "$.EnergyForwardL1", HM_JSON_Out)))
    EnergyForwardL2.postUpdate(Float::parseFloat(transform("JSONPATH", "$.EnergyForwardL2", HM_JSON_Out)))
    EnergyForwardL3.postUpdate(Float::parseFloat(transform("JSONPATH", "$.EnergyForwardL3", HM_JSON_Out)))
    EnergyReverse.postUpdate(Float::parseFloat(transform("JSONPATH", "$.EnergyReverse", HM_JSON_Out)))
    EnergyReverseL1.postUpdate(Float::parseFloat(transform("JSONPATH", "$.EnergyReverseL1", HM_JSON_Out)))
    EnergyReverseL2.postUpdate(Float::parseFloat(transform("JSONPATH", "$.EnergyReverseL2", HM_JSON_Out)))
    EnergyReverseL3.postUpdate(Float::parseFloat(transform("JSONPATH", "$.EnergyReverseL3", HM_JSON_Out)))
end
```

Figure 5.97 – *Rule for processing and updating Home Manager data*

In summary, this section focused on **fixing the Dockerfile configuration**, enabling the **scripts** to function properly. As a result, **item updates** were successfully triggered through **OpenHAB rules**. Understanding this integration is essential for developing a **user interface** to display information from both the SunnyBoy inverter and the Home Manager.

5.9 Telegram and WhatsApp Notifications

5.9.1 Telegram Integration

Telegram notifications were implemented for various system components, including the SunnyBoy inverter, outdoor irrigation, water heater, and lighting. These notifications are managed via **OpenHAB rules**. In addition, **time-based automation** rules were created to control lights and irrigation systems based on predefined schedules.

Furthermore, **conversational interaction** with OpenHAB bot was enabled through the Telegram bot, allowing the user to control the water heater, outdoor irrigation, and lighting using quick replies.

5.9.1.1 Telegram Bot Configuration Steps

The first step in enabling Telegram notifications was to create and configure a Telegram bot. The process consisted of the following steps:

1. **Create a New Bot and Obtain the Token**

   Start a conversation with **BotFather** on Telegram.

   - Send the commands **`/start`** and **`/newbot`**.
   - Follow the instructions to choose a **name** and **username** for the bot.
   - **BotFather** will return a unique **authentication token** required for later configuration steps.

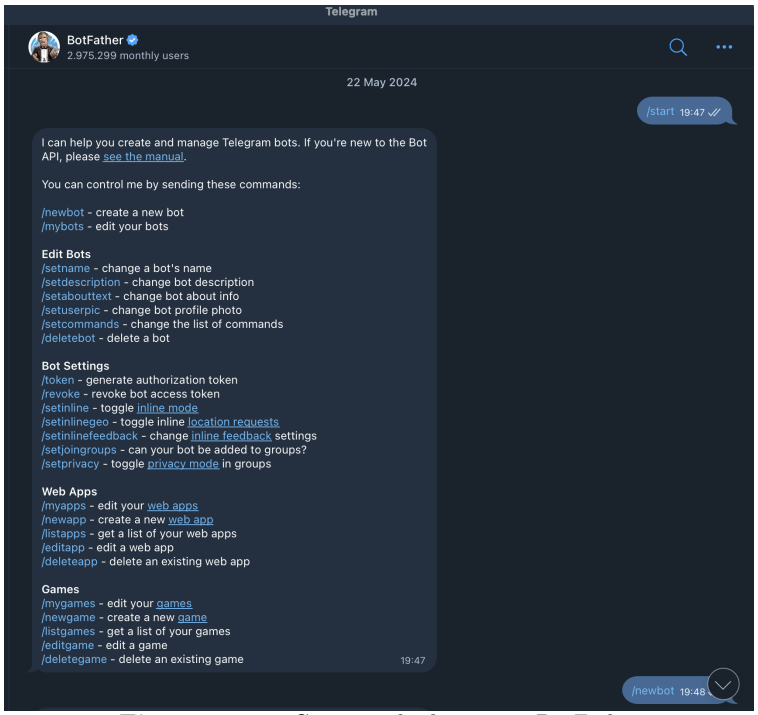


Figure 5.98 – *Creating the bot using BotFather*

Figure 5.99 – *BotFather response with bot token*

2. **Create the Destination Chat**

   Initiate a chat with the newly created Telegram bot and `/start` followed by any message. This step is mandatory; the next API-based steps will not work until the bot has **received at least one message** from the user.

3. **Retrieve the Chat ID**

   There are two ways to **obtain the chat ID** required for sending messages:

   - Open a web browser and navigate to the following URL, replacing `<token>` with your bot's token:
     `https://api.telegram.org/bot<token>/getUpdates`
     In the resulting JSON response, locate the `id` field to obtain the chat ID.

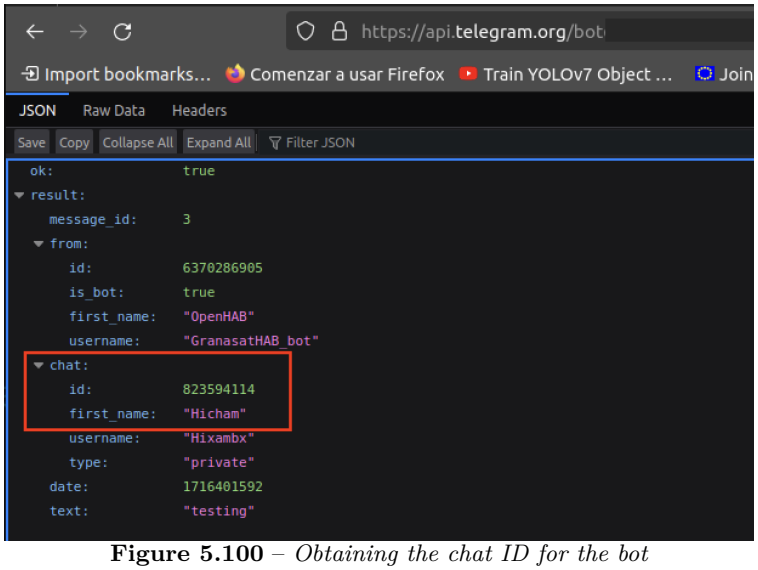


Figure 5.100 – *Obtaining the chat ID for the bot*

   - Alternatively, interact with the Telegram bot `@myidbot` by sending `/start`. The bot will return your chat ID.

4. **Test the Bot**

   To verify the bot configuration, send a test message via the Telegram API:

   - Open the following URL in a browser, replacing `<token>` with the bot token and `<chatId>` with the retrieved chat ID:

     `https://api.telegram.org/bot<token>/sendMessage?chat_id=<chatId>&text=testing`
   - If the configuration is correct, the bot will deliver a message containing the text "testing" to the specified chat.

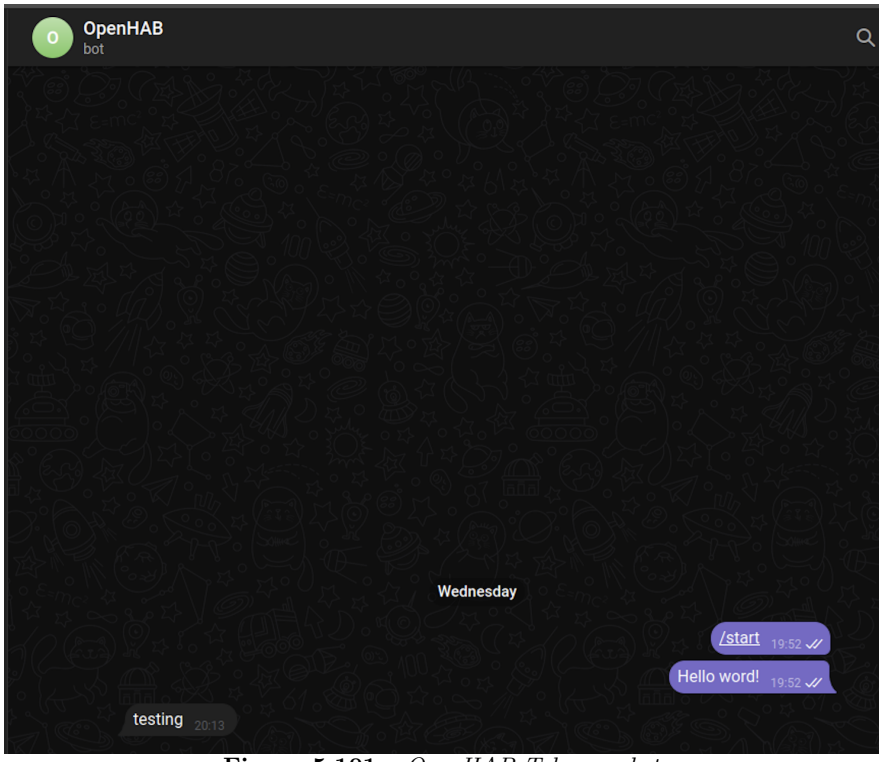


Figure 5.101 – *OpenHAB Telegram bot*

5.9.2 Telegram Binding: Things and Items

Once the Telegram bot was created and successfully tested, the next step was to install the Telegram binding in OpenHAB, create the corresponding **Thing**, and define the necessary **Items** to enable **interaction with the bot**. These Items are used throughout the automation rules to handle Telegram notifications.

1. **Install the Telegram Binding**

   The Telegram binding was installed via the OpenHAB UI.

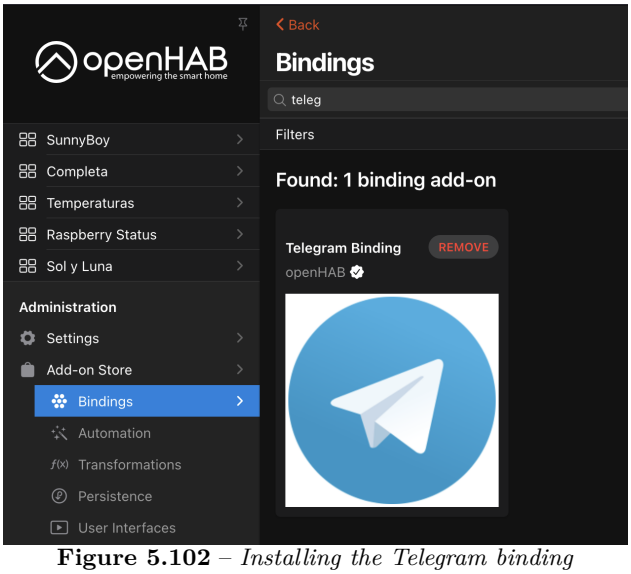


Figure 5.102 – *Installing the Telegram binding*

2. **Create the Telegram bot Thing**

   A new Thing was created using the Telegram binding. The required fields such as the **bot token** and the allowed **chat IDs** were filled in. All other settings were left with their default values.

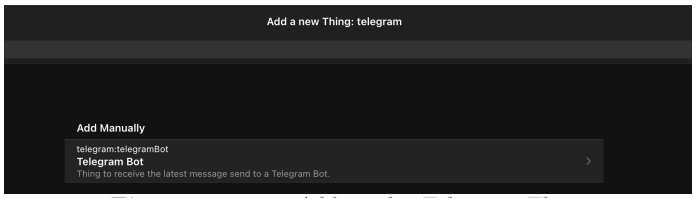


Figure 5.103 – *Adding the Telegram Thing*

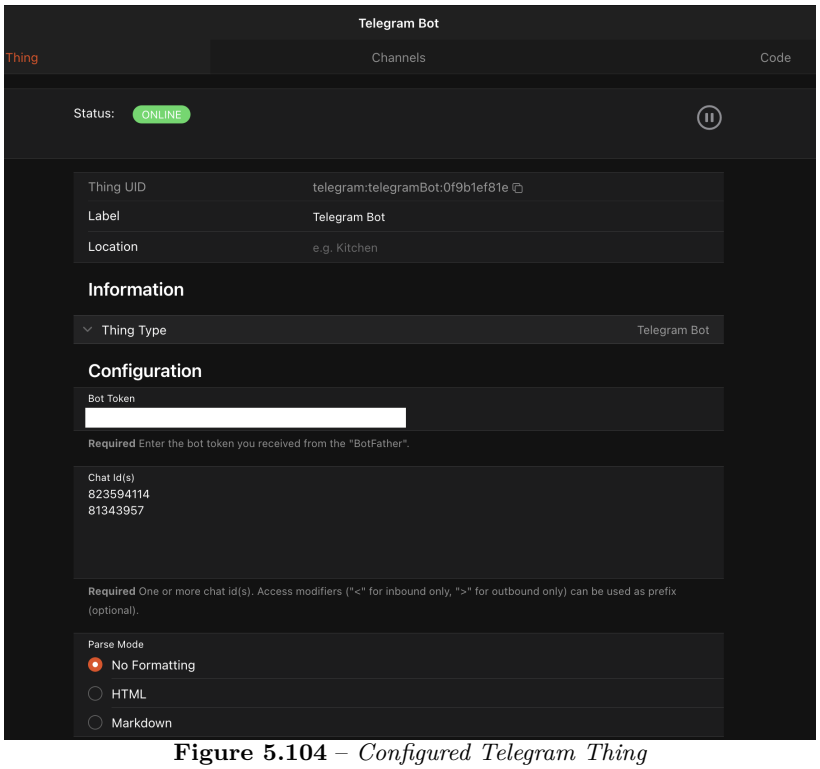


Figure 5.104 – *Configured Telegram Thing*

3. **Create Essential Items**

   After the **Thing** was created, relevant **channels** were linked to newly defined **Items**. These Items allow **rules** to **interact with messages** sent and received by the OpenHAB Telegram bot.

   (a) **Last Message Text** — Represents the most recent message received by the bot.

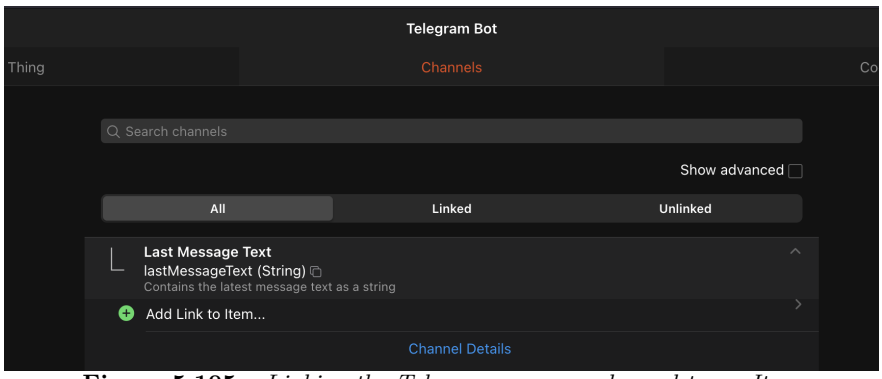


Figure 5.105 – *Linking the Telegram message channel to an Item*

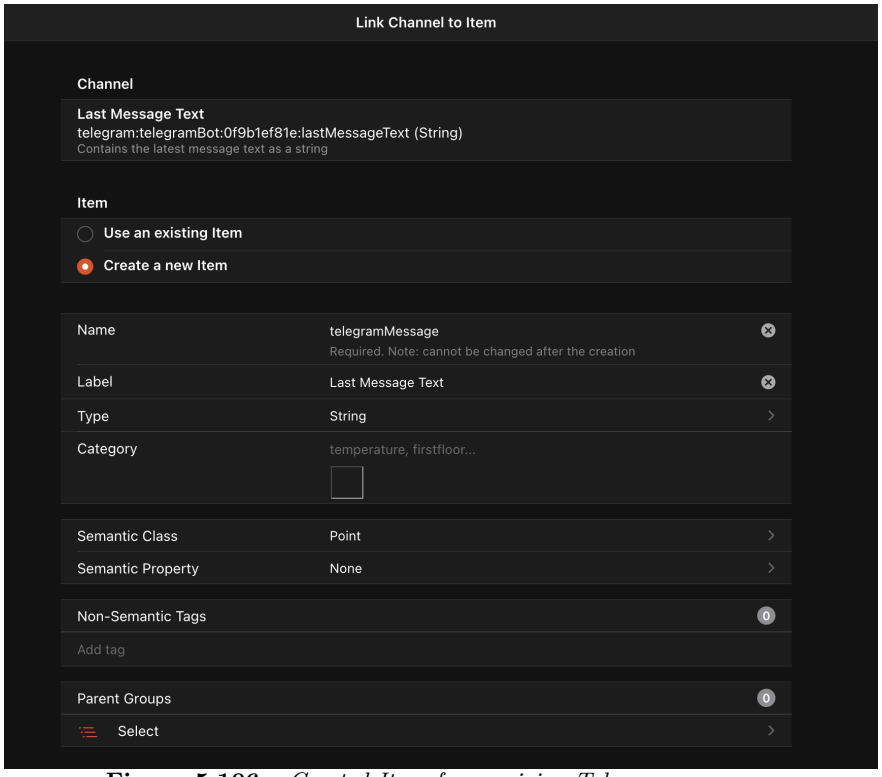


Figure 5.106 – *Created Item for receiving Telegram messages*

   (b) **Reply ID** — This channel contains the ID of the reply passed to `sendTelegram()` as `replyId`. It allows unambiguous assignment of user replies to bot messages. This will be used in later sections.

   (c) **Last Message Name** — Stores the full name of the sender of the most recently received message.

The final result was a set of Telegram-related **Items** prepared for use in **rules** and automation logic.

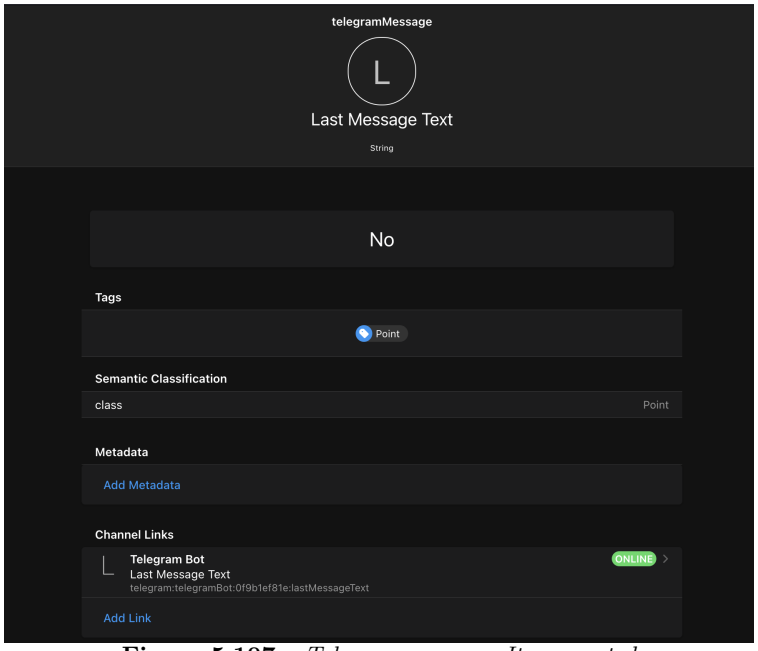


Figure 5.107 – *Telegram message Item created*

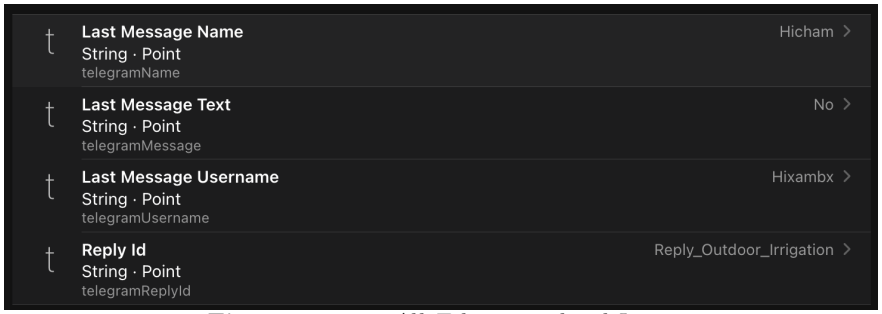


Figure 5.108 – *All Telegram-related Items*

5.9.3 Telegram Rules and automations

As previously mentioned, **notifications** are configured using Rules. However, these rules are not limited to notifications; they also include **automation logic** for controlling devices in the smart home. Since many of these automations are closely related to notification logic, they are included in this section.

Rules in OpenHAB are user-defined pieces of logic that **automate behavior** based on **specific events** (e.g., a motion sensor detecting movement or a switch being activated). When triggered, these rules perform **defined actions**, such as turning on lights or sending Telegram notifications.

- **Rule Implementation in Files**

  In this project Rules are implemented using **files** with the `.rules` extension. Each file can contain multiple rules, and the OpenHAB system supports multiple such files. Although it is also possible to define rules using the graphical user interface (UI), file-based rule definitions offer greater flexibility and are preferred in this project. All rule files are stored in the directory: `openhab/conf/rules`.

```java
rule "<RULE_NAME>"
when
    <TRIGGER_CONDITION> [or <TRIGGER_CONDITION2> [or ...]]
then
    <SCRIPT_BLOCK>
end
```

Figure 5.109 – *Structure of rule files*

- **Rule Language: DSL**

  Rules are written in a Domain-Specific Language (DSL) provided by OpenHAB. This language is based on Xtend, which is similar in syntax to Java. It is designed specifically for writing automation logic within the OpenHAB environment.

  At the beginning of each rule file, users can include **import statements** and declare **global variables**. These must appear at the top of the file to ensure proper execution of the rules.

- **Structure of a Rule**

  A rule in DSL consists of the following components:

  – `<RULE-NAME>` — A unique name for each rule, enclosed in quotation marks.
  – `<TRIGGER-CONDITION>` — The triggering event upon which the rule logic is executed. A rule is executed in reaction to one or more trigger conditions. Multiple conditions are separated by the keyword `or`.
  – `<ACTION-SCRIPT-BLOCK>` — The logic block that defines what actions to perform when the trigger is activated. These actions can include device control, messaging, logging, etc.

- **Trigger Types**

  A rule executes in response to trigger events. The following types of triggers are used in this project:

  – **Item Triggers** — Triggered by commands, updates, or state changes of Items.
  – **Time Triggers** — Executed at specific times or intervals.
  – **Channel Triggers** — Triggered directly by Thing Channels instead of linked Items.

5

5.9.3.1 Sunny Boy Status Notifications

To monitor the status of the Sunny Boy inverter, the item `SMA_STATUS` is used. This item is updated by the rule explained in Section 5.8. Its state change serves as the **trigger** for the notification rule.

```
rule "SMA energy meter connection error"
when
   Item SMA_STATUS changed to OFF
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    telegramAction.sendTelegram("No hay conectividad con el SunnyBoy. Revisar el magnetotermico y el diferencial y que las luces del inversor estan encendidos")
end

rule "SMA energy meter is running again"
when
    Item SMA_STATUS changed to ON
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    telegramAction.sendTelegram("Sunny boy ya esta funcionando nuevamente")
end
```

Figure 5.110 – *Rule for Sunny Boy status notification*

When the item `SMA_STATUS` changes to `OFF`, indicating a loss of connection with the Sunny Boy inverter, a Telegram **notification** is sent through OpenHAB Telegram bot to inform the user of the issue. The message suggests verifying the system's connectivity to ensure everything is functioning properly. Conversely, when the status returns to `ON`, a confirmation message is sent indicating that the connection has been re-established.

To send messages via Telegram, the OpenHAB Telegram binding is used. The function `getActions(binding,thing_uid)` retrieves the Telegram action interface bound to the specified thing

(i.e., the Telegram bot), and the `sendTelegram` method sends the actual message to the configured chat IDs added to the thing (see section 5.9.2).

The **results** and effectiveness of these notifications are further discussed in the **testing section**.

5.9.3.2 Water Heater Notifications

The purpose is to inform the user when the water heater is turned `ON` or `OFF`, and to allow the user to interact with the system via Telegram bot to **control its state remotely**.

The status of the water heater is tracked using an **item** that was previously defined and is used to toggle the device on or off 5.6.2.

```
rule "Water heater OFF"
when
    Item Calentador_Agua_Power_State changed to OFF
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    telegramAction.sendTelegram("El calentador de agua ha sido apagado")
end


rule "Water heater ON"
when
    Item Calentador_Agua_Power_State changed to ON
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    telegramAction.sendTelegramQuery("El caletador de agua ha sido encendido, desea apagarlo?", "Reply_Water_Heater", "Yes", "No")
end
```

Figure 5.111 – *Water heater ON/OFF notification rule*

To avoid **unnecessary use** of the water heater, such as being turned on accidentally or left on, OpenHAB bot sends a Telegram message when the device is switched on. This message provides an **interactive prompt**, asking whether the user wants to **turn it off or keep it on**. The user can respond directly within Telegram bot without needing to access the web interface or mobile app.

The communication with Telegram bot is handled through the Telegram binding. Below are the **key components** that make this **interaction** possible:

1. The `getActions(binding, thingUID)` function is used to retrieve the **Telegram action object** from the configured **Telegram bot Thing**.
2. A message with reply options is sent to the user using `sendTelegramQuery(message, replyId, replyOptions[...])`

   In this case it creates a message with two reply options: `Sí` (Yes) and `No`, and assigns the identifier `Reply_Water_Heater` to the reply.
3. The identity of the user replying to the query is determined using the **item** `telegramName`

   This allows the rule to log or restrict interaction based on the user who responded.
4. A separate rule handles the user's reply. It is triggered when the `telegramReplyId` **item** changes to `Reply_Water_Heater`. The script checks the user's message (via the `telegramMessage` item) and takes the appropriate action:
   - If the **response is** `Sí`, the water heater is turned off, `itemName.sendCommand(OFF)` and a message is sent to the replyier only.
   - If the **response is** `No`, the heater remains on and an informational message may be sent to the replyer only.

```
rule "Reply handler for water heater"
when
    Item telegramReplyId received update Reply_Water_Heater
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    username = telegramName.state.toString
    if (telegramMessage.state.toString == "Yes")
    {
      if (Calentador_Agua_Power_State.state == ON)
      {
        Calentador_Agua_Power_State.sendCommand(OFF)
        telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "De acuerdo " + username + ", apago el calentador de agua.")
      }
      else
      {
        telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, username + " el calentador de agua ya fue apagado.")
      }

    }
    else
    {
      if (Calentador_Agua_Power_State.state == OFF)
      {
        telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, username + " el calentador de agua ya fue apagado, si desea encenderlo debe hacerlo desde OpenHAB.")
      }
      else
      {
        telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "De acuerdo " + username + ", mantengo el calentador de agua encendido")
      }
    }
end
```

Figure 5.112 – *Water heater reply handler*

This setup provides both **notification** and **interaction capabilities**, improving automation efficiency and user control directly through Telegram.

5.9.3.3 Lights Automations and Notifications

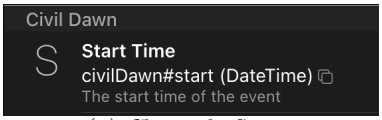
An **automation** system was developed to manage the **lighting** of the house by turning lights **on at sunset** and **off at sunrise**. These actions are complemented with Telegram notifications sent through the OpenHAB bot. The notifications inform users about the **status of the lights** and provide **interactive options** to **confirm or cancel the automation** directly from Telegram. This interaction is available for both turning the lights on and off.

The **basic notification rule** for reporting the current **status of the lights** is shown below:

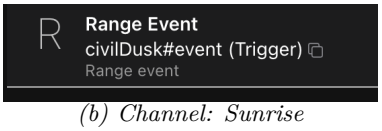
```
rule "Luces OFF"
when
    Item Luz_entrada_Power_State changed to OFF
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    telegramAction.sendTelegram("Luces apagadas")
end

rule "Luces ON"
when
    Item Luz_entrada_Power_State changed to ON
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    telegramAction.sendTelegram("Luces encendidas")
end
```

Figure 5.113 – *Lights ON/OFF status notification rule*

The **automation** is **triggered** based on the **astronomical events** of **sunset and sunrise**. These events are captured through **channels** provided by the **Astro binding**. Specifically, two channels from the `astro:sun:home` thing are used as event-based triggers. These channels emit events when the sun rises or sets, and OpenHAB responds accordingly by executing the relevant rule.



(a) Channel: Sunset



(b) Channel: Sunrise

Figure 5.114 – *Trigger channels for sunset and sunrise events*

The event logs confirms the triggering of these channels and is useful for debugging and verification purposes:

```
ater_Heater No"}
2025-05-16 21:15:00.001 [INFO ] [openhab.event.ChannelTriggeredEvent ] - astro:sun:c7205373d3:set#event triggered START
2025-05-16 21:15:00.003 [INFO ] [openhab.event.ChannelTriggeredEvent ] - astro:sun:c7205373d3:daylight#event triggered END
2025-05-16 21:18:00.001 [INFO ] [openhab.event.ChannelTriggeredEvent ] - astro:sun:c7205373d3:set#event triggered END
2025-05-16 21:18:00.002 [INFO ] [openhab.event.ChannelTriggeredEvent ] - astro:sun:c7205373d3:civilDusk#event triggered START
```

Figure 5.115 – *Channel events recorded in* `events.log`

The rule triggered at **sunset turns on the lights** and sends a **Telegram query** asking the user whether to proceed with the action. A confirmation handler is defined to process the user response, similar to the one described previously for the water heater (see Section 5.9.3.2). OpenHAB rules can manage more than one replyies depending on `telegramReplyId` value. That will be shown in tests section.

```
var String username = ""

rule "Sunset Rule"
when
    Channel "astro:sun:c7205373d3:civilDusk#event" triggered START
then
    Luz_entrada_Power_State.sendCommand(ON)
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    telegramAction.sendTelegramQuery("Las luces se han endendido debido a la puesta del sol, desea apagarlas?", "Reply_Luces_Sunset", "Yes", "No")
end

rule "Reply Sunset Rule"
when
    Item telegramReplyId received update Reply_Luces_Sunset
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    username = telegramName.state.toString
    if (telegramMessage.state.toString == "Yes")
    {
      if (Luz_entrada_Power_State.state == ON)
      {
        Luz_entrada_Power_State.sendCommand(OFF)
        telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "De acuerdo " + username + ", apago las luces.")
      }
      else
      {
        telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "Las luces ya fueron apagadas.")
      }

    }
    else
    {
      if (Luz_entrada_Power_State.state == OFF)
      {
        telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, username + " las luces ya fueron apagadas si desea encenderlas debe hacerlo desde OpenHAB.")
      }
      else
      {
        telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "Mantengo las luces encendidas")
      }
    }
end
```

Figure 5.116 – *Sunset rule with Telegram confirmation*

Likewise, the rule triggered at **sunrise turns the lights off** and **notifies the user**, again **offering the option to cancel the action** through Telegram bot:

```
rule "Sunrise Rule"
when
    Channel "astro:sun:c7205373d3:civilDawn#event" triggered START
then
    Luz_entrada_Power_State.sendCommand(OFF)
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    telegramAction.sendTelegramQuery("Las luces se han apagado debido a la salida del sol, las mantengo apagadas?", "Reply_Luces_Sunrise", "Yes", "No")
end

rule "Reply Sunrise Rule"
when
    Item telegramReplyId received update Reply_Luces_Sunrise
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    username = telegramName.state.toString
    if (telegramMessage.state.toString == "Yes")
    {
      if (Luz_entrada_Power_State.state == ON)
      {
        telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "La luces fueron encendidas si quieres apagarlas debe realizarlo desde OpenHAB")
      }
      else
      {
        telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "Mantengo las luces apagadas")
      }

    }
    else
    {
      if (Luz_entrada_Power_State.state == OFF)
      {
        Luz_entrada_Power_State.sendCommand(ON)
        telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "De acuerdo " + username + ", luces encendidas")
      }
      else
      {
        telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "Las luces ya fueron encendidas")
      }
    }
end
```

Figure 5.117 – *Sunrise rule with Telegram confirmation*

5.9.4 Irrigation Automation and Notifications

An **automation** system was developed to control the **irrigation process**, turning it on at 10:30 and off at **11:30** each day. The automation includes **confirmation** via Telegram and provides the option to **postpone irrigation** by one, two, or three days when it is turned on. **Notifications** are also sent to inform users about the current status of irrigation. The **irrigation time** can be adjusted within the OpenHAB rules by an administrator.

```
rule "Outdoor irrigation ON"
when
    Item EnchufeRiegoExterior_PowerState changed to ON
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    telegramAction.sendTelegram("Riego exterior encendido")
end
```

Figure 5.118 – *Notification when irrigation is turned on*

```
rule "Outdoor irrigation OFF"
when
    Item EnchufeRiegoExterior_PowerState changed to OFF
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    telegramAction.sendTelegram("Riego exterior apagado")
end
```

Figure 5.119 – *Notification when irrigation is turned off*

The **triggers** are based on a simple **cron schedule** that activates irrigation at 10:30 and deactivates it at 11:30 daily. When irrigation is turned on, the OpenHAB Telegram bot sends a confirmation message with options to leave it on, turn it off, or postpone it.

```
rule "Turn on irrigation at 10:30"
when
    Time cron "0 30 10 * * ?"
then
    if (IrrigationPostponeDays.state instanceof Number && IrrigationPostponeDays.state as Number > 0) {
        val daysLeft = (IrrigationPostponeDays.state as Number) - 1
        IrrigationPostponeDays.postUpdate(daysLeft)
        logInfo("Irrigation", "Irrigation postponed for another " + daysLeft + " day(s)")
        return;
    }
    if (EnchufeRiegoExterior_PowerState.state == OFF)
    {
        EnchufeRiegoExterior_PowerState.sendCommand(ON)
        val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
        telegramAction.sendTelegramQuery("El riego exterior ha sido encendido, debido a la programa diario, desea apagarlo?", "Reply_Outdoor_Irrigation", "Yes", "No", "Postpone")
    }
end
```

Figure 5.120 – *Telegram message sent when irrigation is turned on*

Upon receiving the user's **first reply**, the system processes three possible responses: `Yes`, `No`, or `Postpone`. A `Yes` leaves the irrigation on, a `No` turns it off, and `Postpone` prompts the bot to send a follow-up message with available postponement options.

```
rule "Handle main irrigation decision"
when
    Item telegramReplyId received update Reply_Outdoor_Irrigation
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    val response = telegramMessage.state.toString
    username = telegramName.state.toString

    switch response {
        case "Yes": {
            if (EnchufeRiegoExterior_PowerState.state == ON)
            {
                EnchufeRiegoExterior_PowerState.sendCommand(OFF)
                telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "De acuerdo " + username + ", riego apagado")
            }
            else
            {
                telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "Riego ya fue apagado previamente")
            }
        }
        case "No": {
            if (EnchufeRiegoExterior_PowerState.state == OFF)
            {
                telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "El riego fue apagado, si desea encenderlo debe realizarlo desde OpenHAB")
            }
            else
            {
                telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "Mantengo el riego encendido")
            }
        }
        case "Postpone": {
            if (EnchufeRiegoExterior_PowerState.state == ON)
            {
                EnchufeRiegoExterior_PowerState.sendCommand(OFF)
                telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "De acuerdo " + username + ", posponemos riego")
            }
            telegramAction.sendTelegramQuery("Cuantos dias deseas posponer el riego?", "Reply_Irrigation_Postpone","1", "2", "3")
        }
    }
end
```

Figure 5.121 – *Handler for the first user reply*

When the user selects a number of **days to postpone**, that value is assigned to an **integer item** using the command:

```
IrrigationPostponeDays.sendCommand(days)
```

This item is then used in the **10:30 automation rule**. Each day, the rule checks whether the **postpone value** is greater than zero. If it is, the irrigation is not activated, and instead, the **value is decremented** using:

```
IrrigationPostponeDays.postUpdate(daysLeft)
```

The `IrrigationPostponeDays` item was created through the OpenHAB web interface as a Number type, since only a single variable is needed. Go to **Items** section and press **+ icon**.

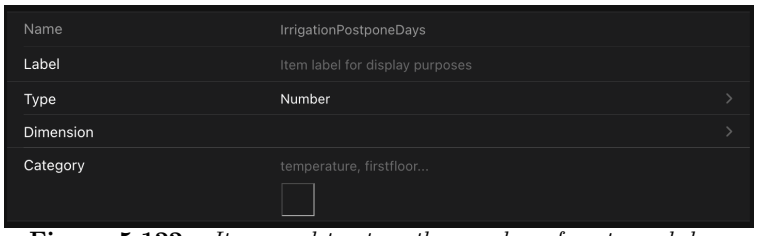


Figure 5.122 – *Item used to store the number of postponed days*

```
rule "Handle irrigation postpone response"
when
    Item telegramReplyId received update Reply_Irrigation_Postpone
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    val days = Integer::parseInt(telegramMessage.state.toString)

    IrrigationPostponeDays.sendCommand(days)
    telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "Riego pospuesto por " + days + " dia(s). Si desea encenderlo dirijase a OpenHAB")
end
```

Figure 5.123 – *Handler for the postpone response*

Each day at 11:30, if irrigation is active, it is **automatically turned off**. If it is already off, the rule takes no action.

```
rule "Handle irrigation postpone response"
when
    Item telegramReplyId received update Reply_Irrigation_Postpone
then
    val telegramAction = getActions("telegram","telegram:telegramBot:0f9b1ef81e")
    val days = Integer::parseInt(telegramMessage.state.toString)

    IrrigationPostponeDays.sendCommand(days)
    telegramAction.sendTelegramAnswer(telegramReplyId.state.toString, "Riego pospuesto por " + days + " dia(s). Si desea encenderlo dirijase a OpenHAB")
end
```

Figure 5.124 – *Rule to turn off irrigation at 11:30*

5.9.5 WhatsApp Rules

Through WhatsApp, only status updates regarding the Sunny Boy are sent. Before presenting the rule responsible for this functionality, the configuration process for the WhatsApp bot is described below.

The integration utilizes the free API provided by CallMeBot. In order to use the API, an **API key** must be obtained from the bot by following these steps:

1. Add the phone number **+34 694 23 41 84** to the contact list of the mobile device and name it `openhab bot`.
2. Send the message `"I allow callmebot to send me messages"` to the newly created contact.
3. Wait until the message `"API Activated for your phone number"` is received. This message includes the **API key** and the **URL required** to use the service.

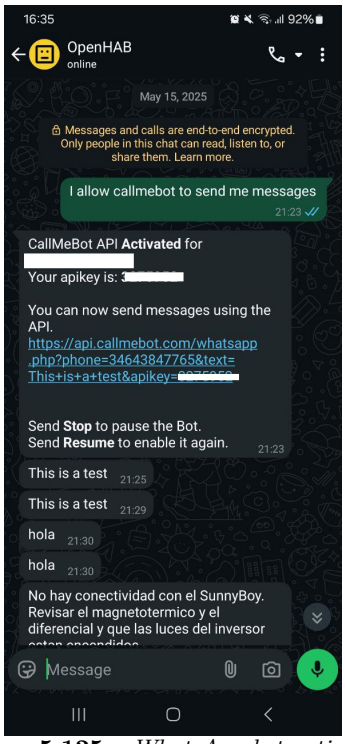


Figure 5.125 – *WhatsApp bot activation*

The rule displayed below functions similarly to the one used for Telegram. However, since there is no official binding available to manage WhatsApp messages within OpenHAB, a **`HTTP GET` request** is used to call the API with the corresponding **API key**, **message**, and **phone number** to whom send the message.

```
import java.net.URLEncoder

rule "SMA energy meter connection error whatsapp"
when
    Item SMA_STATUS changed to OFF
then
    try {
        val urlmessage = URLEncoder::encode("No hay conectividad con el SunnyBoy. Revisar el magnetotermico y el diferencial y que las luces del inversor estan encendidos", 'UTF-8')
        sendHttpGetRequest("https://api.callmebot.com/whatsapp.php?phone=+346[redacted]&apikey=3275952&text=" + urlmessage)
        sendHttpGetRequest("https://api.callmebot.com/whatsapp.php?phone=+346[redacted]&apikey=3526095&text=" + urlmessage)
        logInfo("sma status", "response")
    } catch (exception e) {
        logError("sma_status", e.getMessage)
    }
end

rule "SMA energy meter is running again whatsapp"
when
    Item SMA_STATUS changed to ON
then
    val urlmessage = URLEncoder::encode("SunnyBoy ya esta funcionando nuevamente", 'UTF-8')
    sendHttpGetRequest("https://api.callmebot.com/whatsapp.php?phone=+346[redacted]&apikey=3275952&text=" + urlmessage)
    sendHttpGetRequest("https://api.callmebot.com/whatsapp.php?phone=+346[redacted]&apikey=3526095&text=" + urlmessage)
end
```

Figure 5.126 – *WhatsApp notification rule*

5.10 New Interfaces

With respect to the user interfaces, an **update** was made to the **main interface** page (refer to Section 5.11 previous project [28]). Previously, only one page was available, and although other options such as sitemaps and measurement access pages existed, they are no longer in use.

The OpenHAB platform allows the creation of user interfaces using its built-in design **framework**. Users can build interfaces by simply designing **widgets**, while OpenHAB provides layout tools such as **columns**, **rows**, and various structural elements. Although OpenHAB offers several framework options, this documentation focuses on the use of **Pages**, which are applicable both on the web interface and in the mobile app.

The type of pages utilized and updated in this project are **Layout Pages**. To create a new layout page, the user should navigate to `Settings > Pages > + Create Layout`.

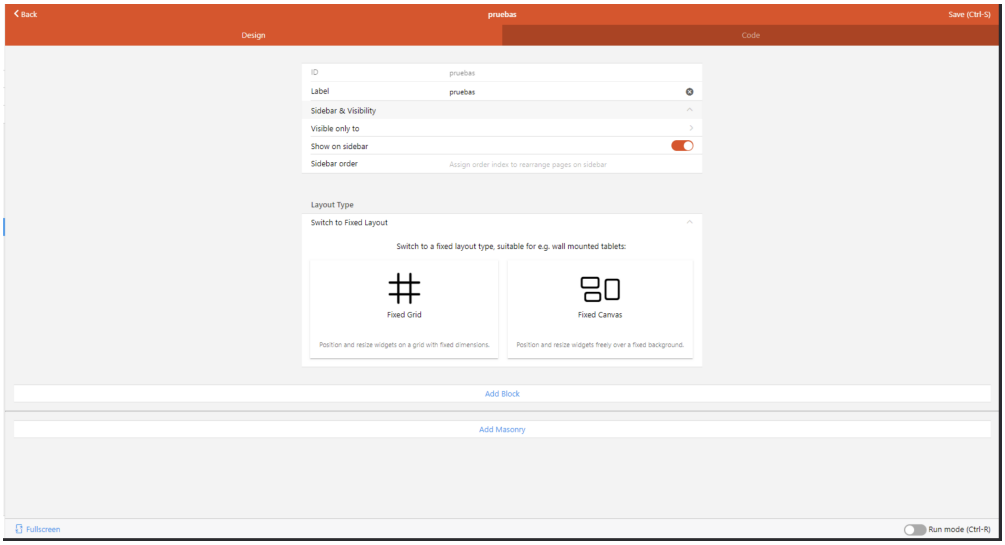


Figure 5.127 – *Layout Pages in OpenHAB*

When creating a layout page, OpenHAB offers two main layout types:

- **Fixed Grid** – All added widgets will share a uniform size by default.
- **Fixed Canvas** – Widgets can be freely placed and resized according to user preference.

The current implementation uses the **Fixed Canvas** layout with a two-part structure. Below are examples of previous web interface layouts:

- **Masonry** – A layout in which all widgets are grouped in a shared space without a specific order.

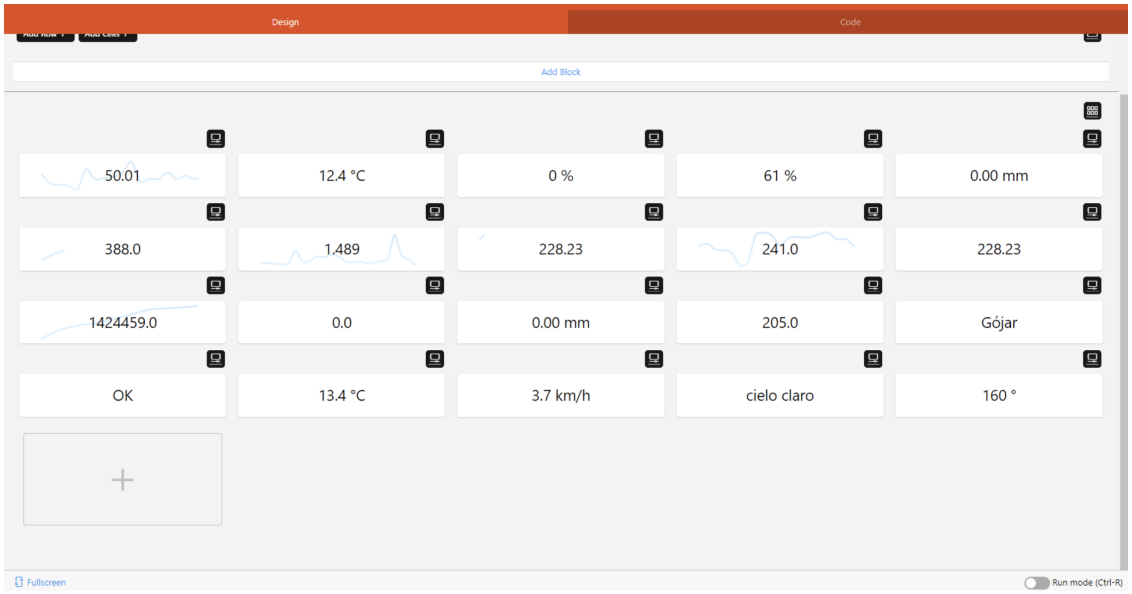


Figure 5.128 – *Masonry Layout*

- **Block** – A block-structured layout further divided into:
  - **Rows and Columns** – Widgets are placed within columns inside rows. The layout automatically adjusts: one column takes the full row width, two columns divide it evenly, and so on.
  - **Cells** – An alternative type of widget block offering a distinct visual appearance.

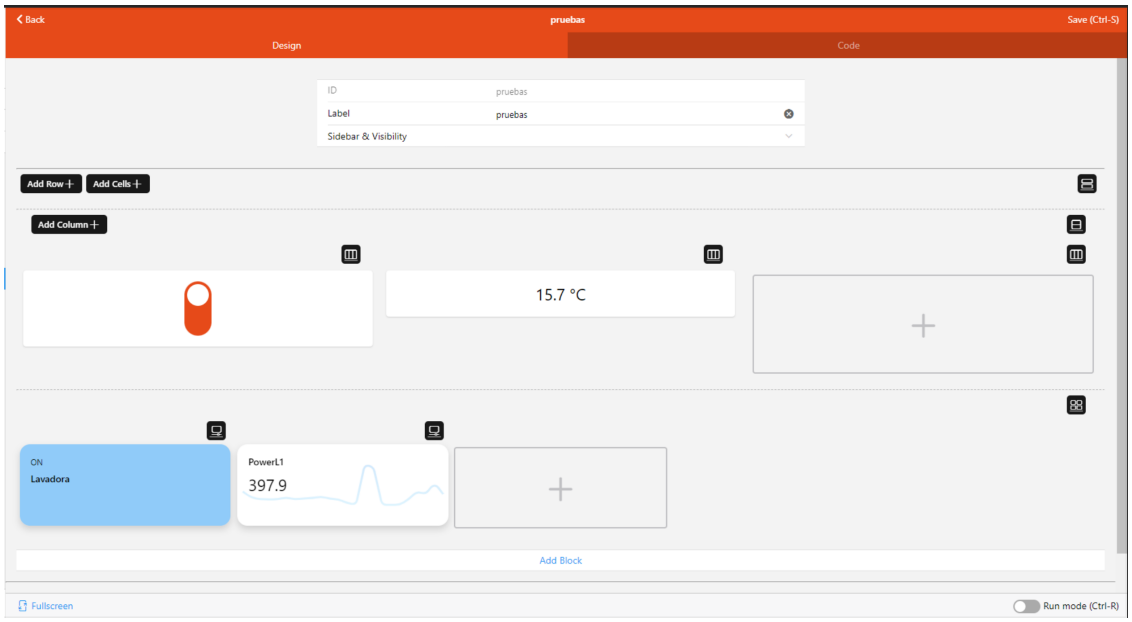


Figure 5.129 – *Block Layout and Cells*

To **enhance the interface**, additional **custom widgets** were implemented with the help of the OpenHAB community Since the **work primarily** consisted of adding **new widgets** and organizing them across **different pages**, each page will be presented along with the specific widgets added to it after showing the new widgets.

To create new widgets, the user must navigate to `Developer Tools > Widgets > +`, where the widget definition is written in YAML. After a widget is created, it can be added to any desired page and configured by assigning it the appropriate **Items** to display.

To **add a widget to a page**, the user should click on the designated **column** within the page layout. A menu will then appear offering a selection of available widgets:

- **Default widgets**, which are preconfigured by OpenHAB. A complete list is available at https://www.openhab.org/docs/ui/components/.
- **Custom widgets**, created by the user specifically for the project.

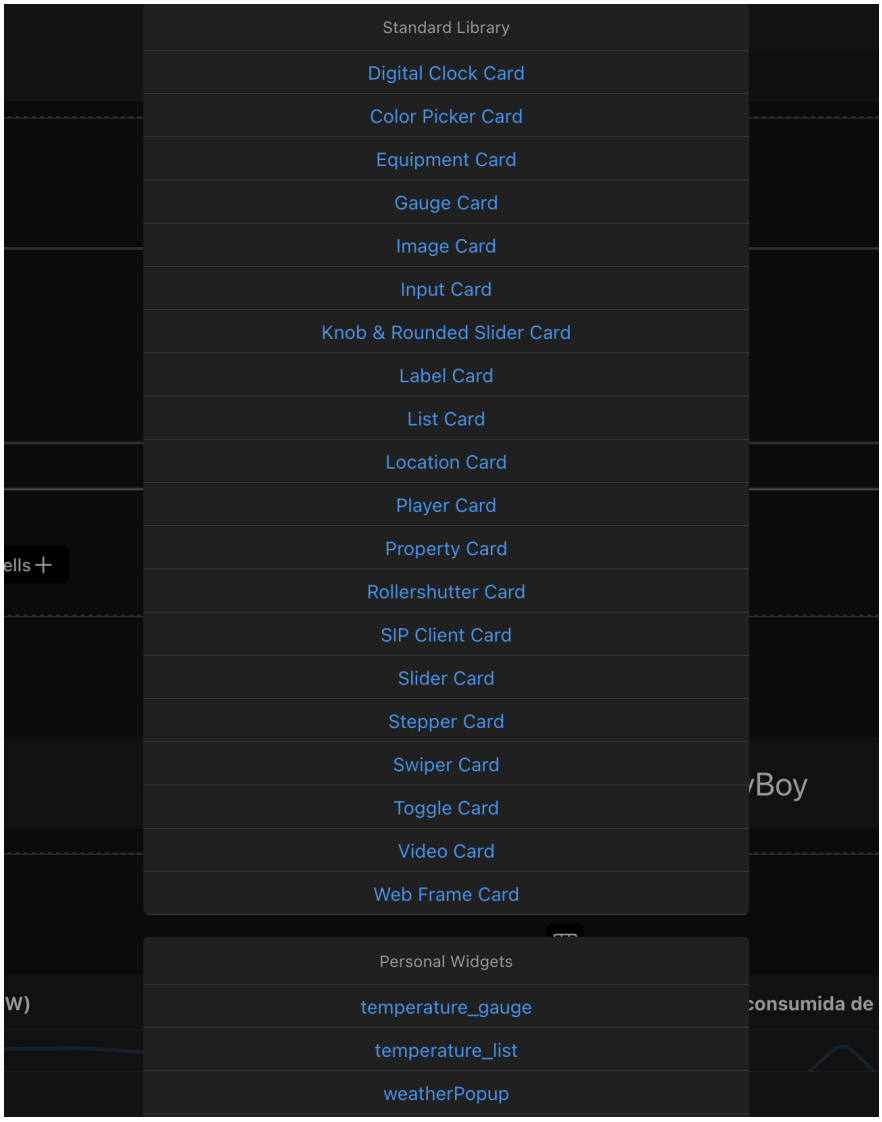


Figure 5.130 – *Add widget*

5.10.1 Improvement of the Default Page

The existing **default page was enhanced** by updating its button widgets, adding a visual representation of the solar energy flow, and incorporating **new widgets and charts** to display data retrieved from the weather API.

1. **Button Widgets:**
   To improve the interface's button functionality, a custom widget was added to the page.

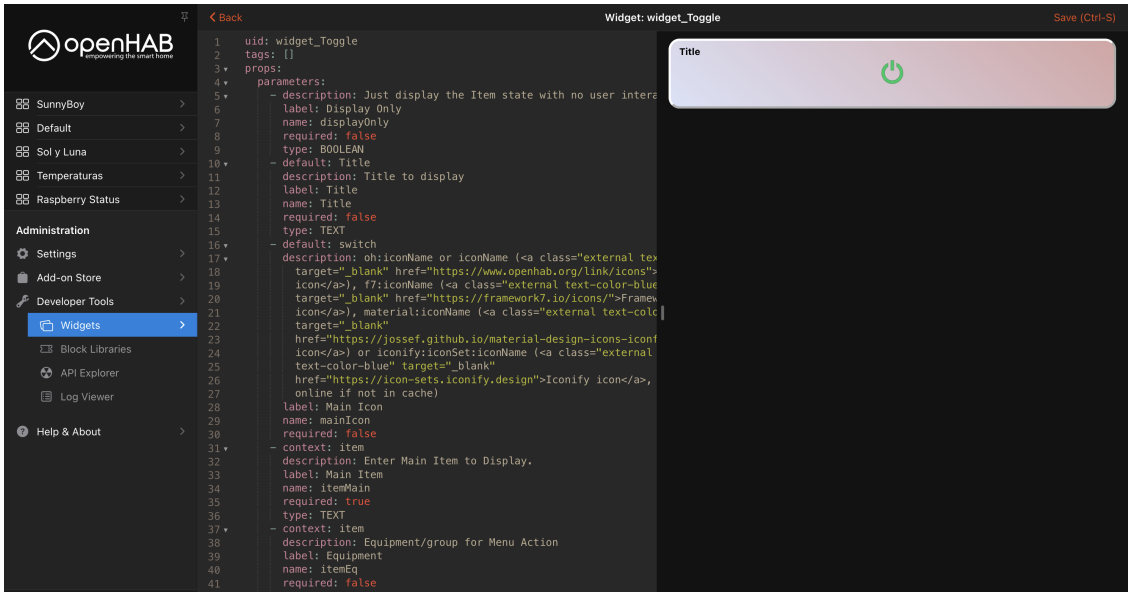


Figure 5.131 – *Button widget*

2. **Solar Energy Widget:**
   A widget was added to visualize the solar energy flow in real time, based on data retrieved from the SMA system.

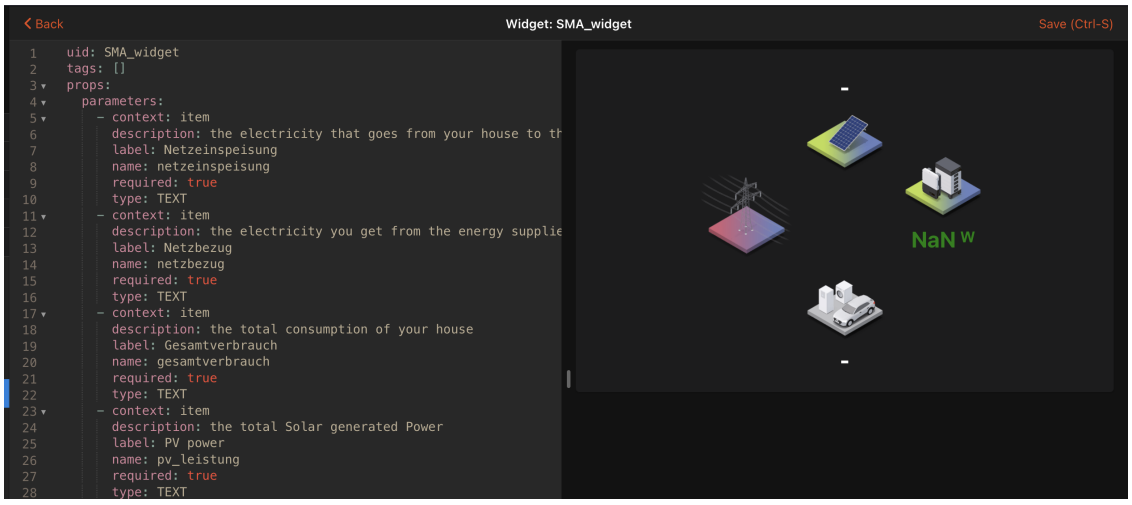


Figure 5.132 – *SMA widget*

3. **Charts and Data Representation:**
   Various widgets and charts were implemented using `Label Cards`, which allow for the selection of different types of analyzers.

4. **Temperature Widgets:**
   Temperature data was also visualized through dedicated widgets designed to present the information.

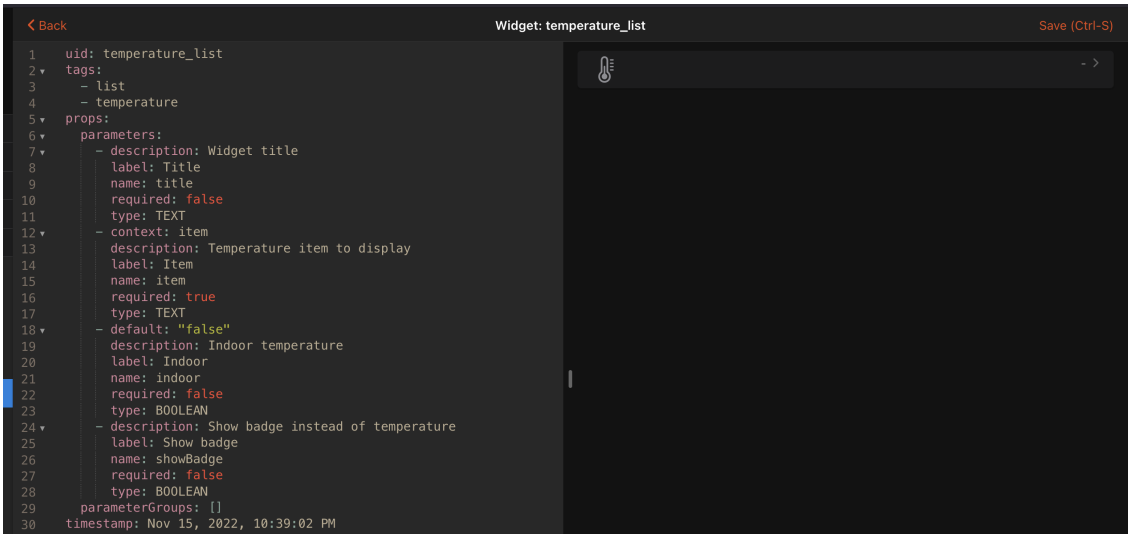


Figure 5.133 – *Temperature widget*

5. **Widget Configuration:**
   Once the widgets were created, they were **added to the page** in the corresponding column. Each widget was carefully **configured** by assigning the appropriate **Items**, **setting icons**, choosing **colors**, and optionally enabling chart displays. These **configurations** were performed either through the **graphical interface** or by directly editing the corresponding **`YAML` code**.

   Below are examples of the widget configurations:

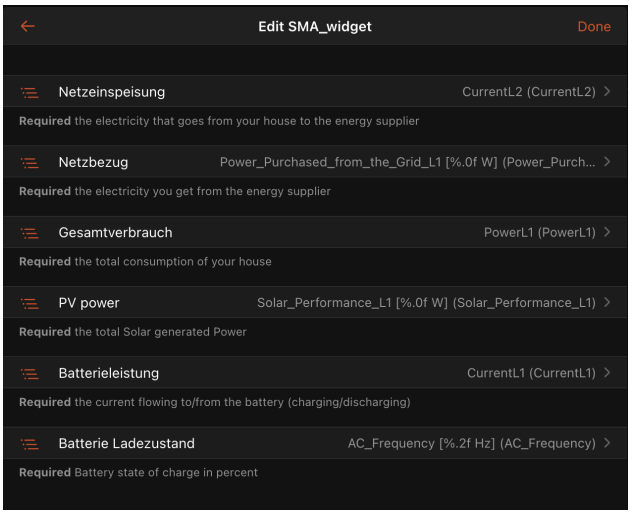


Figure 5.134 – *SMA widget configuration, assigning items*

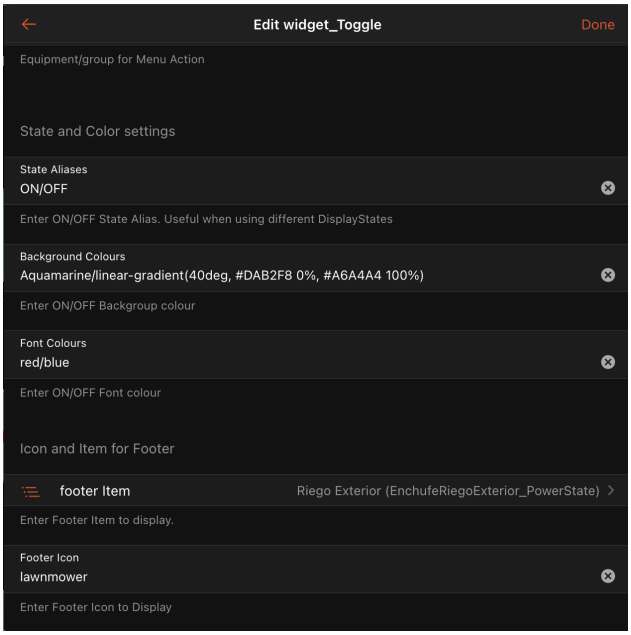


Figure 5.135 – *On/Off button widget configuration, items, colors, icosn, etc*

Final Result:
The improvements made to the default page resulted in a more informative and visually organized interface, combining buttons, temperature data, and solar energy flow representation.

Figure 5.136 – *Default page with buttons, temperature, and energy flow*

Charts are also accessible on the main page, and clicking on them reveals extended visualizations.

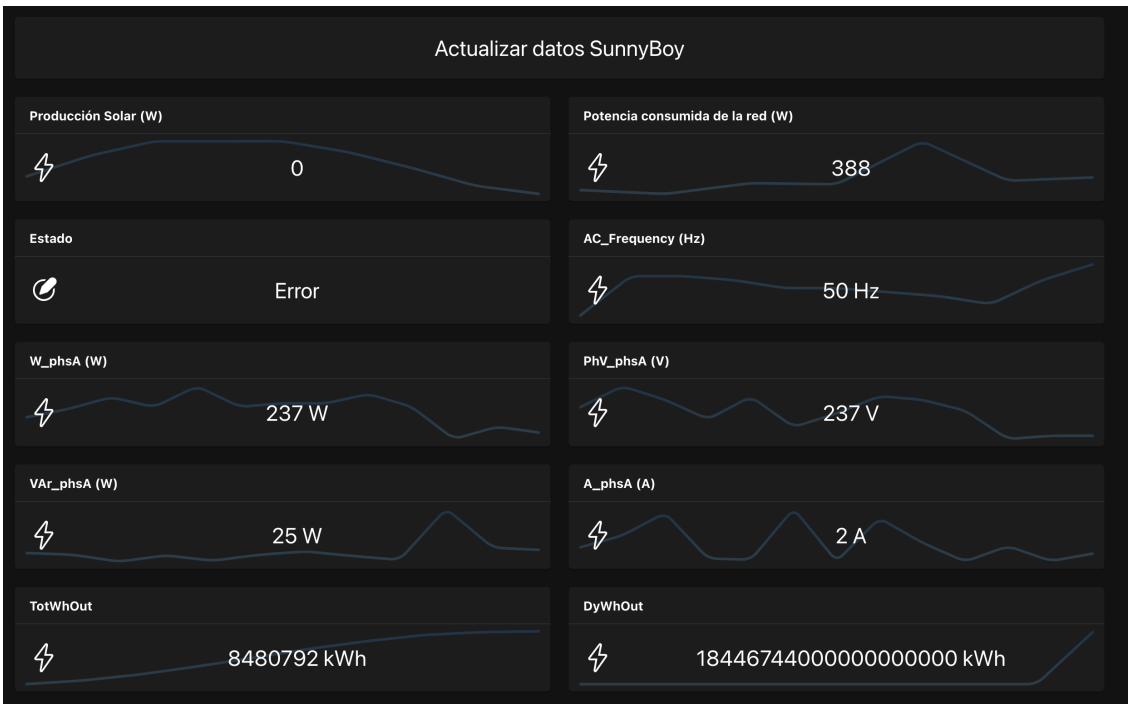


Figure 5.137 – *Charts*

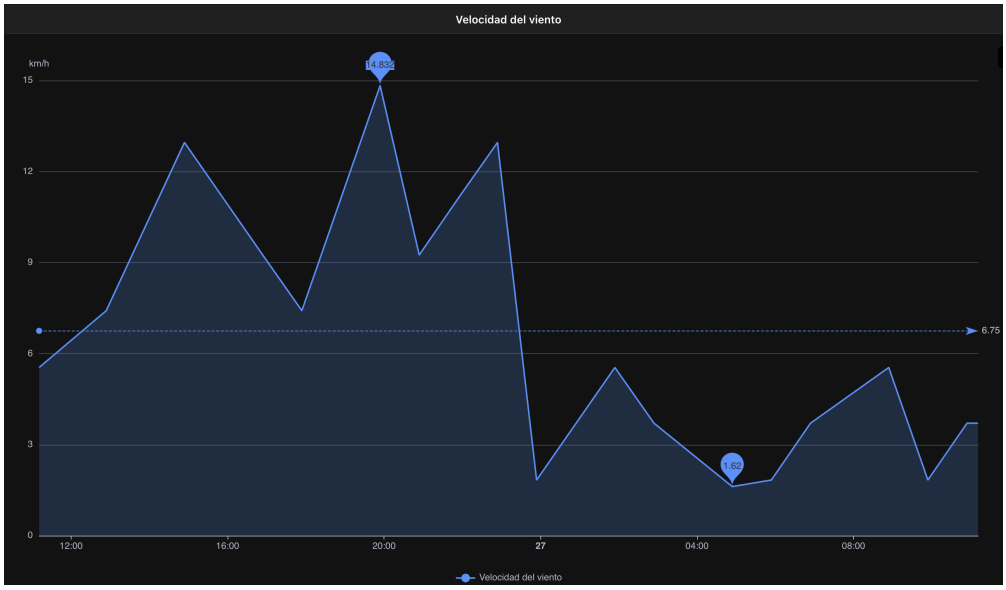


Figure 5.138 – *Extended chart view*

5.10.2 New Pages

After improving the default main page, several **new pages** were created to present additional information related to solar energy, wetaher forecast and temperatures, Raspberry Pi, and sun and moon information. The creation of each page followed a the same approach as updating the default page: **designing** the necessary **widgets**, **assigning** the correct **items** in the **configuration**, and **organizing** the widgets within appropriate layout blocks.

Regarding the addition of **new Things** (such as Sun, Moon, Raspberry Pi, etc.) and their **corresponding Items**, the process follows the same steps as described for other **Things**. First, the appropriate **binding** must be installed. Then, a new **Thing** is created by selecting the binding. Depending on the device or service, the **Thing** may be automatically discovered or manually added. Once the **Thing** is created, its **Channels** can be accessed, and the appropriate **Items** are linked to each channel accordingly.

1. **Solar Energy Information Page:** A dedicated page was developed to display data from the solar system, including inverter readings, energy flow, and production status.

   (a) Sunny Boy Inverter Information:

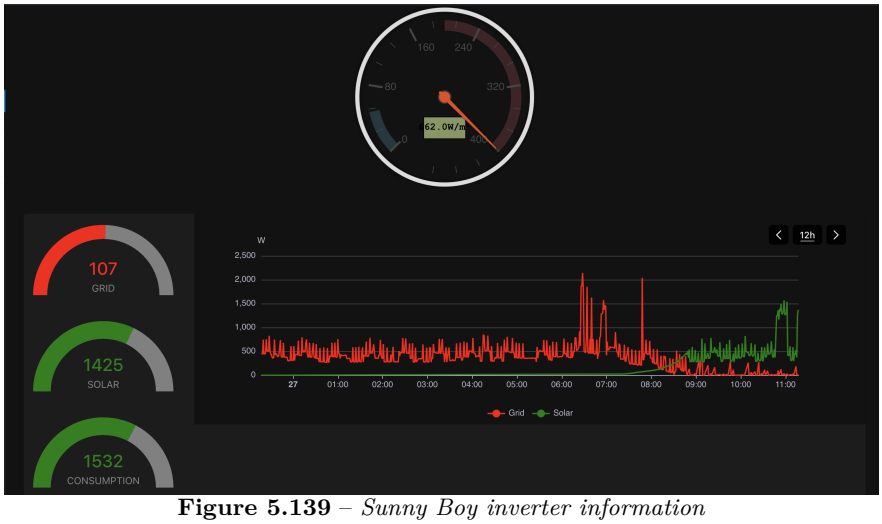


Figure 5.139 – *Sunny Boy inverter information*

   (b) Solar State and Power Graph:

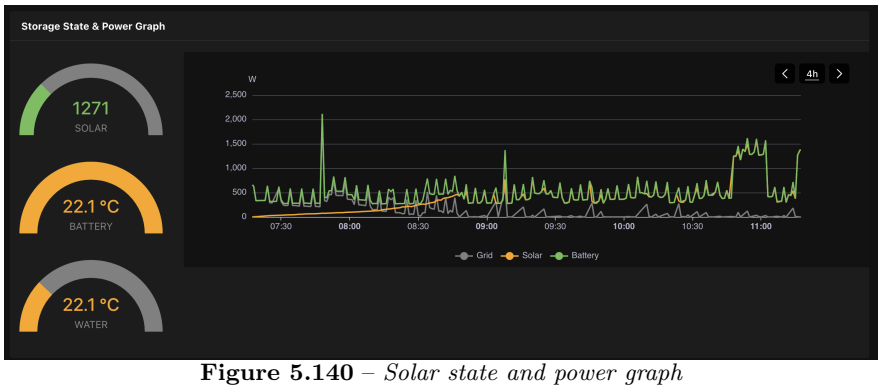


Figure 5.140 – *Solar state and power graph*

   (c) Brief Energy Flow Summary:

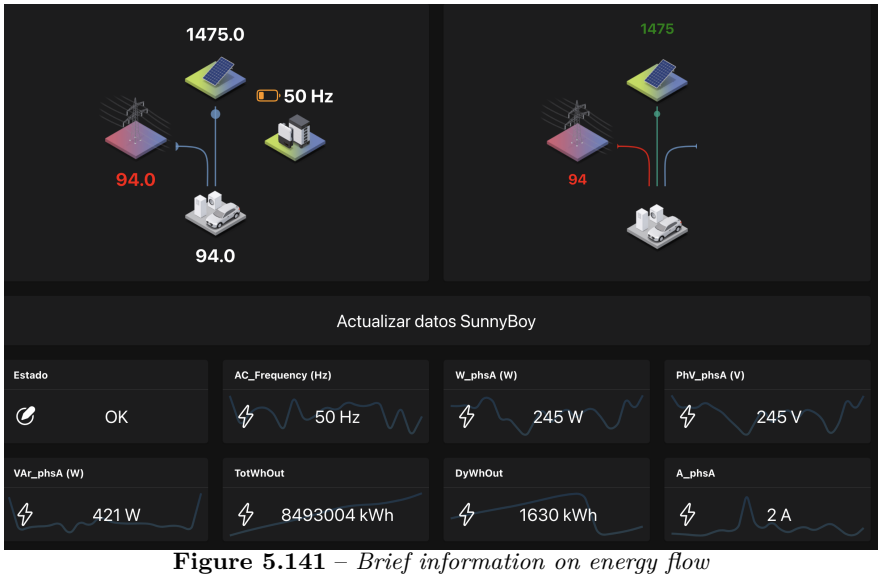


Figure 5.141 – *Brief information on energy flow*

2. **Sun and Moon Information Page:** This page provides visual and textual data about the sun and moon cycles.

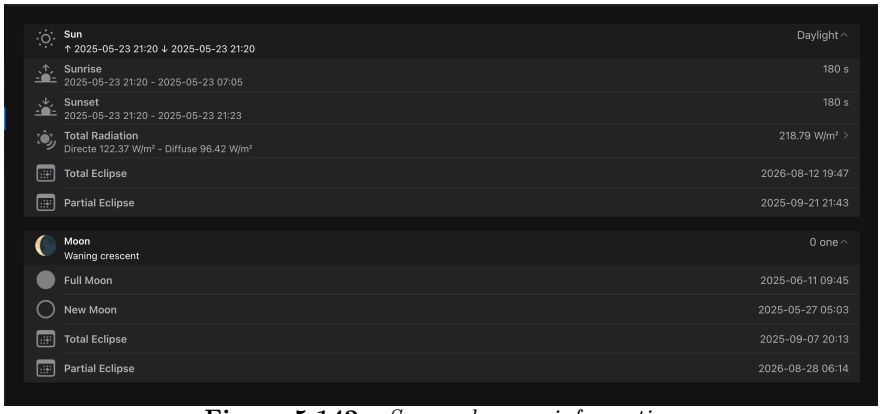


Figure 5.142 – *Sun and moon information*

3. **Raspberry Pi Status Page:** Real-time data about the Raspberry Pi hardware status is displayed on this page.

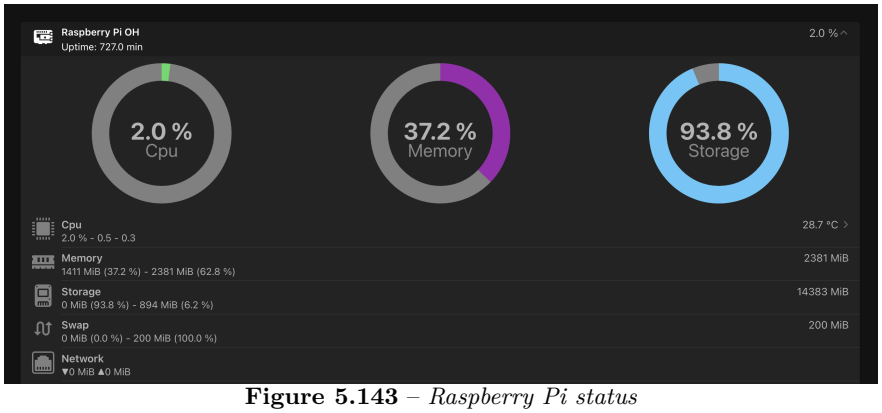


Figure 5.143 – *Raspberry Pi status*

4. **Temperature and Weather Forecast Page:** This page shows both indoor temperature readings and local weather forecasts.

   (a) Indoor Temperature Monitoring:

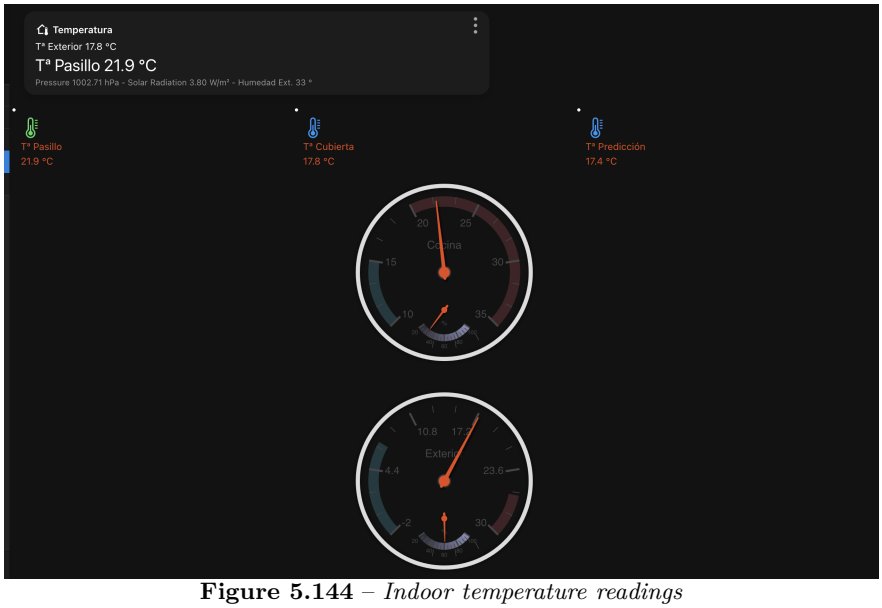


Figure 5.144 – *Indoor temperature readings*

(b) Local Weather Forecast:

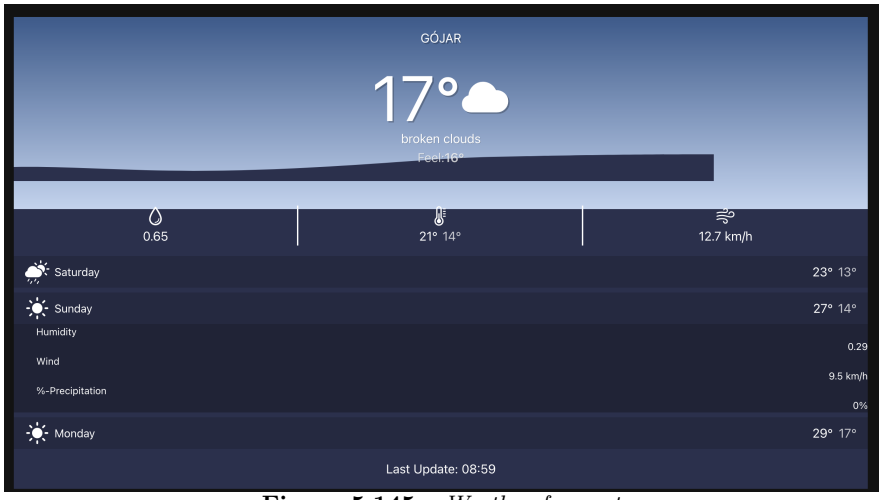


Figure 5.145 – *Weather forecast*

(c) Wind velocity and direction:

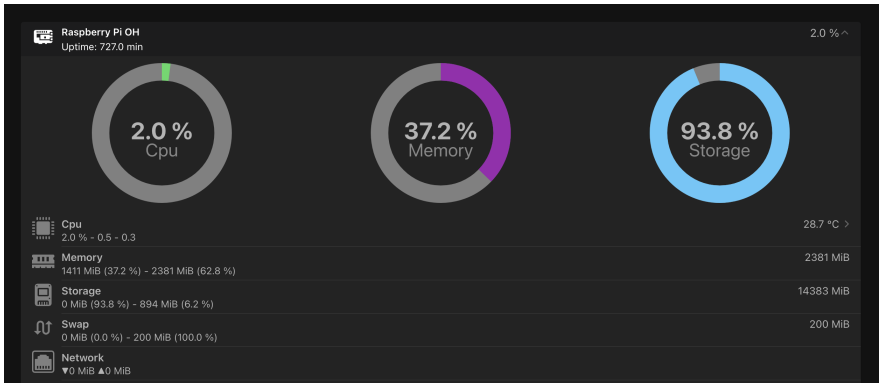


Figure 5.146 – *Wind velocity and direction*

Since the **Items** used for the **weather forecast** were created using a different approach, this process is explained in detail below.

To display the weather forecast, data was retrieved using the OpenWeatherMap binding. After configuring the OpenWeatherMap **bridge** (see Section 5.5.1), a **new Thing** was created. This Thing was configured with the **appropriate location** (Gojar) and a **forecast duration** of four days, which corresponds to the maximum period allowed by the free plan.

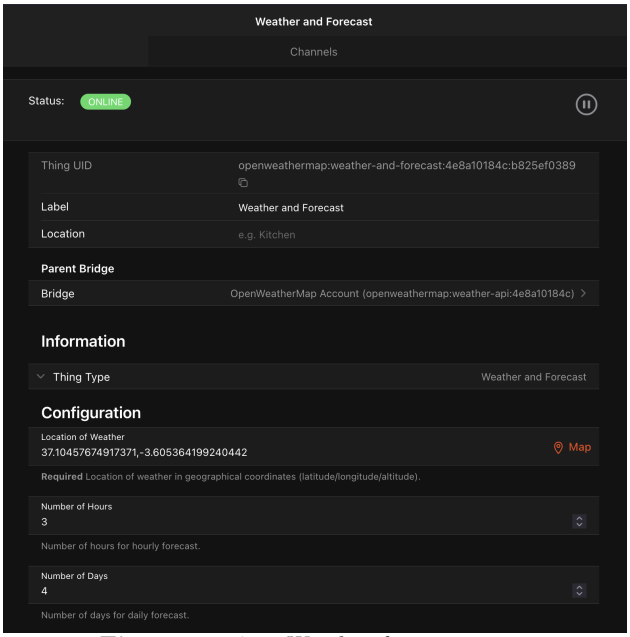


Figure 5.147 – *Weather forecast* `Thing`

Due to the **large number** of **forecast-related items** required, it was more efficient to define them using text files. These files, saved with a `.items` extension, are stored in the `openhab/conf/items` directory. Each item is defined according to the following structure:

```
<ItemType> <ItemName> "Label" <Icon> [SemanticTags] { ChannelBindingUID }
```

```
String Localweatherandforecast_StationName "Station Name" ["Point"]
{ channel="openweathermap:weather-and-forecast:4e8a10184c:b825ef0389:station#name" }

DateTime OneCallAPIweatherandforecast_ObservationTime "Observation Time" <Time> ["Point"]
{ channel="openweathermap:weather-and-forecast:4e8a10184c:b825ef0389:current#time-stamp" }

DateTime OneCallAPIweatherandforecast_Current_Sunrise "Sunrise Time" <Time> ["Point"]
{ channel="openweathermap:weather-and-forecast:4e8a10184c:b825ef0389:current#sunrise" }

DateTime OneCallAPIweatherandforecast_Current_Sunset "Sunset Time" <Time> ["Point"]
{ channel="openweathermap:weather-and-forecast:4e8a10184c:b825ef0389:current#sunset" }

String OneCallAPIweatherandforecast_Current_Condition "Weather Condition" <Sun_Clouds>   ["Point"]
{ channel="openweathermap:weather-and-forecast:4e8a10184c:b825ef0389:current#condition" }

String OneCallAPIweatherandforecast_Current_Conditionid "Weather Condition Id" ["Point"]
{ channel="openweathermap:weather-and-forecast:4e8a10184c:b825ef0389:current#condition-id" }

String OneCallAPIweatherandforecast_Current_Iconid "Icon Id" ["Point"]
{ channel="openweathermap:weather-and-forecast:4e8a10184c:b825ef0389:current#icon-id" }
```

Figure 5.148 – *Example of item definitions*

For the weather widget, instead of assigning each item manually in configuration, a **prefix-based configuration** was used, assigning to the widget all items that starts with specific prefix. This technique allows the widget to **automatically use and display** relevant item data, significantly simplifying the setup process.

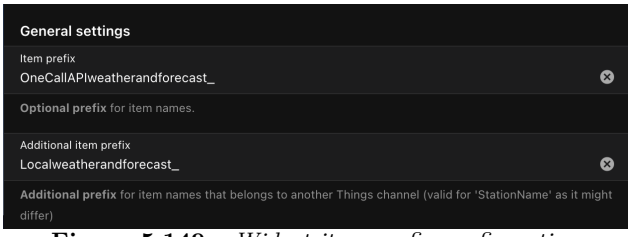


Figure 5.149 – *Widget item prefix configuration*

Furthermore, the widget adapts dynamically to changes in time, such as nighttime, by altering its color scheme and displaying the appropriate forecast icons, all this using **items information**. This enhances the visual experience and provides intuitive cues to the user.

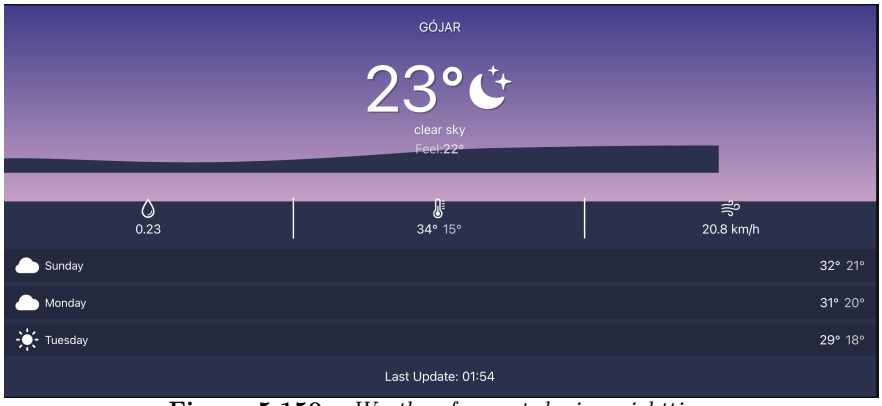
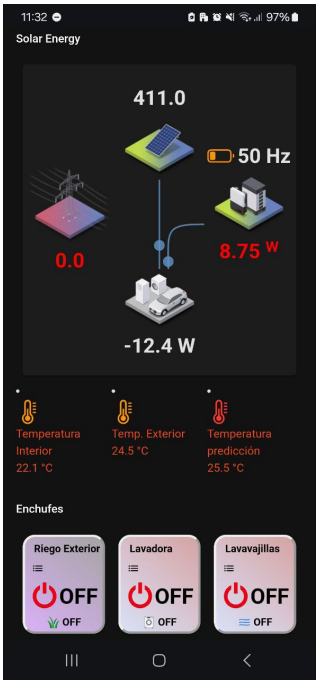
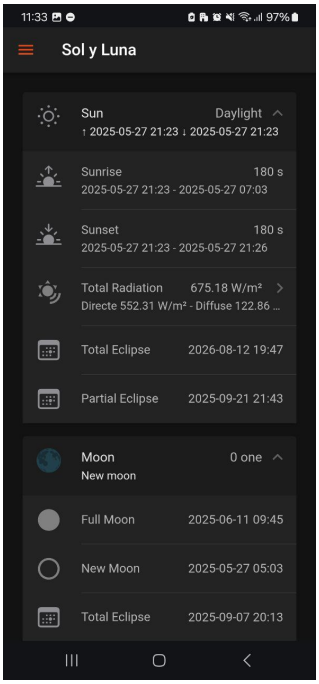


Figure 5.150 – *Weather forecast during nighttime*

All these pages can also be accessed through the mobile application:

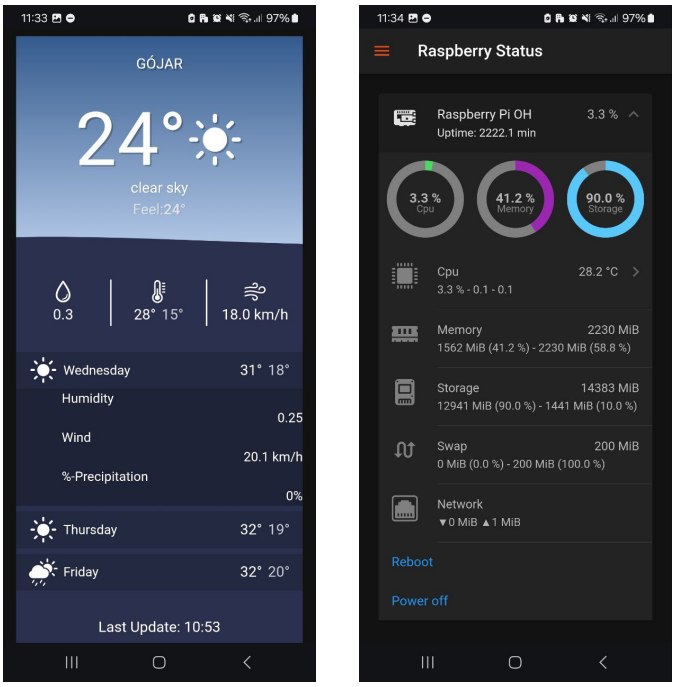


(a) *Main interface view 1*



(b) *Main interface view 2*

Figure 5.151 – *User interface shown in the mobile application (part 1)*



(a) *Main interface view 3* **(b)** *Main interface view 4*

Figure 5.152 – *User interface shown in the mobile application (part 2)*

5.11 Automatic Backups, Storage Configuration, and Logs

5.11.1 Automatic Backup Inside the Docker System

For automatic system backups, Duplicati is used. This backup solution is integrated within the Docker Compose setup, so when migrating the Docker system, no additional configuration is necessary—the backup setup resides entirely within the OpenHAB Docker folders.

A container was created using the official Duplicati image:

```yaml
duplicati:
  image: duplicati/duplicati:latest
  container_name: duplicati
  environment:
    - PUID=1000
    - PGID=1000
    - TZ=Europe/Madrid
    - CLI_ARGS= #optional
  volumes:
    - "./duplicati_config:/config"
    - "../openHAB:/source/openhab"
  ports:
    - 8200:8200
  restart: unless-stopped
```

Figure 5.153 – *Duplicati container*

The container is defined with the following parameters:

- **environment:**
  - `PUID=1000` and `PGID=1000` — Define the user and group IDs under which the container operates. This ensures correct permissions for accessing host files.
  - `TZ=Europe/Madrid` — Sets the container's time zone.
- **volumes:**
  - `./duplicati_config:/config` — Stores Duplicati configuration files persistently.
  - `../openHAB:/source/openhab` — Mounts the OpenHAB directory to make it available for backup.
- **ports:**
  - `8200:8200` — Exposes Duplicati's web interface on the host via `http://localhost:8200`.
- **restart: unless-stopped** — Configures the container to restart automatically unless manually stopped.

Configuration Steps

1. A password is created to encrypt the backups:

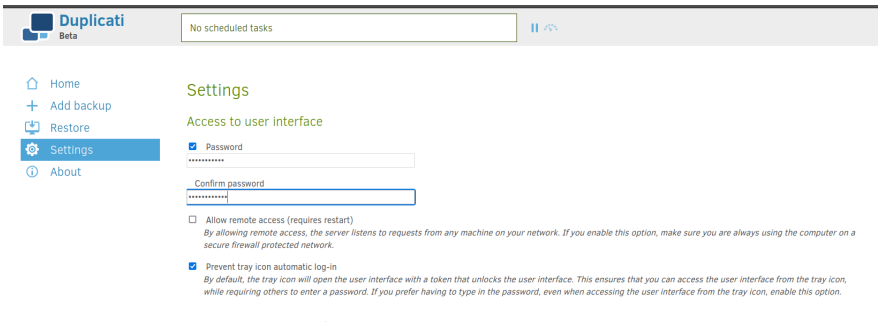


Figure 5.154 – *Duplicati container interface*

2. A new backup is added by specifying a name, encryption method, and password:

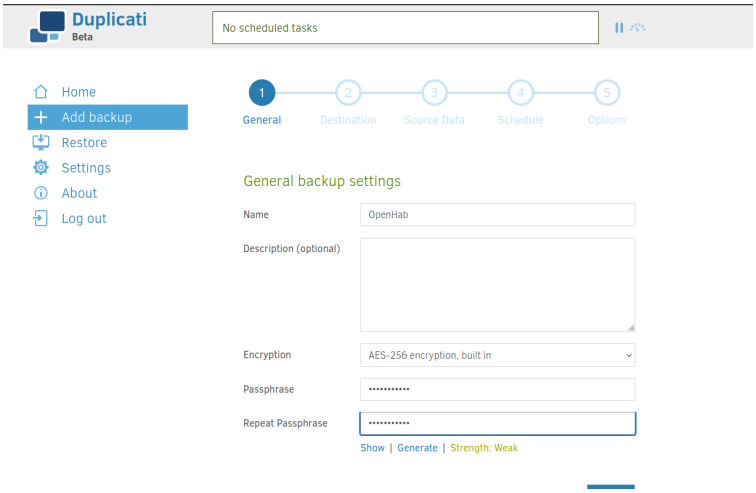


Figure 5.155 – *General backup setup*

3. The backup destination is selected—Google Drive in this case. Upon choosing the destination and folder, Duplicati's authentication handler generates an `authID` to establish a connection. The connection can be tested immediately.

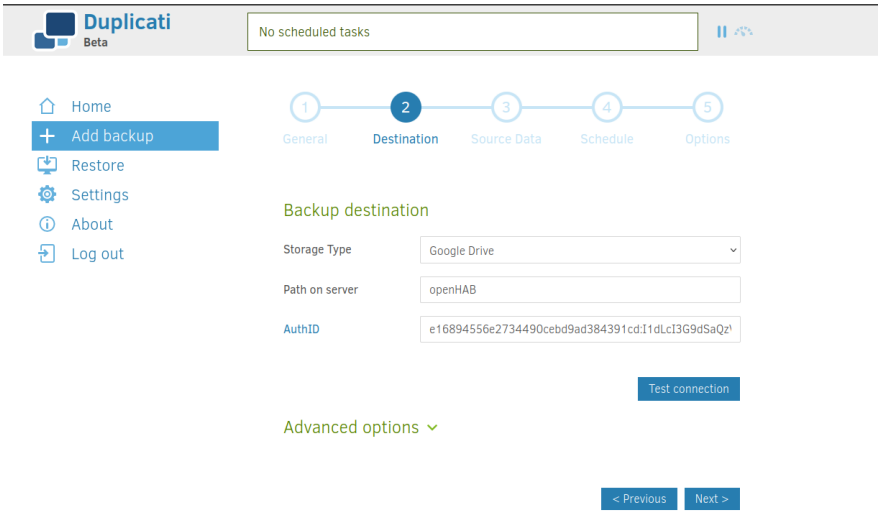
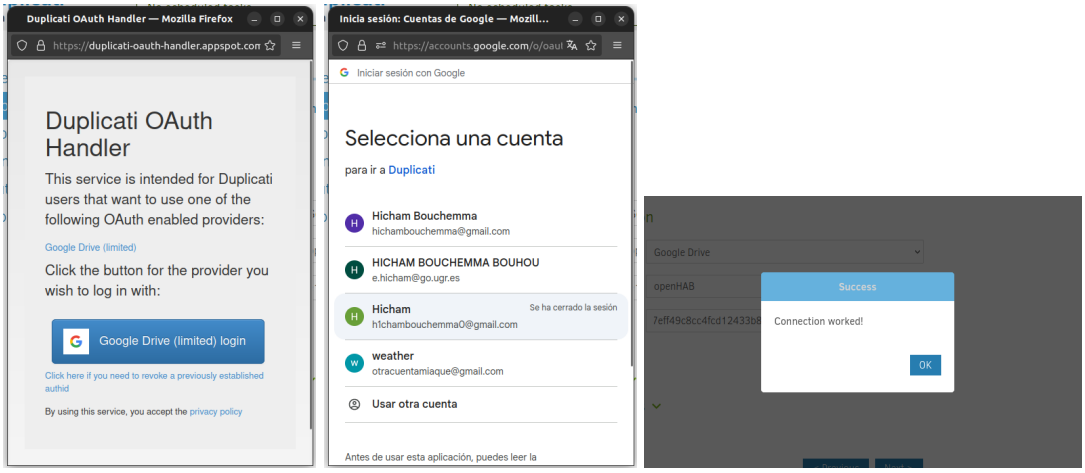


Figure 5.156 – *Backup destination configuration*



(a) *Authorization Step 1* **(b)** *Authorization Step 2* **(c)** *Test Connection*

Figure 5.157 – *Google Drive authorization process*

4. The source data to be backed up is selected, OpenHAB folder:

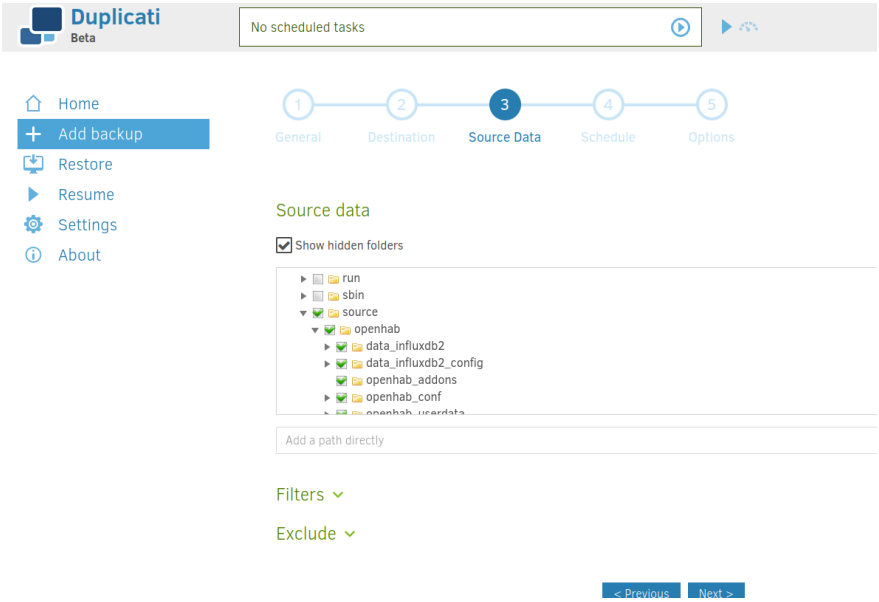


Figure 5.158 – *Selection of source data*

5. A schedule for automatic backups is configured:

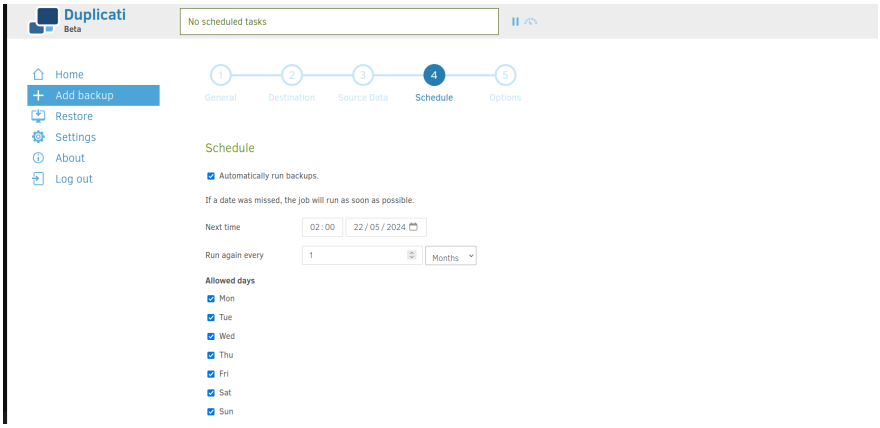


Figure 5.159 – *Backup schedule configuration*

6. Backup retention policy and volume size are defined:

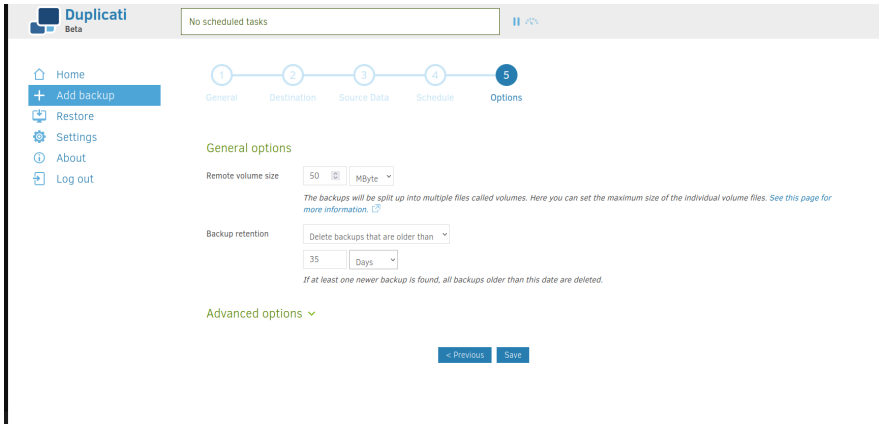


Figure 5.160 – *Backup retention and volume settings*

The configured backups are displayed on the Duplicati home interface:

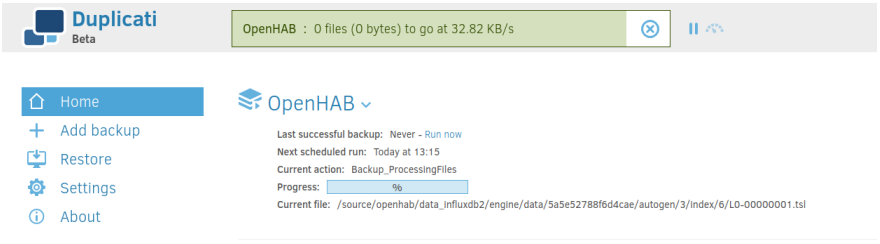


Figure 5.161 – *List of configured backups*

Restoration: To restore a backup, the user simply selects the desired backup through the interface, and Duplicati restores it to the source folder as defined in the container setup.

5.11.2 Backups in the System and Maintenance

System backups are managed using a scheduled script executed by `cron`.

```
#!/bin/bash

#Para Subir Ficheros a la Cuenta de GranaSAT@go.ugr.es

echo "Borrando todos los ficheros de copias de seguridad anteriores"
echo "Borrando /home/granasat/CopiaSeguridad/*.tar.gz"
rm /home/granasat/CopiaSeguridad/*.tar.gz

echo "Apagamos el Docker de OpenHAB"
sudo docker compose -f /home/granasat/openHAB/docker-compose.yml down

echo "Borramos todas las images si contenedor asociado"
sudo docker images prune -a -f

sudo tar czvf /home/granasat/CopiaSeguridad/OpenHAB.tar.gz /home/granasat/openHAB/
sudo chown granasat:granasat /home/granasat/CopiaSeguridad/OpenHAB.tar.gz
rclone -v sync /home/granasat/CopiaSeguridad/OpenHAB.tar.gz AMRoldanGoUgrEs:OpenHAB_raspberryBackup

echo "Borrando /home/granasat/CopiaSeguridad/*.tar.gz"
rm /home/granasat/CopiaSeguridad/*.tar.gz

echo "Levantamos el Docker de OpenHAB"
docker compose -f /home/granasat/openHAB/docker-compose.yml build --no-cache
docker compose -f /home/granasat/openHAB/docker-compose.yml up -d
```

Figure 5.162 – *Backup script*

The tool `rclone` is employed for **remote backups**. **Rclone** is a command-line utility designed to synchronize files and directories with various cloud storage services.

In this setup, after a **compressed backup** of the OpenHAB system is generated locally, `rclone` is used to **upload the backup** file to a specified folder in a remote Google Drive account. This ensures secure off-site storage and provides a reliable method for data recovery in case of local failures.

Once the upload is completed, the **local backup file is deleted** to save disk space, as a secure copy already exists in the cloud. This strategy guarantees regular, automated, and efficient backup cycles with minimal manual intervention.

To **automate the execution** of the backup process, a `cron` job is used. **Crontab** is a Linux utility that allows scheduling commands or scripts to run at specified times and intervals.

In this case, the **script is scheduled** to **run every Sunday at 3:00 AM** by adding an entry using the command `crontab -e`. The output of the script is redirected to a **log file**, ensuring that any issues or execution details are recorded.

```
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
0 3 * * 7 /home/granasat/CopiaSeguridadDriveGranaSAT.sh >> /home/granasat/backup_openhab.log 2>&1
```

Figure 5.163 – *Scheduled execution with* `crontab`

For regular maintenance, all unused Docker images, images without container asociated, are deleted using the command `docker image prune -a -f`. This process is performed after stopping and removing all OpenHAB containers. Doing so ensures that the **latest image version** is pulled the next time the containers are launched. Additionally, using `docker build -no-cache` guarantees that the build process uses the most up-to-date base image of OpenHAB.

This procedure is suitable for **version updates and basic maintenance**. However, for **advanced maintenance** tasks related to system errors or software bugs, ongoing support from the **development team is required**. Therefore, it is recommended that the client establishes a support contract to ensure proper long-term maintenance of the system.

5.11.3 Storage Cleanup and Retention Policy Change

Regarding **storage management**, data older than six months is deleted to free up space. The procedure is as follows:

1. Access the InfluxDB container's bash shell in interactive mode using the command: `docker exec -it influxdb2 /bin/bash`.
2. Since InfluxDB 2 does not include a native shell to manage databases, the `influx v1 shell` command is used. This provides compatibility by **mapping buckets to databases**.
3. Select the `openhab_db` database with the command: `use "openhab_db"`.
4. If no specific measurement is specified, queries can be run across all measurements. Data older than six months is deleted using a command similar to: `DELETE WHERE time < '2025-01-01'`.
5. To view timestamps in a human-readable format, the `precision rfc3339` option is used.

6. Finally, the **deletion is verified** by ordering the data by ascending time to confirm that the earliest records have been removed.

```
granasat@granasat:~ $ docker exec -it influxdb2 /bin/bash
root@5ffc682b758d:/# influx v1 shell
InfluxQL Shell dev
Connected to InfluxDB OSS v2.7.11
> show DATABASES
> use "openhab_db"
> DELETE where time < '2025-01-01'
> precision rfc3339
> select * from "AC_Frequency" order by time ASC limit 10
```

Figure 5.164 – *InfluxDB v1 shell interface*

```
Interactive Table View (press q to exit mode, shift+up/down to navigate tables):
Name: AC_Frequency
┌─────────┬──────────────────────┬──────────────┬──────────────────┐
│ index   │ time                 │ item         │ value            │
├─────────┼──────────────────────┼──────────────┼──────────────────┤
│       1 │ 2025-01-06T22:28:03.581Z │ AC_Frequency │ 50.0100000000 │
│       2 │ 2025-01-06T22:29:01.144Z │ AC_Frequency │ 49.9900000000 │
│       3 │ 2025-01-06T22:30:01.172Z │ AC_Frequency │ 50.0000000000 │
│       4 │ 2025-01-06T22:31:01.226Z │ AC_Frequency │ 49.9800000000 │
│       5 │ 2025-01-06T22:32:01.552Z │ AC_Frequency │               │
```

Figure 5.165 – *Check data deletion*

Changing the retention policy does not require using the InfluxDB v1 shell.

- The existing buckets and their retention policies can be reviewed directly.
- To adjust the retention policy, the retention duration is set in hours, with approximately 4,380 hours corresponding to six months.

```
root@8de3a7a3194a:/# influx bucket list
ID                  Name            Retention       Shard group duration    Organization ID         Schema Type
f960184e644119d0    _monitoring     168h0m0s        24h0m0s                 2fab9bec0e64df9f        implicit
545d883667748ce1    _tasks          72h0m0s         24h0m0s                 2fab9bec0e64df9f        implicit
327523dbe1ff4ae0    adminBucket     infinite        168h0m0s                2fab9bec0e64df9f        implicit
cac1ec9f9e263b92    openhab_db      infinite        168h0m0s                2fab9bec0e64df9f        implicit
```

Figure 5.166 – *Current retention policy*

```
root@8de3a7a3194a:/# influx bucket update -i cac1ec9f9e263b92 -n openahb_db -r 8760h
ID                  Name            Retention       Shard group duration    Organization ID         Schema Type
cac1ec9f9e263b92    openahb_db      8760h0m0s       168h0m0s                2fab9bec0e64df9f        implicit
root@8de3a7a3194a:/#
```

Figure 5.167 – *Updated retention policy*

5.11.4 Logs

OpenHAB generates two primary log files, located in the ‘ `openhab/userdata/logs/` directory:

- `events.log`: Records all updates and commands related to items and channels, such as state changes and triggers.
- `openhab.log`: Contains system-level logs including startup information, rule execution, bindings, and errors.

```
root@granasat:/openhab# tail -f userdata/logs/events.log
2025-05-27 19:53:05.190 [INFO ] [ab.event.ThingStatusInfoChangedEvent] - Thing 'amazonechocontrol:smartHomeDevice:b76179028d:cbb6e901-dd20-4624-a334-9ef8eea6552d' changed from UNKNOWN: Sta
te not found to ONLINE
2025-05-27 19:53:05.191 [INFO ] [ab.event.ThingStatusInfoChangedEvent] - Thing 'amazonechocontrol:smartHomeDevice:b76179028d:8bdf3d94-e1e6-4f73-a13b-024ebea95cda' changed from UNKNOWN: Sta
te not found to ONLINE
2025-05-27 19:53:05.191 [INFO ] [ab.event.ThingStatusInfoChangedEvent] - Thing 'amazonechocontrol:smartHomeDevice:b76179028d:b1ec0b93-26b7-42fe-8f06-f39582441084' changed from UNKNOWN: Sta
te not found to ONLINE
2025-05-27 19:53:05.192 [INFO ] [ab.event.ThingStatusInfoChangedEvent] - Thing 'amazonechocontrol:smartHomeDevice:b76179028d:94724ff6-396a-41da-aeaa-a6f8f6bfa1b8' changed from UNKNOWN: Sta
te not found to ONLINE
2025-05-27 19:53:05.193 [INFO ] [ab.event.ThingStatusInfoChangedEvent] - Thing 'amazonechocontrol:smartHomeDevice:b76179028d:284419a9-5983-4d69-b23c-d5479b74808e' changed from UNKNOWN: Sta
te not found to ONLINE
2025-05-27 19:53:05.194 [INFO ] [ab.event.ThingStatusInfoChangedEvent] - Thing 'amazonechocontrol:smartHomeDevice:b76179028d:2ec384a5-e732-4416-a91a-211642939afc' changed from UNKNOWN: Sta
te not found to ONLINE
```

Figure 5.168 – *`events.log` – Logs related to item events*

```
^C
root@granasat:/openhab# tail -f userdata/logs/openhab.log
2025-05-27 19:56:01.069 [INFO ] [penhab.core.model.script.HM_JSON_Out] - {"SerialNo": "3009888646", "PowerTotal": 203.4, "PowerL1": 203.4, "PowerL2": 0.0, "PowerL3": 0.0, "CurrentL1": 16.0
5, "CurrentL2": 0.0, "CurrentL3": 0.0, "VoltageL1": 231.081, "VoltageL2": 0.0, "VoltageL3": 0.0, "EnergyForward": 12668.8336, "EnergyForwardL1": 12663.6343, "EnergyForwardL2": 0.0, "Energy
ForwardL3": 5.6325, "EnergyReverse": 654.1392, "EnergyReverseL1": 421.644, "EnergyReverseL2": 13.6615, "EnergyReverseL3": 219.2668}
```

Figure 5.169 – *`openhab.log` – System and rule execution logs*

To inspect these logs, one common approach is to attach to the OpenHAB Docker container using an **interactive shell**. However, this method is not ideal for real-time monitoring or continuous observation.

For a more **user-friendly** and **real-time** view of logs, the tool **Frontail** can be used. Frontail is a lightweight web application built with Node.js that displays log files in a web browser with automatic updates.

We can deploy Frontail in a separate Docker container, **mounting the OpenHAB log directory** to allow access.

```yaml
frontail:
  image: welteki/frontail-openhab:latest
  container_name: frontail
  depends_on:
    - openhab
  ports:
    - "9001:9001"
  volumes:
    - ./openhab_userdata:/openhab/userdata
  networks:
    backend-nw:
      ipv4_address: 10.5.0.7
```

Figure 5.170 – *Frontail container for real-time log viewing*

Once configured, Frontail provides a clean web interface to stream logs such as `events.log` and `openhab.log`, making debugging and monitoring much easier with fiter options.

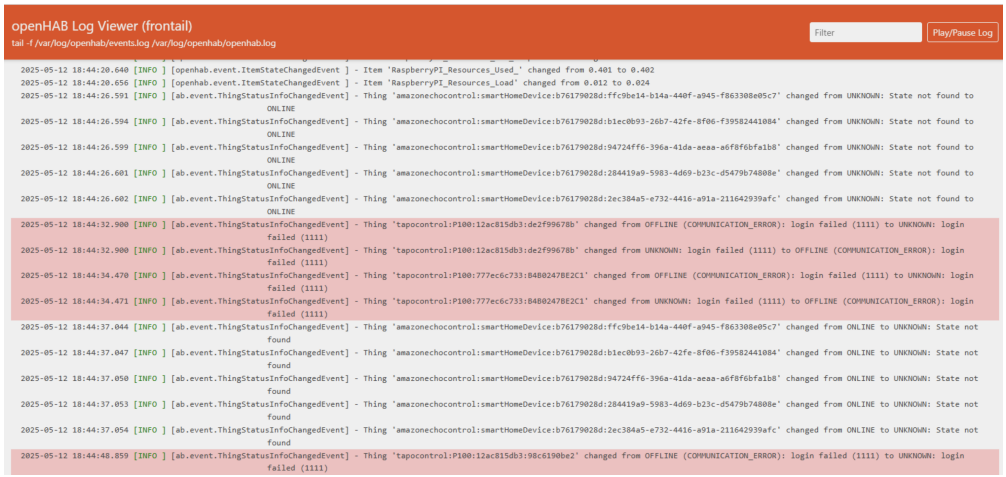


Figure 5.171 – *Live OpenHAB logs in Frontail*

5.11.5 Security

In evaluating the system's security, three main aspects must be considered: the **database**, the OpenHAB platform, and the **communication** between them.

1. **InfluxDB 2.0** uses **token-based authentication** to manage access. Each user or application must present a valid token to perform operations, ensuring that only authorized entities can read or write data. Additionally, InfluxDB can be configured to accept **remote connections** securely, which is essential for distributed systems or cloud-based integrations.
2. **OpenHAB** incorporates several **built-in security mechanisms**. By default, it restricts access to sensitive settings and offers **role-based user management**. OpenHAB Cloud, which enables **remote access and integrations** (e.g., via voice assistants), employs **secure encrypted channels and requires authentication**. These safeguards reduce exposure to unauthorized users.
3. **Communication between OpenHAB and InfluxDB** is secured through **token-based authentication** within an isolated Docker network. This setup ensures that only containers within the same virtual network can communicate directly, minimizing the surface for external attacks. Moreover, external access to the OpenHAB system is protected through a VPN tunnel configured via OpenHAB Cloud, reinforced with secure **login credentials and encrypted traffic**. This multi-layered approach ensures data integrity and protects against unauthorized access.

Chapter 6

System Verification and Testing

This chapter presents the verification and validation procedures applied to assess both existing and newly implemented functionalities in the OpenHAB system, as well as an evaluation of the updated requirements.

The testing strategy followed a functionality-based approach. Each test case was structured around a specific function or feature to verify whether it behaved as expected. A test case included the functionality to be tested, the relevant input or interaction, the expected result, and the observed outcome. The result was marked as *passed* if the actual behavior matched the expected one, and *failed* otherwise.

Given the simplicity and repetitiveness of individual tests, they were grouped by functionality. For example, a test case might consist of turning off a television using the corresponding button in the OpenHAB interface and confirming both the change in device state and the updated status shown in the application. Since many test cases followed similar patterns, only the overall status of each group is presented.

6.1 User Interfaces Interaction Test

This test verified whether the graphical interface responded correctly to user interactions. It included actions such as clicking on various widgets, switching between interface pages, and viewing weather forecasts.

All buttons controlling devices were tested to ensure that:

- They correctly triggered the expected actions.
- The devices responded appropriately and within a reasonable delay.
- The application correctly reflected the device state, even when the device was manually changed outside OpenHAB.

Figure 6.1 – *Button interaction test*

The test was successful in terms of responsiveness and correctness. However, it was noted that the status update in OpenHAB, when devices were changed manually, experienced a slight delay.

6.2 Voice and External Application Control Test

This test evaluated integration with external voice assistants and device-specific applications.

In one scenario, a voice command was issued via Alexa to turn on the outdoor watering plug. The test confirmed that the plug status changed both in the Amazon Alexa app and in OpenHAB. While the functionality worked correctly, there was a noticeable delay in status synchronization.

Further tests involved controlling devices directly from their official applications (e.g., Amazon Alexa) to verify that changes were propagated back to OpenHAB. Again, synchronization was confirmed, although delays were observed.

6.3 Information Display Test

This test focused on validating the accuracy of data presented in OpenHAB from various sources, such as the weather API, local weather station, solar energy systems, Raspberry Pi system information, and astronomical data (sun and moon).

The displayed data was compared against reliable references—for instance, solar energy readings were compared to manufacturer dashboards. The aim was to assess whether the reported values were consistent with reality.

All data visualizations across interface pages, including temperatures, forecasts, solar generation, and system diagnostics, were evaluated. The results confirmed that the information shown was accurate and in line with expectations.

All functionality groups passed the testing procedures. While some minor delays were observed in synchronization—particularly with external applications and voice assistants—the core system operated reliably, and all interface elements behaved as intended.

6.4 Database Storage and Chart Visualization Test

This test evaluated whether the data generated by OpenHAB items was correctly stored in the InfluxDB database according to the configured persistence settings. It also verified that this data could be accurately retrieved and displayed in the OpenHAB charts.

To perform the test, the team compared the values shown in the chart of a specific item with the raw data entries in the corresponding table within the InfluxDB database.

Interactive Table View (press q to exit mode, shift+up/down to navigate tables):
Name: Weather_OWM_Wind_Speed

index	time	item	value
1	2025-05-14T19:57:55.36Z	Weather_OWM_Wind_Speed	4.8240000000
2	2025-05-14T18:51:48.555Z	Weather_OWM_Wind_Speed	9.6480000000
3	2025-05-14T18:42:47.849Z	Weather_OWM_Wind_Speed	1.6200000000
4	2025-05-14T17:42:47.044Z	Weather_OWM_Wind_Speed	23.2560000000
5	2025-05-14T16:51:05.088Z	Weather_OWM_Wind_Speed	1.6200000000
6	2025-05-14T15:51:04.79Z	Weather_OWM_Wind_Speed	14.8320000000
7	2025-05-14T12:51:03.741Z	Weather_OWM_Wind_Speed	8.0640000000
8	2025-05-14T11:51:03.431Z	Weather_OWM_Wind_Speed	17.0640000000
9	2025-05-14T10:51:02.927Z	Weather_OWM_Wind_Speed	15.3720000000
10	2025-05-14T09:51:02.541Z	Weather_OWM_Wind_Speed	13.8600000000

4 Columns, 10 Rows, Page 1/1
Table 1/1, Statement 1/1

Figure 6.2 – *Database value verification*

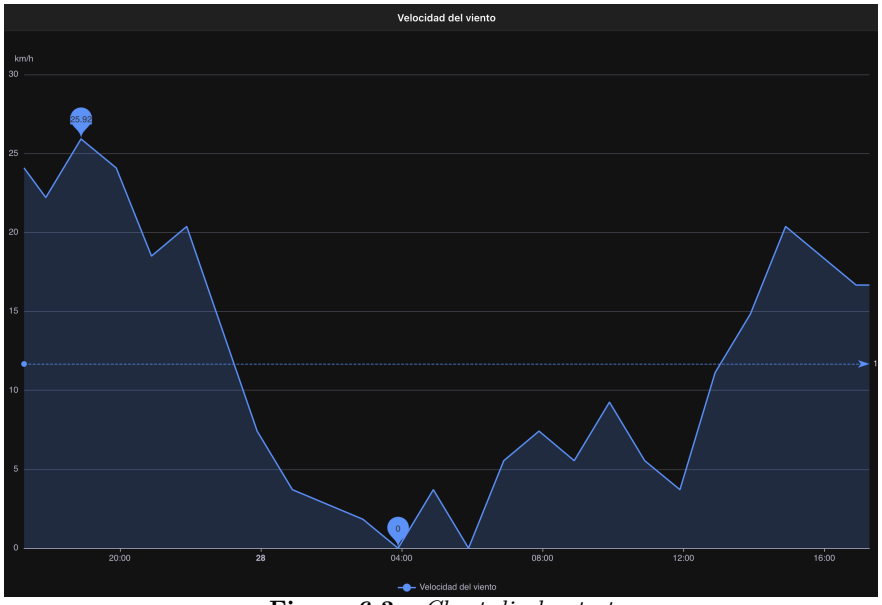


Figure 6.3 – *Chart display test*

The test was successfully passed, confirming the integrity of stored data and its correct visualization.

6.5 Amazon Alexa Player Integration Test

This test aimed to verify the functionality of the Amazon Alexa integration with OpenHAB, specifically for media playback control.

The evaluation involved requesting Alexa to play a song and then checking if the song title, volume level, and playback progress could be monitored and adjusted from both the OpenHAB interface and the Amazon Alexa app.

The integration worked as expected, and the test was passed successfully.

6.6 Backup Automation and Restoration Test

This test confirmed the correct functioning of the system's backup processes, which were configured both via Duplicati and through a custom script scheduled with `cron`.

It was verified that backups were executed automatically at the configured intervals and that restoration procedures reliably recovered the system's data and configurations.

Both backup creation and restoration were successful, and the test was passed.

6.7 Rule Testing

This section presents the testing process and results related to the rules configured in the OpenHAB system. OpenHAB provides functionality to manually execute rules, which facilitates testing and debugging. To access this feature, one must navigate to *Settings* → *Rules*, and within each rule, use the "Run now" option (shortcut: `Ctrl + R`) to trigger the rule manually.

The first stage of testing involved verifying that each rule was correctly triggered based on its defined trigger type, which could be a cron schedule, an item change, or a channel event. Once it was confirmed that all rules responded to their respective triggers, the second step involved verifying that each rule executed its intended actions. At this point, the manual execution feature (`Ctrl + R`) was used extensively.

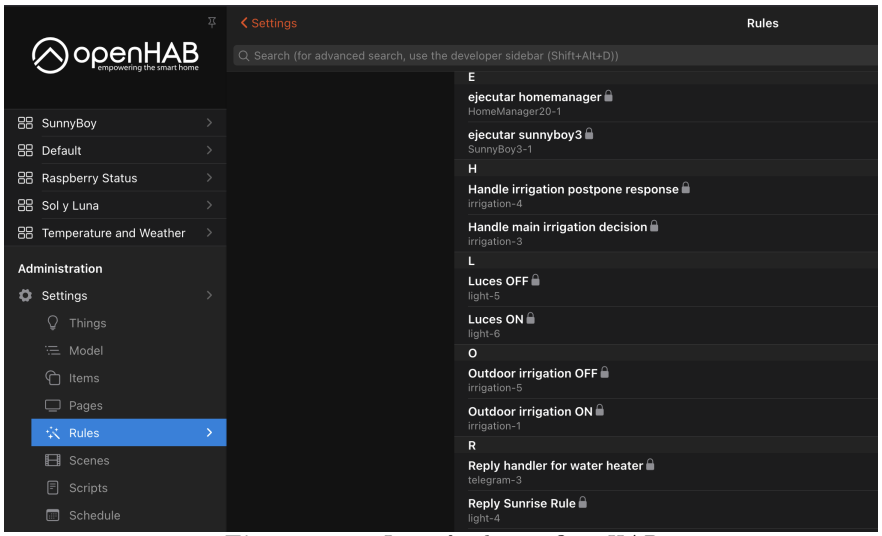


Figure 6.4 – *List of rules in OpenHAB*

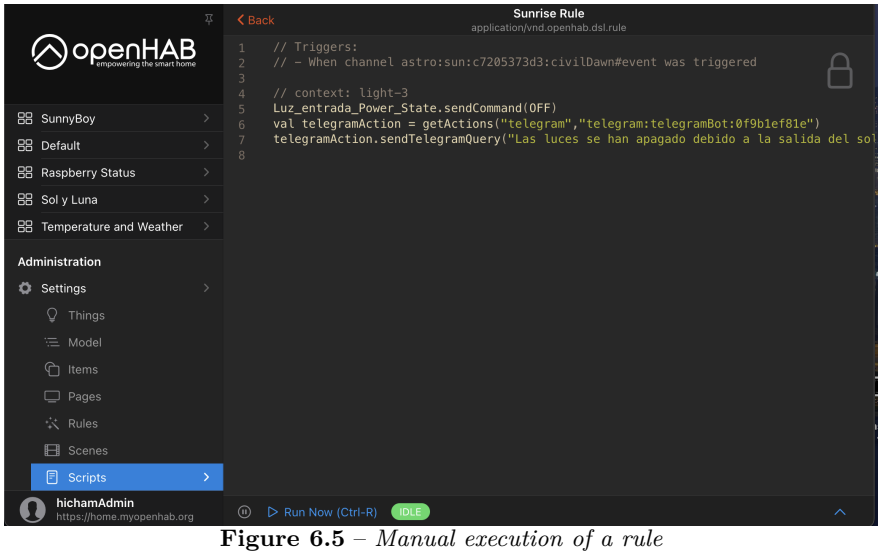


Figure 6.5 – *Manual execution of a rule*

6.7.1 Sunny Boy and Home Manager Rules Test

1. **Sunny Boy Items Update:** This test verified that the rule correctly updated the item values for the Sunny Boy inverter. The log output was compared with the item values in the system, and consistency was confirmed.

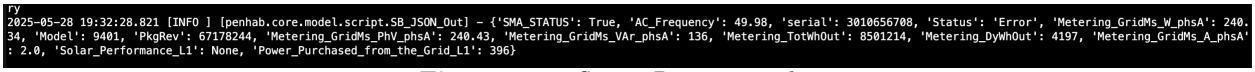


Figure 6.6 – *Sunny Boy item updates*

Test passed.

2. **Home Manager Items Update:** Similarly, the test for Home Manager rules verified that the item values were updated according to the values reflected in the system logs.

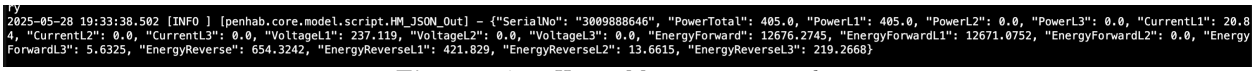


Figure 6.7 – *Home Manager item updates*

Test passed.

3. **Sunny Boy Telegram Notifications:** The system was tested for sending Telegram notifications when a connection error with the Sunny Boy device occurred. Notifications were correctly sent in both error and normal scenarios.

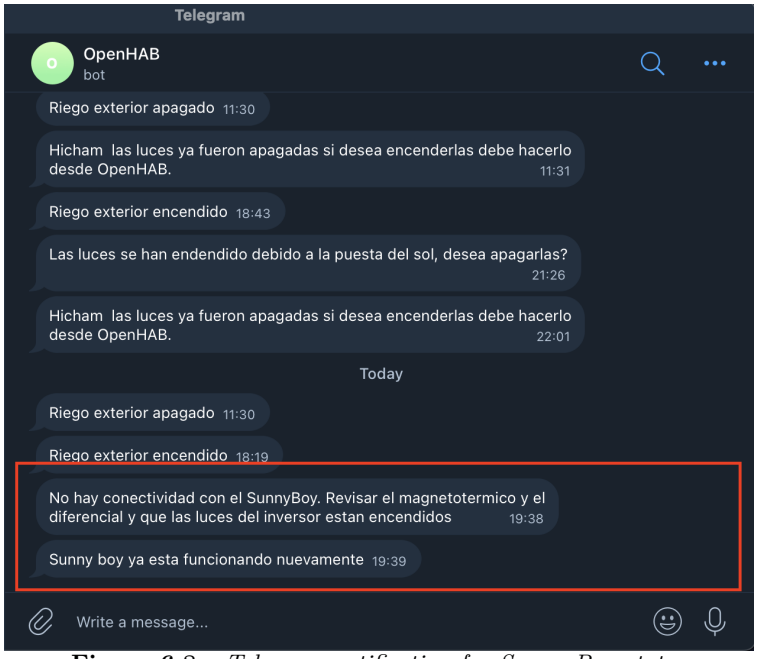


Figure 6.8 – *Telegram notification for Sunny Boy status*

Test passed.

4. **Sunny Bo WhatsApp Notifications:** The same test was performed for WhatsApp notifications. The system correctly sent messages regarding the Sunny Boy connection status.

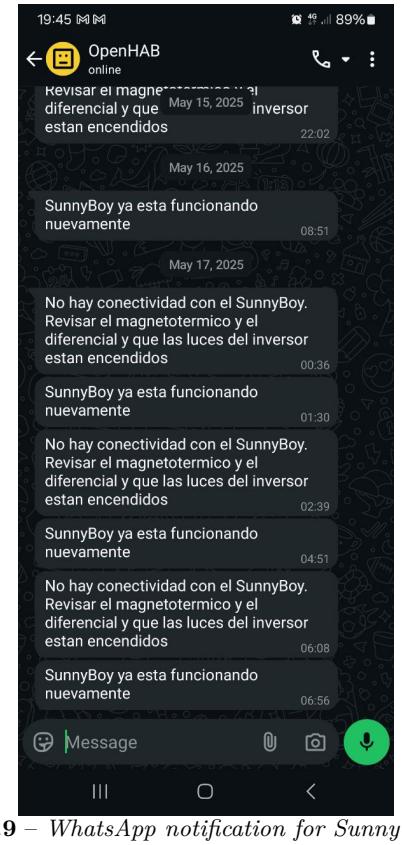


Figure 6.9 – *WhatsApp notification for Sunny Boy status*

Test passed.

6.7.2 Water Heater Rules Test

This subsection presents the results of testing the automation rules related to the water heater. These rules are responsible for notifying users about the water heater's status and handling user responses for control actions.

1. **Water Heater Off Notification:** A test was conducted to verify that the system sends a notification when the water heater is turned off. The expected notification was successfully received.

Figure 6.10 – *Water heater off notification*

Test passed.

2. **Water Heater On Notification and Confirmation Handling:** When the water heater was turned on, the system sent a corresponding notification and a request for confirmation to turn it off.

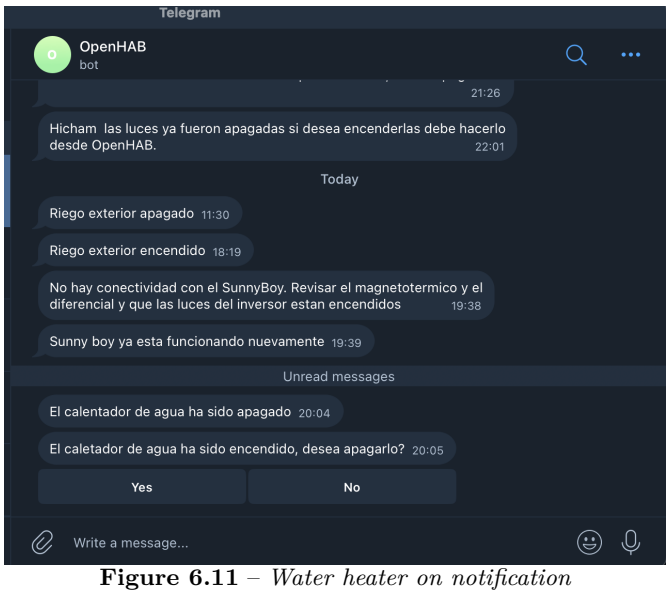


Figure 6.11 – *Water heater on notification*

Test passed.

(a) **User Response: "Yes", Water Heater Still On:** When the user responded with "yes" and the water heater was still on, the system turned off the device and confirmed the action via notification.

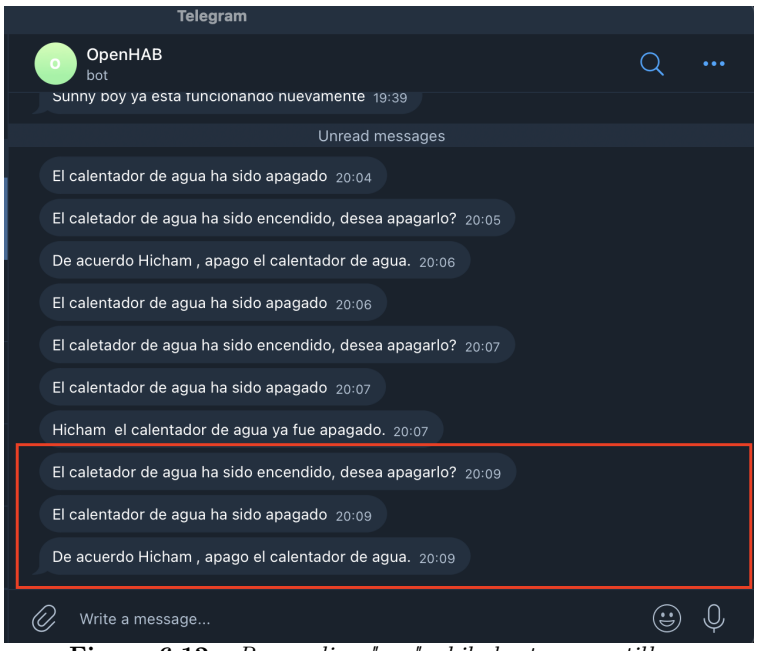


Figure 6.12 – *Responding "yes" while heater was still on*

Test passed.

(b) **User Response: "Yes", Water Heater Already Off:** If the user responded "yes" while the water heater had already been turned off manually via the app, the system correctly notified that it was already off.

Figure 6.13 – *Responding "yes" while heater was already off*

Test passed.

(c) **User Response: "No", Water Heater Still On:** When the user responded "no" and the heater was still on, the system left it on and notified the user of this decision.

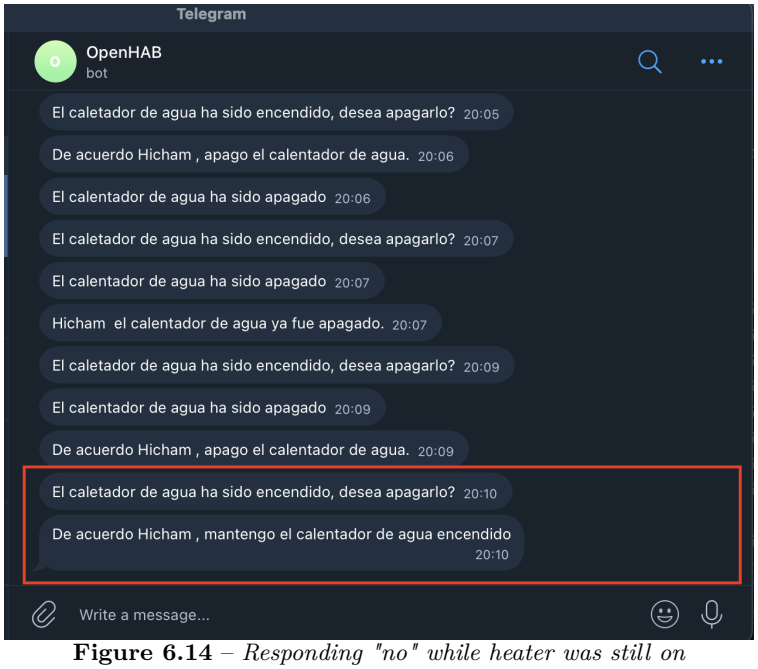


Figure 6.14 – *Responding "no" while heater was still on*

Test passed.

(d) **User Response: "No", Water Heater Already Off:** If the heater had already been turned off through the app and the user still responded "no," the system sent a notification indicating that the device was already off and suggested using the app to turn it back on if desired.

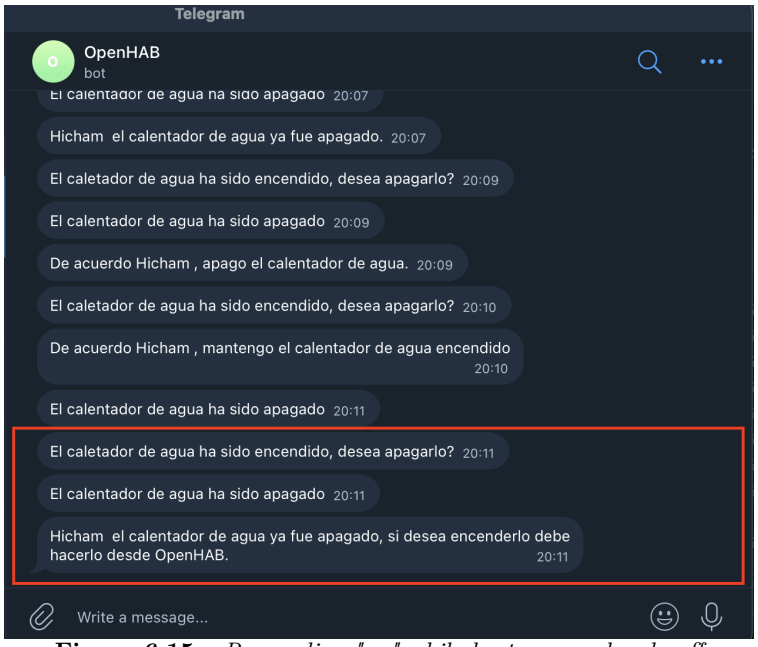


Figure 6.15 – *Responding "no" while heater was already off*

Test passed.

6.7.3 Lights Rules Test

This subsection outlines the verification of automation rules associated with lighting control, particularly those triggered at sunset and sunrise. These rules are designed to either control lighting behavior or notify the user, offering confirmation options for further actions.

1. **Sunset Lighting Rule:** At sunset, the system is configured to either automatically turn on the lights or send a notification informing that the lights are on, along with confirmation options. The rule functioned as expected by activating the lights and sending the correct message.

   (a) **User Response: "Yes" to Turn Off Lights:** When the user responded "yes" to turn off the lights, the system behaved correctly in both scenarios—whether the lights were already turned off via the app (resulting in a message acknowledging this) or still on (in which case they were turned off).

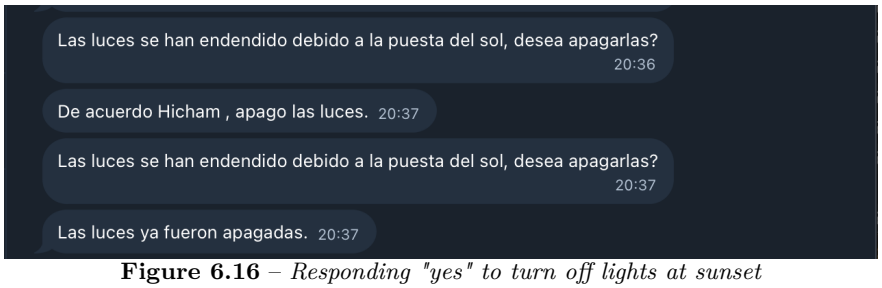


Figure 6.16 – *Responding "yes" to turn off lights at sunset*

Test passed.

   (b) **User Response: "No" to Turn Off Lights:** If the user responded "no," the system behaved as intended, either confirming that the lights remained on or, if they had already been turned off manually, sending a message reflecting that status.

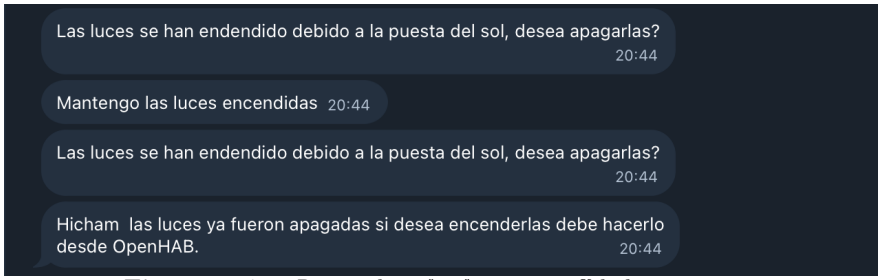


Figure 6.17 – *Responding "no" to turn off lights at sunset*

Test passed.

2. **Sunrise Lighting Rule:** At sunrise, the system is designed to either automatically turn off the lights or notify the user that the lights are off, including confirmation options. The rule performed as expected by turning off the lights and sending appropriate notifications.

   (a) **User Response: "Yes" to Keep Lights Off:** When the user responded "yes," the system behaved correctly, sending a message depending on whether the lights had remained off or had been turned on manually.

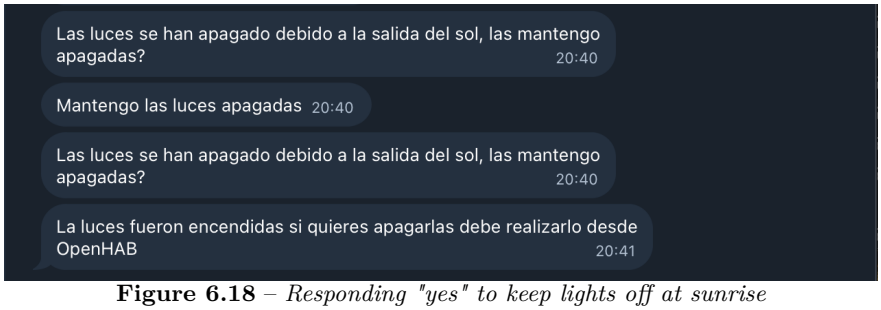


Figure 6.18 – *Responding "yes" to keep lights off at sunrise*

Test passed.

(b) **User Response: "No" to Keep Lights Off:** A response of "no" resulted in the expected behavior: the system either turn on lights or noted if they had already been turned on manually.

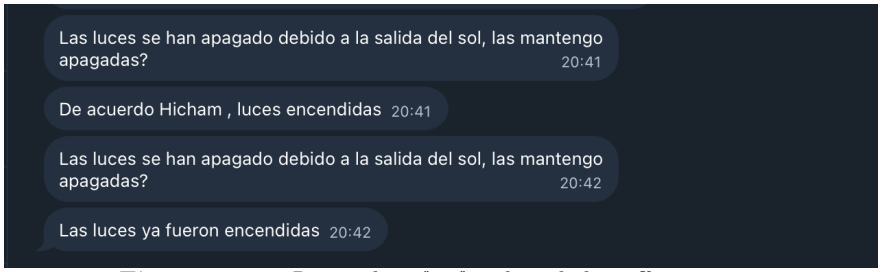


Figure 6.19 – *Responding "no" to keep lights off at sunrise*

Test passed.

6.7.4 Irrigation Rules Test

This subsection presents the evaluation of the irrigation automation rules. The system is configured to automatically turn on irrigation at 10:30 and turn it off at 11:30, unless the irrigation is already active or inactive, in which case no action or notification is triggered—unlike the behavior of the lighting rules.

1. **Irrigation Activation at 10:30:** At 10:30, the system is expected to turn on the irrigation system and send a notification with confirmation options. The rule was executed correctly, with the irrigation being activated and the notification being sent.

Figure 6.20 – *Irrigation turned on at 10:30*

Test passed.

(a) **User Response: "Yes" to Turn Off Irrigation:** When the user selected "yes" to turn off irrigation, the system responded correctly in both cases: turning off the irrigation if it was still on, or acknowledging that it had already been turned off manually.

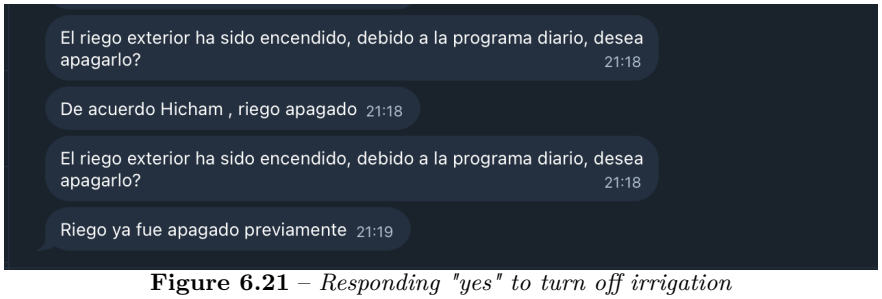


Figure 6.21 – *Responding "yes" to turn off irrigation*

Test passed.

(b) **User Response: "No" to Turn Off Irrigation:** The system also handled "no" responses correctly, maintaining the irrigation status as intended or sending a message based on the irrigation was turned off through app.

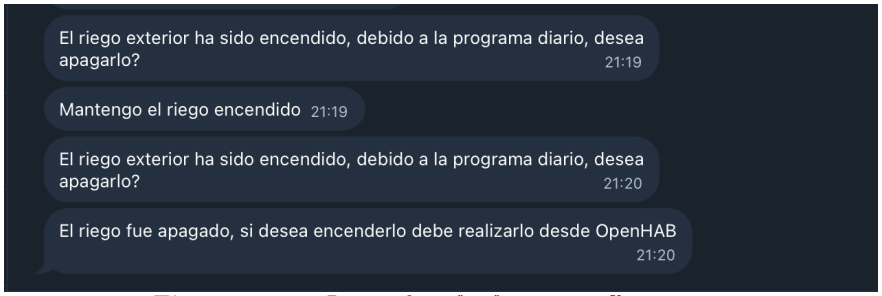


Figure 6.22 – *Responding "no" to turn off irrigation*

Test passed.

(c) **User Response: "Postpone":** When "postpone" was selected, the system returned a menu of postponement options. Upon selecting specific days, the system confirmed the postponed schedule and rescheduled the irrigation accordingly.

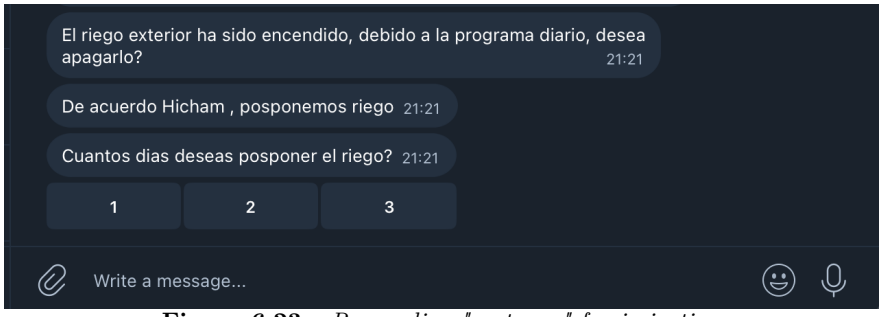


Figure 6.23 – *Responding "postpone" for irrigation*

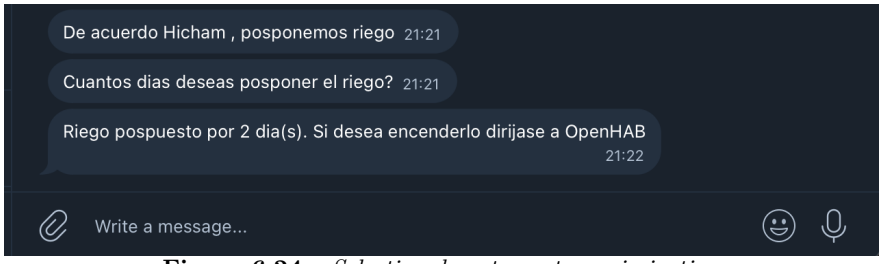


Figure 6.24 – *Selecting days to postpone irrigation*

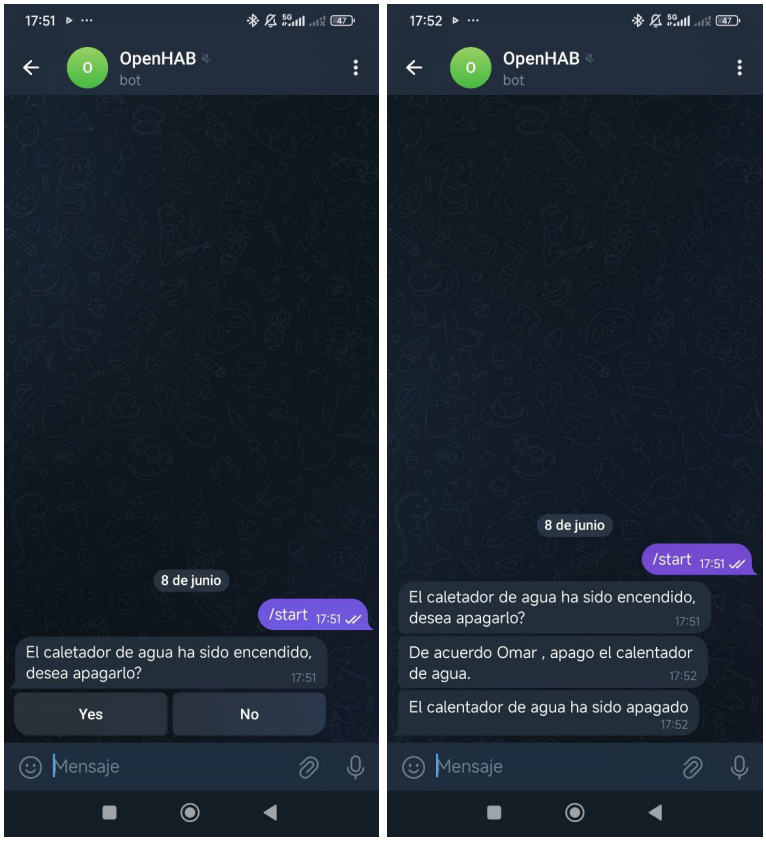
Test passed.

2. **Irrigation Deactivation at 11:30:** At 11:30, the system is configured to automatically turn off the irrigation unless it is already off. This rule executed as expected, deactivating the irrigation system without redundant notifications.

   Test passed.

6.7.5 Replier Recognition

The user (replier) was identified in every interaction with the OpenHAB Telegram bot. The bot responded by including the user's name in its replies, thereby making the interaction more personalized and traceable.



(a) *Water heater ON notification* **(b)** *Responding with "yes"*

Figure 6.25 – *Replier recognition via Telegram bot*

The test was successfully passed.

6.8 Evaluation of Satisfaction of Requirements

This section presents an evaluation of the system's compliance with the defined functional requirements. Each requirement is reviewed to determine whether it has been successfully fulfilled by the implemented features. Requirements of previous project are not reiterated, as their analysis remains unchanged.

Requirement	FR.1 - The system shall display information from the Sunny Boy solar inverter and the Home Manager energy meter in a dedicated tab
Section	Pages
Analysis	Requirement satisfied through the creation of a dedicated page displaying data from the Sunny Boy and Home Manager, including variables, charts, and diagrams illustrating energy flow.
Evaluation	Validated

Table 6.1 – *FR 1. Satisfaction Evaluation*

Requirement	FR.2 - The system shall display temperature readings, weather forecasts, and sun & moon information in a dedicated tab
Section	Pages
Analysis	Requirement satisfied through the implementation of two pages: one showing temperature readings and weather forecasts, and another displaying sun and moon information.
Evaluation	Validated

Table 6.2 – *FR 2. Satisfaction Evaluation*

Requirement	FR.3 - The system shall present Raspberry Pi information in a dedicated tab
Section	Pages
Analysis	Requirement satisfied through the creation of a page displaying system information related to the Raspberry Pi, including CPU, memory, and storage usage.
Evaluation	Validated

Table 6.3 – *FR 3. Satisfaction Evaluation*

Requirement	FR.4 - The user shall receive notifications about the solar inverter status via Telegram and WhatsApp
Section	Rules
Analysis	Requirement satisfied through rules that trigger when the Sunny Boy inverter status changes, sending a message about status.
Evaluation	Validated

Table 6.4 – *FR 4. Satisfaction Evaluation*

Requirement	FR.5 - The user shall receive notifications about the outdoor irrigation status via Telegram
Section	Rules, Telegram Binding
Analysis	Requirement satisfied through rules that trigger when the status of the item controlling irrigation changes.
Evaluation	Validated

Table 6.5 – *FR 5. Satisfaction Evaluation*

Requirement	FR.6 - Lights shall automatically turn on at sunset and off at sunrise
Section	Rules
Analysis	Requirement satisfied through the creation of rules triggered by channel events indicating sunset and sunrise.
Evaluation	Validated

Table 6.6 – *FR 6. Satisfaction Evaluation*

Requirement	FR.7 - The user shall receive notifications about the lights' status via Telegram
Section	Rules, Telegram Binding
Analysis	Requirement satisfied through rules triggered by the item that controls lighting, which send notifications accordingly.
Evaluation	Validated

Table 6.7 – *FR 7. Satisfaction Evaluation*

Requirement	FR.8 - Upon receiving a notification from the lighting system, users shall be able to interact with a Telegram bot to query the current status or change the state
Section	Rules, Telegram Binding
Analysis	Requirement satisfied through rules and Telegram binding features that enable interaction to retrieve or modify the lighting state.
Evaluation	Validated

Table 6.8 – *FR 8. Satisfaction Evaluation*

Requirement	FR.9 - The user shall receive notifications about the water heater status via Telegram
Section	Rules
Analysis	Requirement satisfied through rules triggered by the item controlling the water heater, which send appropriate notifications.
Evaluation	Validated

Table 6.9 – *FR 9. Satisfaction Evaluation*

Requirement	FR.10 - Upon receiving a notification that the water heater is on, users shall be able to interact with a Telegram bot to query the current status or change the state
Section	Rules, Telegram Binding
Analysis	Requirement satisfied through rules and Telegram binding features that support user interaction for querying and modifying the water heater state.
Evaluation	Validated

Table 6.10 – *FR 10. Satisfaction Evaluation*

Requirement	FR.11 - Irrigation shall automatically turn on at 10:30 and off at 11:30, and send a notification via the Telegram bot
Section	Rules, Telegram Binding
Analysis	Requirement satisfied through cron-based rules and Telegram binding functions that send notifications accordingly.
Evaluation	Validated

Table 6.11 – *FR 11. Satisfaction Evaluation*

Requirement	FR.12 - Upon receiving a notification about irrigation being active, users shall be able to interact with a Telegram bot to query the current status, postpone the automation, or change the irrigation state
Section	Rules, Telegram Binding
Analysis	Requirement satisfied through the use of Telegram bot functions and rules that handle user interaction, including querying, postponing, or altering irrigation status.
Evaluation	Validated

Table 6.12 – *FR 12. Satisfaction Evaluation*

Requirement	FR.13 - The system shall identify the last person who replied to the Telegram notification, in order to interact only with them
Section	Telegram Binding
Analysis	Requirement satisfied through the use of a Telegram Thing bot item that stores and identifies the last user who replied.
Evaluation	Validated

Table 6.13 – *FR 13. Satisfaction Evaluation*

Requirement	FR.14 - The system shall allow authenticated and verified administrator users to modify or disable automations related to lights, irrigation, and notifications
Section	Installation
Analysis	Requirement satisfied through authentication mechanisms that restrict configuration changes to administrator-level users.
Evaluation	Validated

Table 6.14 – *FR 14. Satisfaction Evaluation*

Requirement	FR.15 - Users shall be able to access the system through the mobile app, with the same functionalities available as on the web interface
Section	OpenHAB Cloud and OpenHAB Mobile App
Analysis	Requirement satisfied through the use of OpenHAB Cloud, which enables access from devices outside the local network with feature parity between mobile and web interfaces.
Evaluation	Validated

Table 6.15 – *FR 15. Satisfaction Evaluation*

The non-functional requirements were also evaluated during development. However, since these do not require a detailed breakdown of implementation, they are not included in this section. For a full description, please refer to the section dedicated to non-functional requirements.

Chapter 7

Conclusions, Future Work and Lessons Learned

7.1 Conclusions

This document presents the process followed to update and enhance a home automation system based on OpenHAB, adding new functionalities, fixing existing problems, improving the infrastructure for better maintainability, and enhancing the user experience when interacting with the system.

Although projects that improve or continue previous work may appear easier at first glance, they are often more complex. Improving an existing system requires first understanding all the previous work, identifying problems, and then implementing effective solutions. In this sense, it often feels like doing double the work, but the learning outcomes are far more substantial. Progress in knowledge and technology is rarely the result of isolated efforts—it is built upon many previous contributions.

Understanding the documentation was relatively straightforward, but applying it in practice proved difficult. It took time to connect the concepts and understand the system's philosophy and operation in order to successfully fix and improve it. When you work on a system you didn't create, it's common to find that fixing one issue can cause another.

Although the final result may seem modest, developing this project was especially challenging given that I was working full-time at the same time. It is commonly said that "turning it off and on again" is enough in IT, but when your Bachelor's thesis depends on the system working correctly, that approach is insufficient. You must understand exactly why something is not working. While this process is often frustrating, it leads to deep satisfaction when you finally succeed—even if the problem was caused by something seemingly simple. What matters is that you persisted until you understood it.

One of the most valuable aspects of this project was the opportunity to work intensively with Docker. At my company, Docker is used extensively, and this project gave me a much deeper understanding of it. I learned about Docker layers, image efficiency, caching mechanisms, image corruption, and issues such as why deleting a container or image does not always behave as expected. Understanding these aspects was crucial for diagnosing and fixing errors quickly and effectively.

This project also consolidated my understanding of infrastructure and software architecture. I had to connect databases to APIs, APIs to cloud services, and correctly configure ports and services—gaining hands-on experience in system integration.

Finally, I would like to express my sincere gratitude to Andrés for his invaluable support throughout this project. Both the development process and the final results have been highly satisfying—technically and personally—and would not have been possible without his guidance. The system successfully met its objectives and delivered the intended functionality. Furthermore, I acquired new technical and soft skills, particularly through my interactions with the GranaSAT laboratory, the client, and the engineers involved, from whom I learned a great deal.

7.2 Future Work

Several improvements were not implemented due to limitations in time and resources. These represent interesting directions for future work:

- Deploy the system on AWS ECS using ECR. This would provide greater scalability and reliability through cloud infrastructure, eliminating the need for OpenHAB Cloud and allowing remote access via a secure and robust platform.
- Add various sensors and create automations based on their states. For example, turning lights on when motion is detected, or using humidity and smoke detectors to improve safety and automation.
- Integrate smoke detectors and set up automatic alerts via Telegram or WhatsApp when smoke is detected.
- Enhance the user interface by including elements that display which user responded to a Telegram query, what the response was, and what actions were taken.
- Add cameras to the system and manage them through OpenHAB bindings, including displaying live video feeds directly within the OpenHAB interface.

7.3 Lessons Learned

This project has been a valuable learning experience that helped me grow both technically and personally. Some of the key lessons learned include:

- Learning how to professionally manage a project using software engineering methodologies.
- Developing the ability to analyze the current state of a system and plan improvements and updates accordingly.
- Gaining foundational knowledge of home automation systems, including communication protocols, integration methods, and device compatibility.
- Expanding my understanding of the Internet of Things (IoT), networking, and system connectivity.
- Deepening my knowledge of software architecture and infrastructure design.
- Acquiring advanced skills in Docker, including image creation, container management, troubleshooting, and optimization.
- Working with databases—particularly time-series databases—and understanding their applications within a home automation context.
- Learning how to plan and execute system and database migrations.
- Adopting agile methodologies such as iterative development and sprints to manage progress effectively.

7

Bibliography

[1] Openhab. https://www.openhab.org/.

[2] AMAZON. Docker images vs containers. https://aws.amazon.com/compare/the-difference-between-docker-images-and-containers/.

[3] AMAZON, A. What is docker. https://aws.amazon.com/es/docker/.

[4] CLOUDBEES. Dockerfile. https://www.cloudbees.com/blog/what-is-a-dockerfile.

[5] DEV.TO. Api info. https://dev.to/develawyer/que-son-api-s-para-dummies-2adj#:~:text=La%20interfaz%20de%20programaci%C3%B3n%20de,software%20como%20una%20capa%20de.

[6] DOCKER. Docker-compose. https://docs.docker.com/compose/.

[7] DOCKERDOCS. Docker best practices. https://docs.docker.com/build/building/best-practices/.

[8] DOCKERDOCS. Docker image layers. https://docs.docker.com/get-started/docker-concepts/building-images/understanding-image-layers/.

[9] DOCKERDOCS. How compose works. https://docs.docker.com/compose/intro/compose-application-model/.

[10] DOCKERDOCS. Why use compose? https://docs.docker.com/compose/intro/features-uses/.

[11] DOCKERDOCS. Why use docker compose? https://docs.docker.com/guides/docker-compose/why/.

[12] DUPLICATI. Duplicati. https://duplicati.com/.

[13] ESTACIONESMETEOROLOGICAS. Weather station information. https://www.estacionesmeteorologicas.top/blog/que-es-y-para-que-sirve-una-estacion-meteorologica/.

[14] FOUNDATION, O. Openhab foundation. www.openhabfoundation.org.

[15] HARTMAN, J. What is java. https://www.guru99.com/java-platform.html.

[16] HOSTGATOR. Virtual assistant definition. https://www.hostgator.mx/blog/alexa-siri-asistente-virtual-inteligente/definicionasistentevirtual.

[17] HOSTINGER. Ssh. https://www.hostinger.es/tutoriales/que-es-ssh#:~:text=SSH%20o%20Secure%20Shell%2C%20es,de%20un%20mecanismo%20de%20autenticaci%C3%B3n.

[18] INFLUX. Influx glossary. https://docs.influxdata.com/influxdb/v1/concepts/glossary.

[19] INFLUX. Influxdb. https://docs.influxdata.com/influxdb/.

[20] Influx. Influxdb glossary oss v2. https://docs.influxdata.com/influxdb/v2/reference/glossary/.

[21] Influxdata. Influxdb internals. https://docs.influxdata.com/influxdb/v2/reference/internals/.

[22] Influxdata. Influxdb syntax. https://docs.influxdata.com/influxdb/v2/reference/syntax/flux/flux-vs-influxql/.

[23] Influxdata. Influxdb upgrade. https://docs.influxdata.com/influxdb/v2/install/upgrade/v1-to-v2/automatic-upgrade/.

[24] Influxdata. Influxdb v1 authentication. https://docs.influxdata.com/influxdb/v1/administration/authentication_and_authorization/.

[25] Influxdata. Influxdb v1 configuration. https://docs.influxdata.com/influxdb/v1/administration/config/#dir--varlibinfluxdbdata.

[26] Influxdata. Influxdb v2 set up. hhttps://docs.influxdata.com/influxdb/v2/get-started/setup/.

[27] Influxdata. Influxdb wal. https://docs.influxdata.com/influxdb/v2/reference/internals/storage-engine/#write-ahead-log-wal.

[28] Legaza, F. J. J. Single-family home automation portal based on openhab. Bachelor's Thesis, University of Granada, Granada, September 2023. https://digibug.ugr.es/handle/10481/80478.

[29] Medium. Influxdb backup and restore. https://medium.com/@creil/backup-and-restore-influxdb-database-into-a-docker-container-932b1468b2cf.

[30] Nerdomus. Virtual assistant definition 2. https://nerdomus.com/googlehome-vs-alexa/paraquesirvenlosasistentesvirtuales.

[31] OpenHAB. Definition states things. https://www.openhab.org/docs/concepts/things.html#thing-status.

[32] OpenHAB. Openhab cloud. https://www.openhab.org/addons/integrations/openhabcloud/.

[33] OpenHAB. Openhab persistence. https://www.openhab.org/docs/configuration/persistence.html.

[34] OpenHAB. Openhab ui. https://www.openhab.org/docs/ui/.

[35] OpenHAB. State diagram. https://www.openhab.org/docs/concepts/things.html#status-transitions.

[36] Openhab. Websocket in openhab. https://www.openhab.org/docs/configuration/websocket.html.

[37] OpenHAB. What is openhab cloud. https://github.com/openhab/openhab-cloud/blob/master/README.md.

[38] Oracle. Internet of things. https://www.oracle.com/in/internet-of-things/what-is-iot/.

[39] org, R. Rclone. https://rclone.org/.

[40] Raspberry PI 5, 2024. Available at https://arstechnica.com/gadgets/2024/01/what-i-learned-from-using-a-raspberry-pi-5-as-my-main-computer-for-two-weeks/.

[41] Raspberrypi, 2024. Available at https://www.raspberrypi.com/documentation/computers/getting-started.html.

[42] Redhat. What-is-a-rest-api. https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[43] Rincondomotica. Openhab. https://rincondomotica.com/openhab-la-mejor-plataforma-domotica-open-source.

[44] SMA. Energy meter info. https://www.sma.de/es/productos/monitorizacion-y-control/sma-energy-meter.

[45] SMA. Sunny boy info. https://www.sma.de/es/productos/inversor-fotovoltaico/sunny-boy-30-36-40-50-60.

[46] Techtarget. What is docker. https://www.techtarget.com/searchitoperations/definition/Docker.

[47] teradata. What is python. https://www.teradata.com/Glossary/What-is-Python.

[48] Tmate. Tmate. https://tmate.io/.

[49] Triwicaksono, M. Networks in compose. https://medium.com/@triwicaksono.com/networks-in-docker-compose-0943abe3de54.

[50] Wikipedia. Api definition. https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones.

[51] Wikipedia. Http. https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

[52] Wikipedia. Solar inverter info. https://es.wikipedia.org/wiki/Inversor_fotovoltaico.

[53] Wikipedia. Time series db. https://en.wikipedia.org/wiki/Time_series_database.

[54] Wikipedia. Websocket. https://en.wikipedia.org/wiki/WebSocket.

[55] Wikipedia. What is raspberry pi. https://es.wikipedia.org/wiki/Raspberry_Pi.

[56] Wikipedia. Wifi. https://en.wikipedia.org/wiki/Wi-Fi.

[57] Wikipedia. Zigbee. https://en.wikipedia.org/wiki/Zigbee.

[58] Youtube. Raspberry pi 5 installation. https://youtu.be/nOxywVabojk.