Monitorización automática de condiciones ambientales para espacios cerrados

---

Automatic monitoring of environmental conditions in indoor spaces

---

TRABAJO DE FIN DE GRADO

Curso 2025-2026

Roi López Barata

Directores:
Daniel Báscones García
Juan Carlos Fabero Jiménez

Calificación: 10 - SB

Grado en Ingeniería Informática

Facultad de Informática
Universidad Complutense de Madrid

25 de Mayo del 2026

A mis padres, por demostrarme con su ejemplo lo que implica el sacrificio.
A mi hermano, por haberme guiado hasta aquí.
A mí, por todo lo demás.

To my parents, for showing me through their example what sacrifice means.
To my brother, for guiding me to this point.
To me, for everything else.

Agradecimientos

En primer lugar, me gustaría agradecer a mi familia su amor y apoyo incondicional durante todos estos años. Nunca hubiera llegado aquí sin ellos.

Por supuesto, tengo que agradecer también a los dos profesores que han sido mis tutores de este TFG. Daniel y Juan Carlos, muchas gracias por vuestra ayuda, vuestros consejos y vuestra paciencia.

No puedo olvidarme de compañeros y amigos que me han acompañado durante tantos años. Nombrarlos a todos es casi imposible, aunque una buena aproximación incluiría a **Antón, Iago, Irene, Marcos, Alfonso, Cristian, Belén, Xiana y a mis compañeros del CMU Chaminade**. Un grupo aparte son mis compañeros de clase de estos útlimos años, quienes me han ayudado y acogido como a uno más y a los que tengo especial cariño. **Tere, Sofía, Rocío, Jorge, Adri (los dos), David, Natalia, Miguel, Sandra, Carlos**, sois geniales.

Algunos de los profesores que he tenido a lo largo de mi vida han tenido mucho que ver en que esté escribiendo estas líneas. Muchos de ellos han tenido una gran influencia en mí, y desde luego no sería como soy si no fuese por ellos. Y, asumiendo el riesgo de olvidarme de alguno, merecen un reconocimiento especial **Toño, Maribel, Dª Carmelita, Dª María del Carmen, Marcos, Xoán, Darío, Doval, Emilia, David, Diego, Gabriel, Dª Carmen Llano, Cristina, Dª Olga, Robert, Luis Hervella, Ana Vieites, Óscar, Rafael, Alejandro.**

Gracias también a todo el equipo de TheirStack por darme la oportunidad de trabajar en un SaaS y tener la paciencia suficiente para enseñarme tantas lecciones de tantos ámbitos distintos.

Por (casi) último, que no falten aquellas personas que me han hecho daño o causado algún perjuicio a lo largo de estos años. Siempre sin rencor, pero nunca sin memoria, no olvido cuánto aprendí de mí mismo gracias a ell@s. Lo que no te mata...

No podría concluir esta breve lista sin una mención afectuosa a mi hermano, **Xoel**. Él me ha ayudado y guiado desde que era un niño, y gracias a él he tenido algunas de las experiencias más increíbles de mi vida.

En definitiva, infinitas gracias a todo@s los que habéis sido parte de este camino y en algún momento me hayáis ayudado de una forma u otra.

Sen esforzo non hai aprendizaxe.

Resumen

El objetivo de este proyecto es crear un sistema de monitorización ambiental interior asequible y listo para su puesta en producción, basado en una red de estaciones de sensores interconectadas. Ya existen dispositivos similares en el mercado, pero, o bien tienen capacidades muy limitadas, o están orientados al ámbito industrial, por lo que resultan demasiado costosos para el ciudadano medio.

Cada estación contiene un conjunto de sensores que realizan lecturas de diferentes variables ambientales, como la temperatura, la humedad, la presión atmosférica o las concentraciones de varios gases, por ejemplo monóxido de carbono o metano. Las estaciones también incluyen componentes adicionales, como un zumbador que reproduce una melodía específica cuando se detecta un riesgo ambiental a partir de las lecturas de los sensores, o una pequeña pantalla donde el usuario puede ver la última medición tomada de los distintos parámetros ambientales registrados.

Existen dos tipos de estaciones: inalámbricas y centrales. Las estaciones inalámbricas funcionan únicamente con la energía aportada por su batería, proporcionan solo lecturas ambientales básicas y se asocian con estaciones centrales mediante enlaces de radio cifrados. Las estaciones centrales, por otro lado, están conectadas a una toma externa de corriente, ofrecen una amplia gama de lecturas (incluyendo concentraciones de diferentes gases) y actúan como enlace entre las estaciones inalámbricas y el resto del sistema.

Con el fin de aumentar aún más la usabilidad del sistema, se ha desarrollado en paralelo a los firmwares de las estaciones una aplicación web completa. En esta aplicación, los usuarios pueden registrar estaciones a su cuenta y acceder a las lecturas ambientales —actuales y pasadas— de todas sus estaciones. Para ello, la aplicación genera gráficas interactivas que muestran los valores registrados de cada parámetro ambiental.

El resultado de este Trabajo de Fin de Grado es un sistema integral de monitorización interior que incluye todas las funcionalidades de un producto comercial de Internet de las Cosas (IoT), desde la recolección, procesamiento y transmisión de datos hasta el desarrollo de una aplicación de gestión para el usuario final.

Palabras clave: Internet de las Cosas (IoT), monitorización ambiental interior, redes de sensores inalámbricos, sistemas embebidos, monitorización de la calidad del aire, aplicación web, visualización de datos en tiempo real, detección remota.

Abstract

The goal of this project is to create an affordable and production-grade indoor environmental monitoring system based on a network of interconnected measuring stations. Similar devices are already available commercially, but they either have very limited capabilities or are industrially-focused and are therefore too costly for the average citizen.

Each station contains a set of sensors that take readings of different environmental variables, such as temperature, humidity, air pressure or the concentrations of several gases, for example carbon monoxide or methane. Stations also include additional components, such as a buzzer that plays a distinctive melody whenever an environmental hazard derived from the sensor readings is detected, or a small display where the user can see the latest taken measurement of the different environmental parameters read.

There are two types of stations: wireless stations and central stations. Wireless stations operate solely on battery life, provide only basic environmental readings and are associated with central stations through encrypted radio links. Central stations, on the other hand, are connected to a power outlet, offer a wide range of readings (including concentrations of different gases) and act as the link between wireless stations and the rest of the system.

In order to further increase the usability of the system, an entire web application has been developed parallel to the stations' firmware. In this app, users can register stations with their account and access the current and past environmental readings of all their stations. To do that, the application renders interactive plots that display the recorded values of every environmental parameter.

The result of this thesis is an integral indoor monitoring system featuring all the functionalities of a commercial Internet of Things (IoT) product, from data collection, processing and transmission to the development of a management application for the end user.

Keywords: Internet of Things (IoT), indoor environmental monitoring, wireless sensor networks, embedded systems, air quality monitoring, web application, real-time data visualization, remote sensing.

Contents

List of Figures

Chapter 1

Introduction

1.1 History of indoor environmental management

For millennia, indoor air quality was severely overshadowed by thermal comfort as the main priority of the population. Approximately 1.5 million years ago, when early humans began using campfires [17], the primary purpose of starting a fire was to provide a heat source, not for comfort, but purely for survival in a hostile environment. Even if the air around the fire source became heavily polluted, breathing it for a few hours was still a better alternative than dying of hypothermia or being attacked by a predatory animal, both realistic situations that could occur without a campfire.

Fast forward several hundred thousand years, Romans integrated existing heating techniques [28] and created **the hypocaust** (*Hypocaustum*) [17], an underfloor heating system used to warm their public hot baths and the villas of the wealthier individuals. In some hypocausts, walls would have inner hollow pipes through which hot air could reach the upper floors of the building (Figure 1.1).

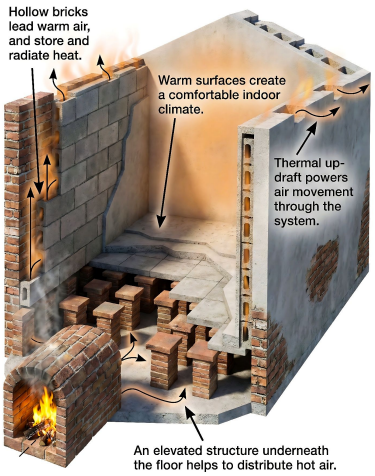


Figure 1.1: Cross-section of a hypocaust. Source: Dfaguimba [21].

Despite being the forerunner of modern central heating, the hypocaust did not solve the issue of worsening air quality. Lehar provides direct evidence that malfunctioning or damaged hypocausts could allow soot and flue gas to escape into occupied spaces [40], which would cause its occupants to breathe toxic fumes.

It would not be until the late Middle Ages that a key event regarding domestic heating systems and air quality control would take place: **the mass adoption of the chimney**. This presented a significant improvement in the management of fumes indoors. The rationale for that statement is that before this time most homes had open hearths, with a large fire used both for heating and cooking. This layout caused the smoke generated to billow around the inside of the building for a long time, until it gradually escaped through gaps or louvers in the roof. Chimneys solved this issue by providing smoke a direct pathway to the outside of a home [49].

Chimneys were not adopted overnight, but rather over a long period spanning several hun-

dred years. During that period, their heat transfer capabilities would increase steadily due to continuously improving manufacturing processes. Moreover, the drive to achieve better thermal efficiency led to the creation of new innovative designs such as the Franklin stove (Figure 1.2).

Despite all this innovation, chimneys would not be the main characters in the next relevant milestone in the history of indoor air quality control. That role would be played by academics, doctors and nurses.

Figure 1.2: Franklin stove, an advanced fireplace with improved heat transfer capabilities invented by Benjamin Franklin in 1742. Source: The Amazing Adventures of Ben Franklin [11].

Throughout the 19th century, some members of the healthcare community, such as **Max von Pettenkofer**, **John S. Billings**, or **Florence Nightingale** (widely considered the founder of modern nursing) [79] among others, highlighted the importance of **ventilation** for indoor air quality. Pettenkofer approached the need to introduce ventilation air in rooms with scientific rigor, since he considered this procedure a key item in his research on hygiene [41]. Billings, on the other hand, derived from his extensive studies the volume of fresh air needed to ventilate different types of buildings [13], and his work was instrumental in prompting many US states to pass minimum ventilation laws [35]. Through her own extensive nursing practice, Ms. Nightingale advocated thoroughly for hospitals to be well ventilated, as she believed that pure air was essential to help patients heal [48].

Research on indoor air ventilation similar to that conducted by these individuals was particularly enlightening, especially considering that the population's attitude was not keen on ventilation. At the time, indoor temperature could only be regulated through chimneys, which provided heat, and windows, whose opening during winter would cause much of the accumulated heat to escape, thereby cooling the entire dwelling.

Fortunately, these problems would soon fade away. At the turn of the 20th century, powerful mechanical ventilation systems started to appear as a consequence of the electrification of industries. One of these devices stands out for the profound impact it has had on people's lives thereafter: **air conditioning** or AC. This invention marked a turning point for indoor thermal control, as for the first time in human history indoor environmental conditions, such as temperature or humidity, could be controlled with a degree of precision never seen before.

Figure 1.3: A 1930s air conditioning unit manufactured by the De La Vergne Machine Company. Source: Tony Mormino [46].

Throughout the early decades of the century, many iterations over the original designs substantially increased the functionalities of the initial prototypes. Companies like the Carrier Air Conditioner Company (founded by the father of AC, Willis Carrier) constantly developed and manufactured newer, smaller and more capable air conditioning units to satisfy the rising demand for these products. An example of one of these improved air conditioning systems is the *De La Vergne Self-Contained Air-Conditioning Unit* (Figure 1.3), created by the De La Vergne Machine Company, which be-

came one of the first commercially available air conditioners
targeted at individual consumers [46].

Overall, air conditioning systems experienced a remarkable surge in both popularity and technological development. In little more than half a century, what had once been large and expensive machines designed primarily to provide cooling for large theaters, office buildings or ships [69] evolved into compact, affordable devices being installed in most new homes in the United States by the late 1960s [25].

Just as air conditioning began its phase of technological maturity around the 1930s, another key component in indoor environmental management was slowly becoming more widespread. **Gas-fueled central heating systems** had already entered the market around 20 years earlier. However, it was not until the wider development of gas pipeline networks that happened in the 1940s and 1950s that such systems became a common choice for new home construction across Europe and North America. This adoption occurred, not only due to the extended support infrastructure for the aforementioned pipeline networks, but also because of the development of smaller, more efficient gas furnaces and boilers that offered a simpler and cost-effective alternative to coal furnaces [71].

Figure 1.4: Modern domestic gas boiler. Source: Leroy Merlin [42].

For the next couple of decades, gas heating systems continued to be used extensively, as they remained the cheapest heating option for homes with access to gas pipelines. The oil embargo of 1973, despite causing oil prices to quadruple [32] and being the most serious recession to date since 1937 [80], did not significantly affect the gas heating industry. In contrast, oil shortages in the immediate aftermath of the crisis forced manufacturers to develop more efficient designs in order to maintain a competitive edge over other heating technologies [71].

This continuous effort to develop increasingly affordable and efficient gas furnaces enabled these systems to remain popular throughout much of the world to the present day. Even despite the steady rise in the demand for more sustainable alternatives, such as electrical or biofuel systems, gas-powered systems like the one in Figure 1.4 remain the most widely used domestic heating method in many places around the planet, including the European Union [27] and the United States [1].

1.2 Motivation

1.2.1 The silent killer

Every year, an average of 125 deaths due to carbon monoxide (CO) poisoning occur in Spain alone [57]. Sadly, this country is just one more in the long list of nations affected by this deadly issue. CO poisoning has been indicated to be the leading cause of death from accidental poisoning in Europe [14]. France (61 deaths per year between 1985-1998 and 2001-2002), the Czech Republic (269 deaths per year from 1986 to 2008) or Germany (1541 deaths per year from 1980 to 2007) are also plagued with cases of CO-related deaths [14]. North America is not in a better position either. Annual non-fire related deaths due to CO poisoning averaged 422 in the United States between 2005 and 2018 [68] and 62 in Canada between 1981 and 2009 [39].

Most of these accidents occur as a consequence of operating faulty heating or cooking sys-

tems, improperly vented gas appliances, blocked chimneys or improperly installed wood burners [14] [39] [18]. Therefore, it is no surprise that most of these intoxications take place in indoor home environments, since that is where people interact with those systems most often [14] [39]. Faulty gas appliances like the heating systems mentioned in the previous section are some of the leading causes of carbon monoxide poisoning [14].

Furthermore, the fact that carbon monoxide is the poisonous agent also explains the high death toll. This substance is a colorless and odorless gas —hence its nickname, 'the silent killer'— and it is therefore undetectable by the human senses. The only way people can detect its presence without specific sensors is through the symptoms it causes: shortness of breath, headache, irritability, fatigue or dizziness are among the mildest ones. However, these symptoms are not specific to CO intoxication, which further hinders a person's ability to detect a rise in the levels of carbon monoxide. Moreover, if a person remains exposed to the gas after the phase of moderate intoxication, acute adverse effects such as loss of consciousness, seizures, coma and, ultimately, death can occur [14] [39].

Figure 1.5 illustrates the effects and symptoms of exposure to carbon monoxide, categorized by level and duration of exposure.

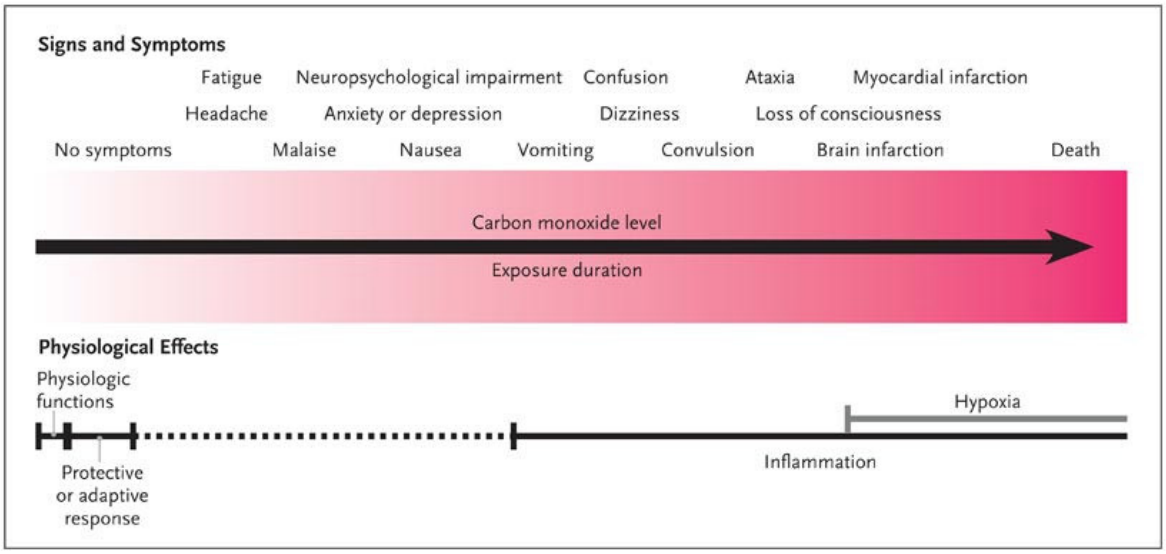


Figure 1.5: Effects of exposure to carbon monoxide, according to the level and duration of exposure. Source: New England Journal of Medicine [78].

Nevertheless, as numerous as these accidents are, the combined thousands of annual accidental deaths due to CO poisoning could be avoided almost completely with a relatively simple solution: **a carbon monoxide sensor attached to some sort of alarm**. This simple setup would ensure occupants of the room where it was installed (ideally, somewhere with potentially hazardous elements such as gas stoves or wood burners) would be alerted whenever the concentration of CO exceeded a given threshold and could then act accordingly.

1.2.2 Temperature and humidity sensors

As previously described, a surge in carbon monoxide concentration can be fatal. However, that is not the only environmental parameter whose continuous rise could also be catastrophic. A sustained increase in the temperature of an indoor space could be a potential indicator of a developing fire. Similarly, a rapid increase in humidity could indicate flooding inside that room.

In both cases, a similar setup to the one described for the carbon monoxide scenario would suffice; but there is an even better approach. These two micro systems are not mutually exclusive, so nothing forbids them from functioning together. Therefore, the aforementioned CO sensor-alarm bundle could be combined with a **temperature sensor** and a **humidity sensor** to form a compact yet capable **station of sensors**. The working of this system is conceptually simple: if any of the parameters read by the sensor kept increasing for too long or passed a certain threshold, the alarm would emit a sound, thereby warning anybody in the vicinity.

1.2.3 Other gas sensors

Until now, the "station of sensors" or simply "station" can measure several environmental parameters, including the concentration of carbon monoxide. Indeed, carbon monoxide is a hazardous by-product of the use of faulty domestic gas appliances. Nevertheless, there are other relevant gases emitted throughout the transportation and use of the fuel used by such appliances.

- **Methane** (CH_4) is the most abundant component within Liquified Natural Gas (LNG), which is the main fuel used in domestic gas appliances supplied via pipeline networks. Methane is an odorless and colorless gas that is extremely flammable and is also an asphyxiant [43].
- **Propane** (C_3H_8), a by-product of LNG processing, is one of the main fuels used to power common gas appliances. Like methane, it is flammable and asphyxiant [54].
- **Hydrogen gas** (H_2), despite not being used as fuel for gas appliances and not normally generated as a by-product of their usage, is still asphyxiant and highly flammable [33].

Due to their hazardous nature, it would be appropriate for the sensor station to also monitor these substances. The fact that they are all flammable and relatively common in everyday life is particularly relevant, as any spark or electrical malfunction in a room with a high concentration of any of these components can potentially cause an explosion.

In addition, since the sensors that measure the concentrations of these gases are very similar to the carbon monoxide sensor —they all belong to the MQ series, as described in section 3.3.2 "The problem with gas sensors"—, including them in the station would not pose significant technical challenges.

1.2.4 Additional components

Aside from the list of sensors mentioned above, it would be ideal to include some additional components into each station, beginning with **some sort of display** that shows the values read by each sensor. Moreover, it would also be desirable for the stations to have a means to communicate with one another, preferably through a wireless medium. To do this, **a small radio module** could be installed in each station so that it can receive updates and send its telemetry to nearby stations.

Furthermore, it would be optimal to add a **sensor that could measure Volatile Organic Compounds (VOC)**, as that parameter is instrumental in assessing the overall air quality of an indoor space.

Other sensors such as **a light intensity sensor** and **an air pressure sensor** could also be added to the stations. Although not directly related to any of the hazardous conditions against

which the rest of the sensors act as safeguards, they could still provide useful information and increase the coverage of environmental data.

Lastly, **a controller device** would be needed in order to manage all these elements. This component would synchronize and handle the sensors taking the readings, the display showing them and the radio transmitting them and receiving other stations'.

1.2.5 Web application

As useful as these sensor stations are with all their functionalities, they have one major limitation. If no external interface was added to the system, the only way users could monitor environmental parameters and detect a potential hazard would be if they were physically close to a station. Only then would they be able to look at the sensor readings in the display and hear the alarm sound whenever an environmental hazard surged.

For this reason, the system could be expanded with a new interface, a mechanism that users could refer to to check their station readings, even when they were not in the vicinity of any of those devices. This interface would be some sort of application (for example, a mobile app).

1.3 Objectives

The primary objective of this thesis is to develop a commercially feasible system made up of networks of interconnected sensor stations. These stations will measure the environmental parameters described in the previous section at regular intervals and send that telemetry to other stations. Some stations will be connected to a database in which they will insert environmental readings. In addition, some stations will be capable of operating solely on battery life in a completely wireless manner.

Furthermore, an application will also be developed to provide an external interface for users to visualize their stations' readings. This app will connect to the aforementioned database, retrieve the current and past telemetry of a user's stations and plot each environmental variable into its corresponding graph.

Even though the development of such a system involves working with multiple approaches, environments, technologies and building phases, there are four core design principles that will be observed throughout the entire project.

1. **Commercial viability**. This project is the seed for what could one day become a real product. Not just any other electronics gadget, but **a device capable of potentially saving thousands of lives each year**; that is the ultimate goal. However, to achieve this, every detail must be designed with scalability, usability and affordability in mind.
2. **Battery life**. Since some stations will operate using their battery as the sole power supply, it is crucial that emphasis is placed on ensuring these devices use as little energy as possible. Not putting enough effort into this item could directly handicap the commercial viability of the project, since users would not be likely to keep using a product that, despite being advertised as fully wireless and with a considerable battery life, would still require them to charge the product every few days.
3. **Secure communications**. Like any Engineering project that involves communications between two different systems, effort must be put into designing a secure channel that

is resilient to cyberattacks. Therefore, it will be necessary to continuously analyze the communication scheme in order to detect potential vulnerabilities promptly and provide effective and secure solutions to them.

1.4 Guide to the document

The current document offers a comprehensive report on the entire project. Due to the extensive amount of information it contains, it was decided to include in this section a brief overview of each chapter.

The following chapter, **Chapter 2**, includes a review of the state of the art in indoor environmental monitoring. In it, alternative systems to the one proposed here are analyzed as part of a small market research conducted specifically for that purpose.

The core sections of this thesis appear in **Chapters 3 and 4**. The former describes everything related to the stations themselves, including system architecture, components, communication protocols, Internet access protocols and much more. Meanwhile, chapter 4 is dedicated entirely to the system's web application. It includes detailed explanations of its structure, development process, data architecture, test suite, etc.

Chapter 5 offers the results obtained from the development of the system. Their heterogeneous distribution includes quantitative results, such as the calculations that derive the expected battery life of wireless stations, and non-numerical results, such as a contribution to an open source repository done during the development of this project, among others.

Finally, **Chapter 6** presents the conclusions drawn from the project, as well as potential improvements that could be used to further extend the system's functionality.

Chapter 2

State of the art

The Internet of Things (IoT) has clearly revolutionized the electronics consumer market since its advent a few years ago. In fact, the word "revolution" might be an understatement, especially considering that entire new industries based exclusively on this field have arisen since it was born. One of such expanding industries is indoor air monitoring, which is the one that this project belongs to. As a result, this area is particularly relevant to the current discussion and will therefore be analyzed in the following paragraphs.

2.1 Indoor air quality monitoring devices: a case study

A quick search for "indoor air monitor" in a random browser automatically returns a myriad of different devices. As shown in Figure 2.1, there are countless different models of air quality monitors, each with their own shape, size, interface and, most importantly, functionalities.

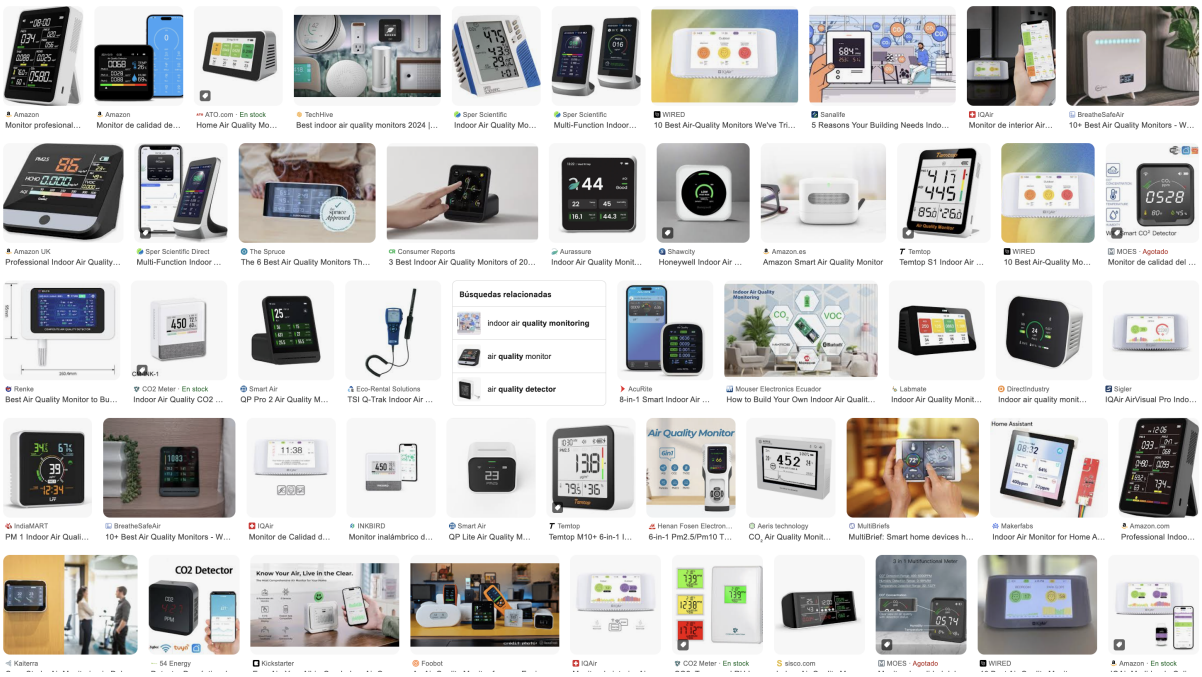


Figure 2.1: Result of searching "indoor air monitor" using Google Chrome.[1]

[1] All figures that do not explicitly mention a source are owned by the author of this document.

In order to provide a clear picture of the current state of this industry, a thorough analysis of the devices returned by this search had to be carried out. With this purpose, more than 25 different products were analyzed and their characteristics were registered in the detailed spreadsheet that can be found in Appendix A "Comparison of Indoor Air Monitoring Alternatives".

The resulting dataset consists of 27 registers, each corresponding to a different product. Despite the pool of entries not being large enough to provide conclusive results, it is still comprehensive enough to probe this particular market and depict the general distribution of its products and their capabilities.

In the aforementioned spreadsheet, the columns whose values can only be checked or unchecked boxes represent all the relevant environmental parameters that were considered necessary for this system in the previous chapter. It is important to mention here that most of the products analyzed advertise a wider set of monitored parameters than what was collected in this study (e.g., carbon dioxide, PM2.5, PM10). The purpose of excluding these features from the spreadsheet is not related to introducing a bias into the study or belittle these products, but rather to provide an account of their capabilities that is consistent with key parameters previously identified as essential for the system. From this point onward, any mention of environmental variables or parameters in the context of this study will refer exclusively to those represented in the spreadsheet unless otherwise indicated.

One of the columns in the spreadsheet, named "Amount of relevant parameters monitored", contains the total number of checked boxes in that row. It represents those environmental parameters considered essential for our system that that particular device can monitor.

Despite the accuracy of its data, the aforementioned spreadsheet does not make it easy to quickly determine the relative position of different entries to one another, especially in the two most relevant dimensions of the dataset: the price and the amount of relevant parameters that each device monitors. For that reason, a plot was generated with the information included in the dataset using only those two features (Figure 2.2). Please note that the vertical axis is shown on a logarithmic scale.

A brief inspection of this plot yields several important conclusions.

1. None of the 27 devices that were analyzed monitors more than 5 of the 9 target environmental variables, with the majority of them (16 out of 27) only monitoring 3 of those parameters. This fact proves that **there is a fundamentally unexploited market niche** for devices like the one described in this document.
2. For the amount of relevant parameters that these devices monitor, they are very overpriced. Even taking into account all the parameters their manufacturers advertise as being monitored, **they remain far too costly for the average citizen**.

Another conclusion can be extracted by analyzing the "Battery life" column in the spreadsheet: all except three of the devices are either not designed to operate on battery life alone or can only work continuously for less than two days before they need to be recharged.

These key findings are the foundational basis on which the development of the entire project is based. The analysis described here proves not only that there is a niche market for gas-focused environmental monitoring with essentially no competition, but also that no manufacturer is building products that combine so many capabilities with such a focus on affordability and battery life management as this one.

This study of the current state of the indoor environmental monitoring industry has been

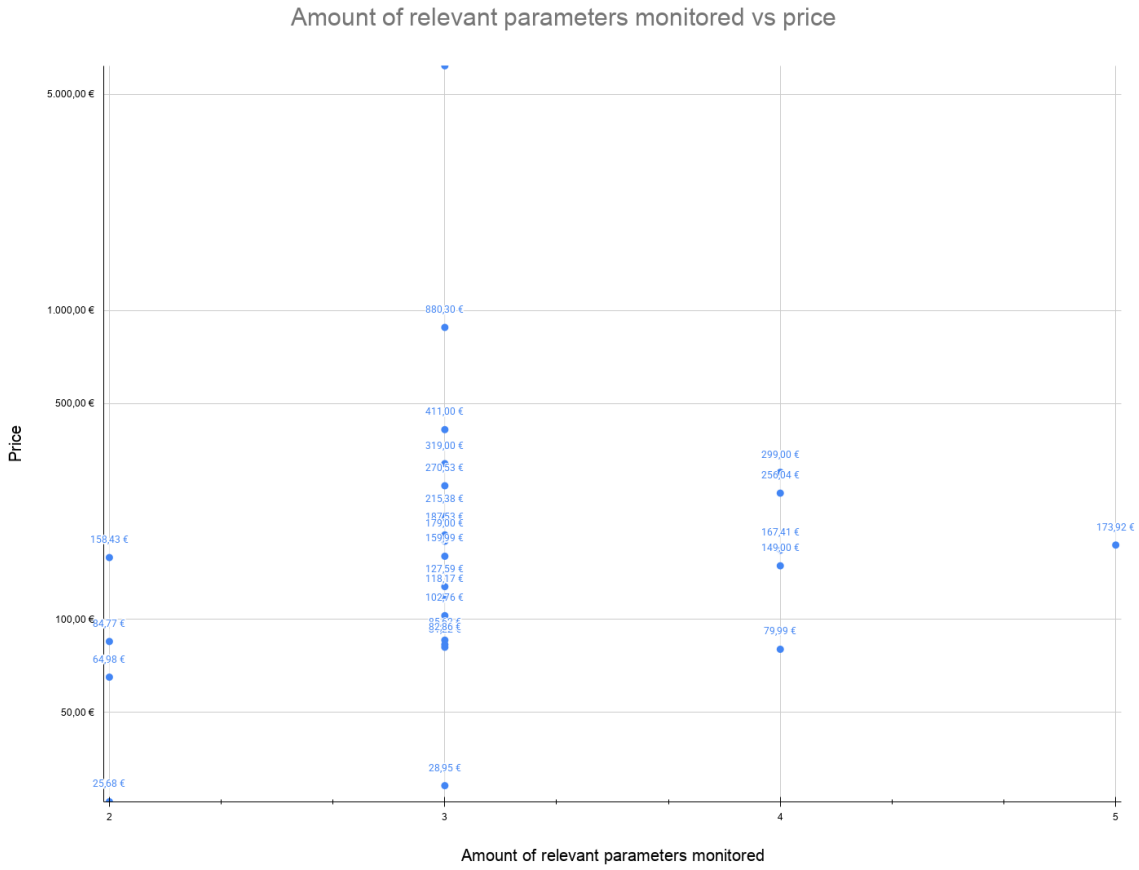


Figure 2.2: Amount of relevant parameters monitored by each product and their respective price.

instrumental in analyzing and comparing existing alternatives to the proposed system. From this research effort, it can be concluded that there is indeed a technological and commercial gap that could be addressed through the development of this system, while still adhering to project objectives set in section 1.3 “Objectives”.

Chapter 3

Stations

3.1 Overview

Stations are the cornerstones of this entire system. They are in charge of reading environmental data through the sensors they contain, processing it, establishing and managing connections to other stations, sending their readings to associated stations via radio and uploading all that generated data to the system database.

Due to its importance in the overall system, the planning, design, coding and evaluation of the different features have taken most of the time allocated for this project. Furthermore, throughout its development, a myriad of new features with which to improve the stations' capabilities have come up. A few could be implemented on time and are now some of the most visible features of the stations, while others had to be left on the back burner because of the limited time frame within which this project had to be completed.

The next sections of this chapter contain a detailed summary of every aspect of the stations and its development process, from system requirements to the mode of operation and the role of each of their main components.

The firmware for the stations and its associated files is publicly available in the following GitHub repository: `https://github.com/RoiCorporation/tfg-firmware` [62]. The repository and all associated materials may be modified at the discretion of the author.

3.2 Requirements

Building a project of this magnitude would be impossible without establishing a basic set of requirements early in the planning process. These requirements set the baseline for how the final system is expected to operate and guide the entire development process by defining the key features to prioritize.

Consequently, the following paragraphs carefully nest the different system requirements inside the type of requirement they belong to, i.e. **functional requirements** and **non-functional requirements**.

Functional requirements

- At least some of the stations must be able to operate on their own power, without being connected to any power outlet.
- There must be some method or protocol for wireless communication between stations.
- At least some of the stations need to provide reliable measurements of the concentrations of different gasses in the air.
- At least some of the stations must be considerably light and small; ideally each must be easily picked with one hand.

Non-functional requirements

- Each station must take environmental measurements once every minute.
- Wireless stations must be able to operate under normal workloads for a continuous period of not less than three (3) months on one battery charge.
- Stations' Microcontroller Unit (MCU) must support the Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I^2C) protocols.

3.3 Design

3.3.1 Initial approach to the system architecture

Throughout the project's initial planning and early development stages, the best architecture for this particular system was believed to be a star-like topology made up of wireless stations at the edges and a central station as the central hub. In this arrangement, each wireless station would send its readings directly to the central station it was associated with, and the latter would, in turn, handle incoming radio messages with the telemetry of all the wireless stations it was linked to and upload that data to the system database.

Furthermore, due to the increased workload of the central stations, these devices would be powered from a mains outlet to reduce the load on the onboard battery and allow unrestricted operation independent of battery life.

Initially, the aforementioned division of tasks between the two types of stations was coincidental, i.e., all stations had the same components and could read the same environmental parameters. Since their hardware was identical to each other, the firmware they were booted with depended only on whether they were going to be used as central stations or wireless stations. Initially, the aforementioned division of tasks between the two types of stations was coincidental, i.e., all stations had the same components and could read the same environmental parameters. Since their hardware was identical to each other, the firmware they were booted with depended only on whether they were going to be used as central stations or wireless stations.

This approach, despite its simplicity, had one important caveat: it failed to consider how a station's onboard power supply (the only power source that wireless stations could use during normal operation) would affect its performance.

3.3.2 The problem with gas sensors

One of the key performance parameters that stations needed to achieve was an outstanding battery life —at least three months—. Not only was this item one of the initial requirements, thus having top priority from the beginning of the project, but it was also a metric of utmost importance if the system was to become commercially feasible (since IoT devices such as these are expected to operate remotely and purely on battery power alone). The combination of this requirement with the request that there be wireless stations in the system soon became the biggest challenge that was faced in the design stage.

The aforementioned challenge presented itself as soon as testing of the gas concentration sensors began (see section 3.4.2 "Sensors and components" for further details). All of these devices belong to the MQ-series, a family of electrochemical sensors whose analog output voltage varies when exposed to their target gasses. For them to work properly, the metal oxide semiconductor they contain (which is the component whose conductivity actually changes with the target gas concentration) must be preheated to a significant temperature.

Fortunately, these sensors already come with an onboard heating resistance that uses the supplied power to increase their internal temperature. Unfortunately, they draw a substantial amount of power while doing so. After cross-checking their datasheets against custom tests performed using a multimeter, it was found that the power drainage during this process stood at the high hundreds of milliwatts. These figures are completely unacceptable with the stations' current power setup, as it would be impossible to power the station for more than a few hours with only one 18650 3500 mAh battery.

When thinking about a solution to this problem, one could perhaps believe that, since stations only take one reading per minute, gas sensors could be powered off most of the time and only turned on momentarily to make their respective readings. This strategy, although ideal for the current use case, is hopelessly flawed due to a limiting detail about the preheating process of these gas sensors. According to their datasheets, the recommended preheat time necessary to bring up the internal temperature of each sensor to a proper working level is between 60 seconds and 48 hours, depending on the sensor. Therefore, the idea of constantly switching these components on and off is simply not feasible.

Another valid line of reasoning would be to consider a different set of gas sensors with which to equip stations, different from the MQ-series ones. However, this approach also turned out to be unviable. In order to replace the MQ-series, the new set of gas sensors would need to be:

a) Similarly priced (below 5 € per sensor).

b) Capable of detecting individual gases, rather than merely responding to a range of gases.

c) Less power-hungry, ideally featuring a low-power mode or at least not requiring preheating.

Despite constant search, all attempts to find a family of sensors that met these three conditions simultaneously failed. As a consequence, the possibility of replacing the MQ sensors to overcome their high power consumption was also discarded.

3.3.3 Extending the battery life of wireless stations

After much consideration, it was decided that the most reasonable solution for this limitation was a modification of the original design. Just as with the previous iteration, the topology would still

be star-like, the communication schema would rely on wireless stations sending their readings to a central station acting as a hub, and the responsibility to upload data to the database would still rely exclusively on the central station.

However, the improved system design had a key difference with its previous version: instead of all stations sharing the exact same hardware, central stations would be the only devices with gas sensors. Wireless stations would be stripped off all gas sensors, thereby avoiding all limitations regarding battery life that operating such sensors entails. Moreover, gas concentration data would still be read and registered in the system, through specific firmware booted exclusively into central stations.

Of course, this approach was not perfect. Its only drawback, albeit a significant one, was that the system would only be able to register gas concentrations in the vicinity of central stations. Unfortunately, this was a necessary evil to ensure that wireless stations had a realistic possibility of meeting all their requirements. The alternative was to reduce these stations' battery life to just a few hours, which would render their wireless capabilities completely useless (as they would need to be either charged several times per day or kept constantly plugged into a power outlet).

3.3.4 Definitive system architecture

As was mentioned before, the system follows a star-like topology with wireless stations at the edges and central stations in the middle. In addition, central stations connect to an MQTT (Message Queuing Telemetry Transport) broker, which uploads data to the database through a custom API. An explanatory diagram of the entire system and the technologies it uses can be seen in Figure 3.1.

The definitive architecture of the system considers two types of stations: wireless stations and central ones. Both take environmental readings and feed them to the system, but they play two very different roles within it.

Wireless stations take minute-by-minute readings of temperature, humidity, light intensity, air pressure and Air Quality Index (AQI). These measurements are then sent to the central station it is associated with, so that the latter can upload them to the system database. Between measurements, these stations are set to low-power mode, thanks to which their battery life exceeds three months between charges under normal operating conditions (see section 5.2 "Battery Life" for a detailed analysis on the power consumption of wireless stations).

Central stations register the same environmental variables with the same frequency as their wireless counterparts, plus atmospheric concentrations of carbon monoxide, methane, propane and hydrogen gas. Each receives by radio all the readings from the wireless stations associated with it and manages the upload of both those stations' readings and its own readings to the system database. Since they must be constantly listening for radio messages sent by other stations, they are not set to low-power mode between readings. Unlike wireless stations, however, battery life is not a limiting factor here because these stations are designed to operate with an external power source (for example, a power outlet).

Figure 3.2 provides a quick reference of the functionality of each type of station by listing the environmental parameters monitored by each one.

There is one more important detail about the design of the system that needs to be addressed. Each station is identified by the ID carved at the top of its casing. This number is a Version 4 Universally Unique IDentifier (UUID), a 16-byte ID in which most bytes are generated randomly or pseudorandomly [20]. Since they uniquely identify elements, these UUIDs are used extensively

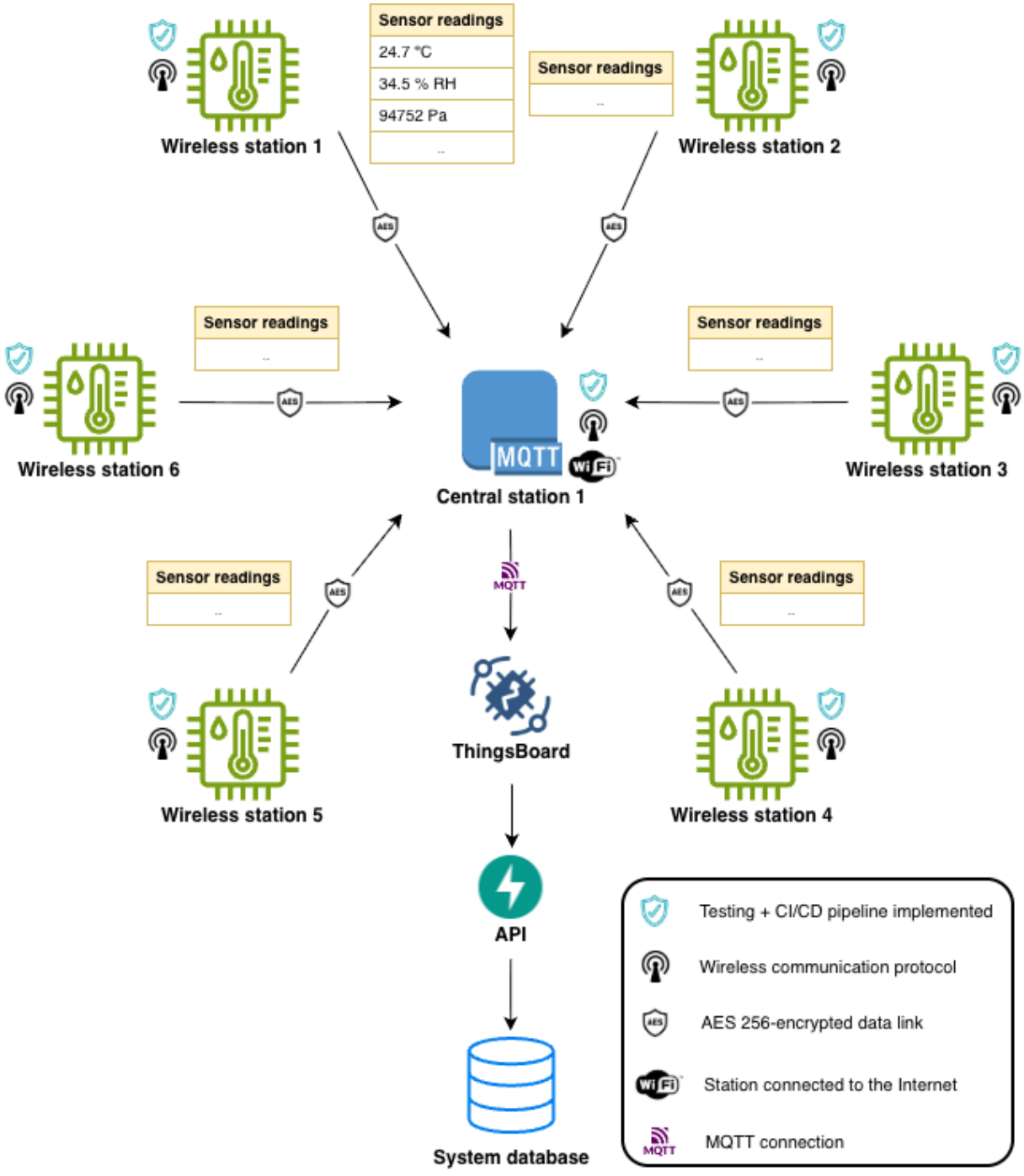


Figure 3.1: Overview of the system architecture.

Environmental parameter
Temperature
Humidity
Light intensity
Air pressure
Air Quality Index (AQI)

Environmental parameter
Temperature
Humidity
Light intensity
Air pressure
Air Quality Index
Carbon monoxide (CO) concentration
Methane (CH_4) concentration
Propane (C_3H_8) concentration
Hydrogen gas (H_2) concentration

Figure 3.2: Environmental data collected by wireless stations (left) and central stations (right).

throughout the system. For example, when a user wants to add a particular station to their account in the web application, he must use that station's ID (section 4.3.4 "Data architecture" explains this process in detail).

3.4 Implementation

3.4.1 Development framework

Choosing the right framework to build the stations was one of the key decisions of the planning stage. Failing to select an appropriate platform for the firmware would likely result in unmet system requirements. Because of this, the following paragraphs are dedicated to explaining the reasons for selecting the framework and MCU.

The Raspberry Pi platform and the Pico 2 W board

Raspberry Pi [58] is, along with Arduino [9] and Espressif [26], one of the leading manufacturers of development boards. When considering all the alternatives that these platforms offered, some characteristics were particularly important for this particular use case:

- **Price**. For the project to be scalable and commercially feasible, choosing a board that was too expensive would not be a good idea from a financial point of view.
- **Capabilities**. The board would have to support several hardware protocols (SPI, I^2C), have enough General Purpose Input/Output (GPIO) pins to connect all the peripherals, support WiFi connectivity and have an adequate overall performance.

After searching the catalog of each of the manufacturers listed above and filtering out those items that did not meet the aforementioned requirements, the list of candidate boards was narrowed down to only three options, namely Raspberry Pi's Pico 2 W and Espressif's ESP32-S2. These two boards had very similar features and their price was also pretty much the same, around the 6-9 € range [72] [70]. Arduino's Nano 33 IoT, Nano RP2040 and Nano ESP32, all boards with similar features to the former two, were two to three times more expensive than them, costing 20-26 € each [6] [7] [8].

Ultimately, the choice between the Pico 2 W and the ESP32-S2 came down mostly to a matter of personal preference, as they both had the same price, same wireless capabilities,

enough GPIO pins and both supported basic hardware communication protocols. **The Pico 2 W (Figure 3.3) ended up being the option selected**, mainly because, if it ever became necessary during the project to set up a local server of any sort or run a service locally for an extended period, the default hardware choice would likely have been a Raspberry Pi board. In such a scenario, basing the codebase on the Raspberry Pi ecosystem would likely have resulted in fewer compatibility issues than relying on the Espressif environment.

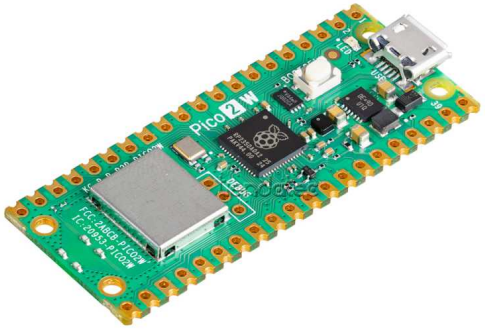


Figure 3.3: The Raspberry Pi Pico 2 W board. Source: tiendatec [72].

The choice of the programming language

Once the Pico 2 W was chosen as the MCU for the system's stations, there was only one last general design decision left with regard to firmware: which programming language to use. Since Raspberry Pi boards can be programmed using both MicroPython and C, this decision was not trivial.

On the one hand, since MicroPython is essentially just an efficient implementation of Python 3, the code generated using this language is necessarily shorter and requires less boilerplate than its C counterpart. Furthermore, package management and library installation is much simpler with the former option, since it does not require configuring any file for the CMake build system and it is based on a derivative of Python's popular and extremely easy-to-use "pip" package manager.

However, it is worth noting that bare C programs, which the Pico 2 W can run thanks to Raspberry Pi's Pico SDK, run significantly faster than those written using MicroPython (sometimes up to several orders of magnitude faster) [30] [31]. Besides, the ability to handle memory allocation at a lower level, pass arguments by reference instead of rellying on MicroPython's *pass by assignment* model or explicitly define the bit-width of each integer variable are all features that made C a worthy candidate language.

After considering both alternatives, the decision was made to use C. This turned out to be a good decision, since many of its low-level features were needed on multiple occasions. Moreover, developing the firmware entirely in C gave provided an incredible opportunity to learn more about this programming language as well as how to work with external C libraries and manage the structure of production-grade embedded systems' firmware codebases.

3.4.2 Sensors and components

For the stations to be useful, the Pico boards must rely on multiple different components that perform a number of specialized functions. From the sensors that take readings of the target environmental variables to the buzzer that plays a distinctive alarm when a hazard is detected, every device is necessary to ensure that the station works properly and uses its full capabilities.

Before discussing all the components and how they are used in the system, a brief note on their layout is in order. In this document, all components are referred to by their name (e.g., BH1750, MQ-5, NRF24L01). However, with the exception of the buzzer, they are not used in isolation but integrated into the breakout boards on which they are shipped. Thus, whenever a component is mentioned, it refers to the entire breakout board assembly, including the component itself as well as any supporting resistors, capacitors, pins, and other auxiliary elements.

Sensors

Sensors are the centerpiece of a station, whether central or wireless. They are responsible for taking readings of their target environmental parameters and therefore have a key role in the operation of each station.

There are two sensors common to every station (central or otherwise), namely the BME680 and the BH1750.

- The **BME680** sensor is a digital sensor that can measure temperature, humidity, air pressure and VOCs (the latter ones are used to provide a measure of the Air Quality Index). Manufactured by Bosch, this compact device communicates with the controller via the I^2C protocol and it is a fundamental piece in the whole system, as it alone provides measurements of almost all basic environmental parameters.
- The **BH1750** is a digital ambient light sensor that stations use to take measurements of light intensity. Just like the BME680, this device also uses I^2C to communicate with an MCU.

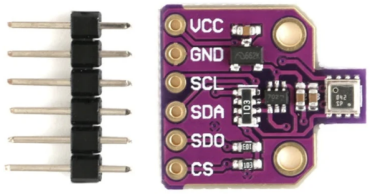
As mentioned above, both types of stations read at least these environmental parameters (that is, temperature, humidity, air pressure, Air Quality Index (AQI) and light intensity). Nevertheless, central stations also monitor the concentrations of four different gases in the air. To this end, an additional four sensors from the MQ-series are used: **MQ-7** (carbon monoxide), **MQ-4** (methane), **MQ-6** (propane) and **MQ-8** (hydrogen gas).

Figure 3.4 shows some of the sensors described above. In addition, Figure 3.5 maps each environmental variable to the sensor that monitors it, along with the corresponding measurement unit and detection range.

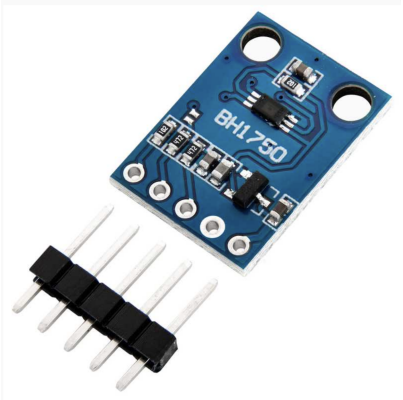
Operational components

In addition to sensors, all stations have several operational components that help them increase their capabilities.

The first of such components is a **passive buzzer**. Whenever the Pico detects that one of the monitored environmental variables has kept increasing for a few minutes or, alternatively,



(a) BME680. Source: Satkit [63].



(b) BH1750. Source: tiendatec [73].

(c) MQ-7. Source: tiendatec [74].

Figure 3.4: Examples of some of the key sensors used in the system.

has surpassed a certain threshold, it activates the buzzer and plays a distinctive sequence of tones. It does this by using different frequencies and varying the PWM signal fed to the buzzer. Thanks to this feature, anyone in the vicinity of the station can be alerted of the potential hazard and identify which environmental parameter is causing it by listening to the melody played.

Another component that is also of great use is an **128x64 pixels OLED display**. This device plays a distinguished role in the operation of each station, since it provides the only physical interface through which the user can actually see the last environmental readings taken by the station sensors. The display shows each of the environmental readings for a few seconds, iterates twice through the whole set of measurements, and then turns off automatically. There are three reasons that justify this last step:

1. Maximizing battery life in the case of wireless stations.
2. Preventing the display from lighting up when no one is present or directly looking at it.
3. Ensuring stations do not increase brightness on intentionally dark environments (for example, rooms at night where occupants may be sleeping).

Since central stations take readings for more environmental variables than wireless ones, their display sequence lasts longer (because their displays also have to add the gas concentration readings to the list of measurements to show).

The third main operational component is a simple **capacitive touch button**. This allows the user to interact with the station via either a short or long press.

Environmental parameter	Sensor	Detection range	Unit
Temperature	BME680	-40 – 85	ºC
Humidity	BME680	0 – 100	RH (%)
Air pressure	BME680	30000 – 110000	Pa
Air Quality Index (AQI)	BME680	0 – 500	AQI
Light intensity	BH1750	1 – 65535	lx
Carbon monoxide (CO) concentration	MQ-7	20 – 2000	ppm
Methane (CH_4) concentration	MQ-4	300 – 10000	ppm
Propane (C_3H_8) concentration	MQ-6	300 – 10000	ppm
Hydrogen gas (H_2) concentration	MQ-8	100 – 1000	ppm

Figure 3.5: All target environmental parameters along with the sensors that monitor them.

- A **short press** is used to wake up the station from low power mode, initialize all components and sensors, take readings from the latter, send those readings via radio and begin the display measurement-showing sequence. If the station was already out of "hibernation", the first two steps are omitted.
- A **long press**, aside from behaving very similarly to a short press with regards to the board's low power status, also starts the handshake protocol that allows a wireless station to be associated with a central one and vice versa. The details of this procedure are explained in section 3.4.3 "Communication between stations".

The last member of this set of devices is the **NRF24L01 radio module**. This SPI-controlled 2.4 GHz transceiver is responsible for sending and receiving radio messages between stations. More precisely, wireless stations send their readings through their NRF24L01 and the central station they are associated with receives them through its own NRF24L01. This method of communication is preferred over connecting wireless stations to the system via Wi-Fi, because the latter would require those stations to remain active for longer periods of time (it takes central stations between 5 and 10 seconds on average to connect to the MQTT broker). That extra time would result in substantially higher current consumption, thereby significantly reducing battery life.

Due to its important role as the backbone for cross-station communication, the technical difficulties that arose when working with it and the complexity of the custom handshake protocol that uses it to link two stations together, section 3.4.3 "Communication between stations" is dedicated to describing the operation of the NRF24L01 in great detail.

Power components

From the beginning of this project, it was clear that all stations would require robust power sources. Given that wireless stations were to rely on battery life alone as their only power source, choosing an appropriate battery cell became crucial. Not only that, but also the station size requirement had to be taken into account too, which in itself already ruled out the possibility of including a large battery pack in each station.

After factoring in these considerations, it was decided that the best solution for this use case would be to use a single **18650 3500 mAh 3.7 V battery** per station. Its compact size and high capacity made it ideal for IoT devices such as the system's wireless stations, and its voltage rating is perfect for powering their Pico 2 W directly.

Furthermore, it was decided to use a battery charger module, in order to:

a) Provide a means to supply external power to central stations.

b) Recharge the batteries of both station types without having to disassemble them altogether.

Including this device turned out to be a very good idea, especially considering how easy and straightforward it is to charge a station's battery once it is fully assembled inside its casing.

This setup, despite being appropriate for a wireless station, is not applicable to a central one (even if it is connected to an external power source). The reason is simple: MQ-series gas sensors need to be powered with 5 volts, and the aforementioned battery model can only provide around 4.2 volts when fully charged. Therefore, a **DC-DC step-up converter** was connected to the output pins of the battery charger module to provide the necessary 5 V input voltage to the gas sensors (Figure 3.6). As a result, in central stations, the power for the Pico 2 W and all its basic components still comes from the battery, while the gas sensors are powered by a different 5 V power line, bypassing the MCU altogether.

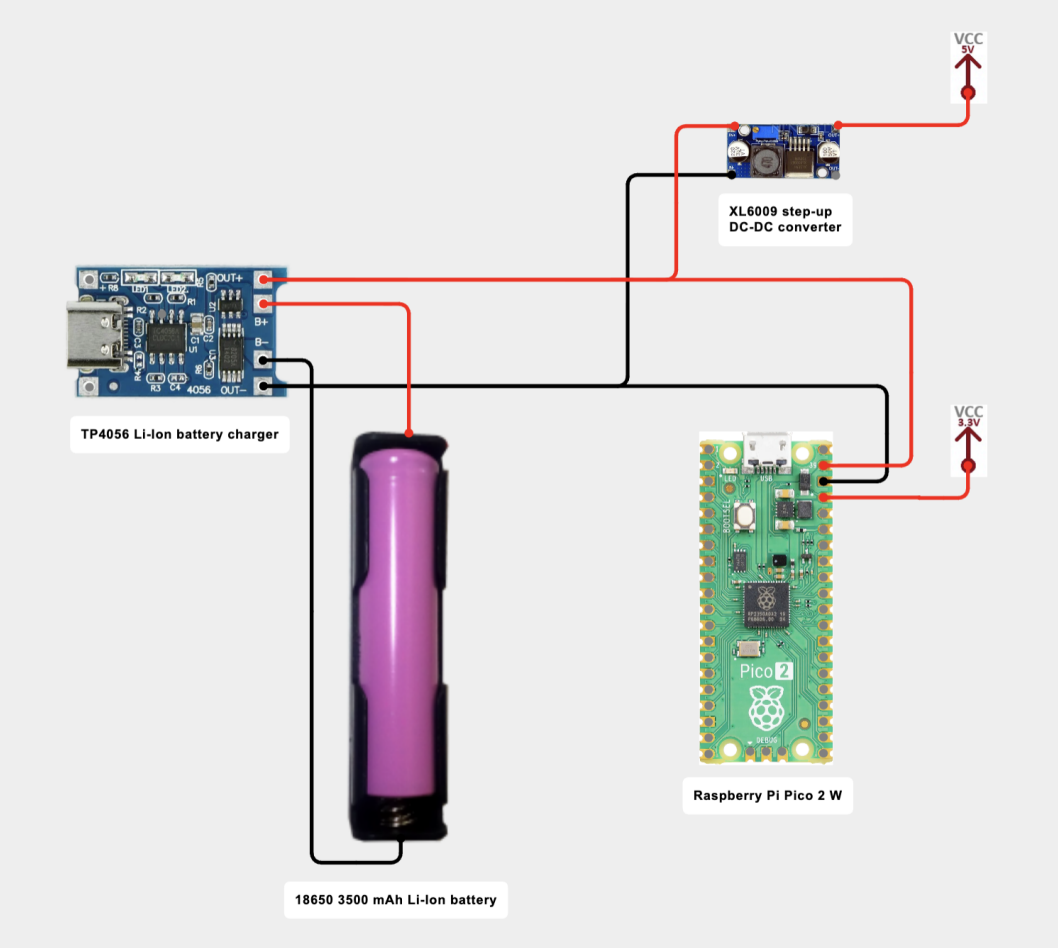


Figure 3.6: Schematic of the central stations' power supply system.

3.4.3 Communication between stations

One of the system's initial requirements was that there needed to be some sort of wireless communication method for the stations. Due to the complexity and amount of time allocated to implement this feature, it is arguably the most important part of the entire system. The following subsections offer a thorough description of the elements involved in such communication system,

as well as the different approaches, limitations and solutions that were implemented to satisfy this key requirement.

The NRF24L01 module

The **NRF24L01 transceiver** works at 2.4 GHz, can transmit at a rate of up to 2 Mbps and, more importantly in terms of of battery life, has an extremely low power consumption when on power mode (below the 1 µA mark). In addition, its supply range of 1.9-3.6 V means that it can be powered directly with the Pico's 3.3 V pin (3V3).

There are two characteristics of these modules that need to be explained to better understand the following sections. The first is the concept of *data pipes* in the NRF24 and how the system utilizes them.

When in receiving or *RX* mode, the NRF24L01 module uses a feature called Multiceiver, which contains a set of 6 data pipes. A data pipe is a logical channel in the physical RF channel, and each pipe has a unique address associated with it. The size of this address can be configured and in the NRF24 modules used in this project, it is set to its maximum value of 5 bytes. Since there are 6 data pipes, a station in RX mode can receive packets from 6 different stations simultaneously, which is why the system allows star networks of dimensions up to 6:1 (six wireless stations connected to a central one).

The second key feature of NRF24 transceivers is their most distinctive limitation: packets sent and/or received through them can only be at most 32 bytes long (see Figure 3.7 for an overview of the entire packet format). The reason for this is that the 3 RX and TX FIFOs where inbound and outbound packet payloads are stored, respectively, are exactly 32 bytes long.

Field	Preamble	NRF24L01 Destination Address	Packet Control Field	Payload	CRC
N. Bytes	1	5	9	0-32	2

Figure 3.7: Format of the Enhanced ShockBurst™ packet used by the NRF24L01.

This constraint turned out to be particularly limiting, especially considering that the ID of each station is 16 bytes long and that the combined environmental data that each wireless station needs to send to its associated central station in every message is 20 bytes. Therefore, it is not possible to fit both the station ID and all its sensors' telemetry in the payload of a single NRF24 message.

This made it clear that it would be necessary to develop a protocol that would somehow allow a central station to identify which station sent a given packet without the latter containing any ID or means of knowing where it had originated whatsoever.

After an analysis of the entire system architecture, it was decided that the best approach to implement this feature would be a handshake protocol. This routine would take place at the start of the communication between a wireless station and its soon-to-be associated central station, and during the process, both devices would exchange the necessary information to establish an association between them (mainly, station IDs and NRF24 addresses).

Initial version of the handshake

Initially, the procedure to associate a wireless connection with a central one was implemented as a two-way handshake, resembling a simplified version of the three-way handshake used in TCP (see Figure 3.8). This method was rather simple, as it only involved transmitting two pieces of data: the wireless station ID and the NRF24 destination address assigned to that station.

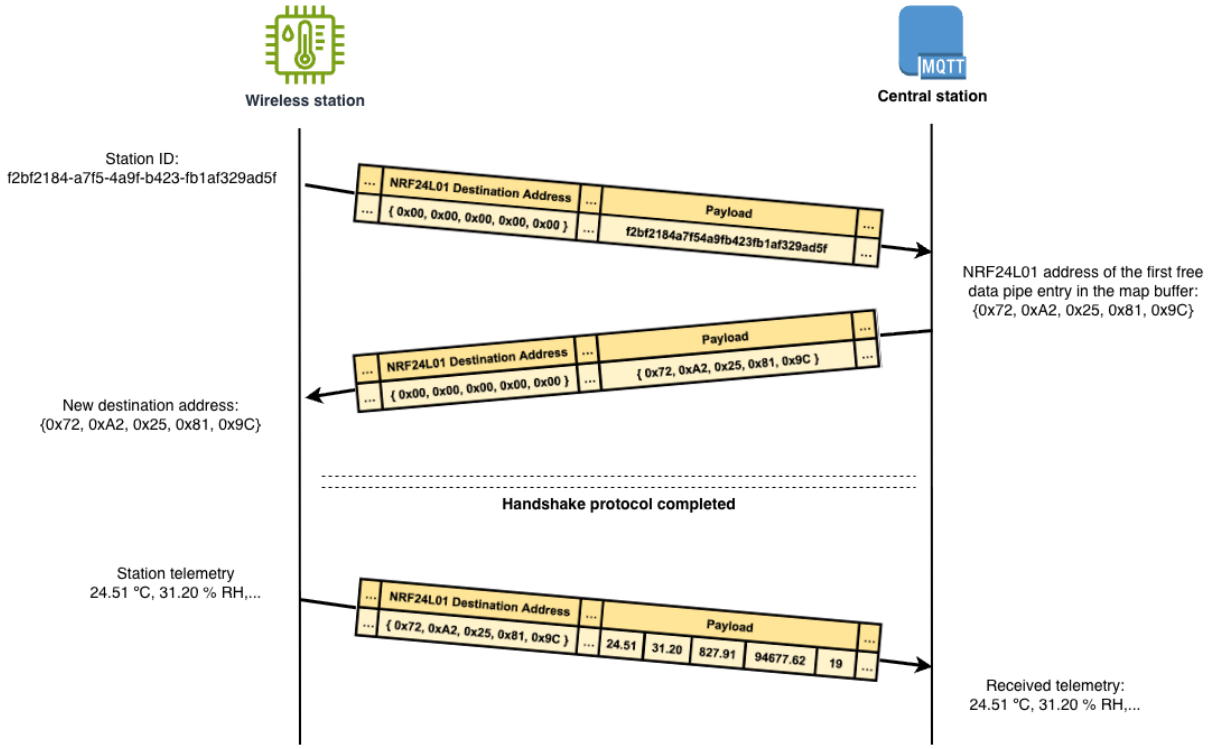


Figure 3.8: First version of the handshake protocol.

The protocol works as follows:

1. The wireless station sends an NRF24 packet with the NRF24 destination address `0x00 0x00 0x00 0x00 0x00`. In this system, that sequence represents the **broadcast address**, and just like Ethernet's `FF:FF:FF:FF:FF:FF` broadcast address, packets sent to that address are expected to be received by every nearby station. In the payload field of that message, the wireless station includes its own ID.

2. The target central station receives the packet and extracts the ID from the message's payload field. Every central station has a buffer where they map the IDs of the wireless stations that are associated with them to the NRF24 address where they expect to receive their packets. Therefore, once the central station gets the ID of the soon-to-be associated wireless station, it stores it into the first free entry of that buffer and selects the randomly pre-generated NRF24 address in that entry. Then, it includes that address in the payload of a new packet, which is addressed, once again, to the system broadcast address.

3. When the wireless station receives this packet, the handshake protocol is finished. From now on, the NRF24 packets it sends with its sensor readings will be directed to the destination addressed that it received from the aforementioned packet.

This procedure, albeit simple, proved to be quite reliable and worked reasonably well most of the time. In all its simplicity, however, it had one important flaw: all the information was

exchanged in plain text. As explained in the following paragraphs, a careful analysis of the consequences of this fact revealed unexpected security vulnerabilities. These issues would end up ruling out this version of the handshake protocol in favor of a more secure one, explained under the Advanced handshake protocol subsection below.

Security vulnerabilities

When examining potential security-related scenarios that involved the handshake described above, two security vulnerabilities were detected that had not been properly identified and needed to be addressed. Both could be exploited by a malicious third party located near a set of stations, as there are no obstacles to eavesdropping on the 2.4 GHz channel used by the NRF24 modules.

The first vulnerability is related to the wireless station ID, which is one of the most important pieces of information sent during the handshake. If the channel was not encrypted, the malicious third party could listen to the exchange of NRF24 packets until it found the one that contained the station ID and then extract it from that packet's payload. If the legitimate owner of the station had not previously registered the wireless station with his account in the web application, the malicious party could link that station to his account instead (see section 4.3.4 "Data architecture" for a detailed explanation of this procedure). From then on, even if the legitimate owner tried to associate it with his own account, the system would detect that the station already belonged to a different user and would not allow him to register it. Not only that, but also the malicious party would then have access to all the readings —past and current— from that station.

The second vulnerability also involves a malicious party eavesdropping on the handshake, but now, when the station ID of the wireless station was transmitted, this third party would immediately proceed to send that wireless station a spoofed destination NRF24 address. If the wireless station received this message before the legitimate one sent by the real central station, it would set its destination address to the one sent by the malicious party. From then on, all the data it sent would only be received by the third party, effectively allowing the latter to hijack all communication between the wireless station and the legitimate central station.

These two security vulnerabilities could not be left unattended, as they could potentially compromise the entire system. Fortunately, a very common and widespread mechanism could be used to eliminate these issues: **encryption**.

The encryption system

This system uses two main encryption schemes to ensure secure communications between stations. In particular, **Elliptic Curve Diffie-Hellman** and the **Advanced Encryption Standard** are the two protocols used during the handshake and/or normal operations of the stations to encrypt, transmit and decrypt data safely and reliably.

Elliptic Curve Diffie-Hellman (ECDH) [36] [38] is a key agreement protocol that allows two parties, each having their own elliptic curve public-private key pair, to establish a shared secret over an insecure channel. This scheme is used during the handshake procedure as it allows the stations involved in it to exchange their public keys in plain text through an open channel, without this posing a security risk.

The **Advanced Encryption Standard (AES)** [19] is a symmetric encryption protocol

used by the stations to encrypt and decrypt the payload of the NRF24 packets they send and receive. More specifically, the system uses the AES-256 CTR version, which is based on a 256-bit key (hence the name) and an incremental counter. The key, an initialization vector and the counter are used together to encrypt/decrypt the payload field of the radio packets sent and received by the stations.

A brief annotation is due regarding the aforementioned initialization vector. All stations have the same 16-byte initialization vector when they start for the first time. However, the last four bytes of that array are replaced by the counter value every time they encrypt and decrypt a packet. This strategy ensures that only legitimate stations that know the initialization vector can successfully encrypt and decrypt each other's packets.

In addition to these protocols, the system uses two other schemes only during the handshake, namely **BLAKE2b** [10] and **ChaCha20** [12]. Inside the Key Derivation Function (KDF) with which the AES key is generated, BLAKE2b hashes the input shared secret created through ECDH, and then ChaCha20 takes that hash as input and uses it to generate a pseudorandom stream of numbers. That array of numbers is the AES key that will be used to encrypt all subsequent communications between the wireless-central station pair that is involved in the handshake. The Monocypher library [45] and its documentation were really useful to implement this KDF.

A more detailed overview of this key generation process is provided in the following subsection, Advanced handshake protocol. Moreover, Figure 3.9 provides a visual example of the same idea for a key exchange and subsequent KDF process as the one that takes place during this handshake.

It is far beyond the scope of this document to offer a detailed explanation of the inner workings of these protocols, as there are many great resources on the Internet that explain them much better than could be done here. Nevertheless, it is still well within the purpose of this paper to explain why they solve the vulnerabilities described in the previous section. To do that, however, it is necessary to thoroughly explain one of the key features of the entire system: **the advanced handshake protocol.**

Advanced handshake protocol

As described in the previous section, the initial handshake had serious flaws that made it unsuitable for its role. As a result, a new protocol had to be devised, one sophisticated enough to overcome the security vulnerabilities that the previous version exposed.

The new approach to this procedure made use of the encryption mechanisms described in the previous subsection. In order to include them in the new variant of the protocol, several additional steps were needed. These consisted mainly of extra radio messages sent between the two participating stations and new inner stages between each transmission. The following paragraphs describe the structure and mode of operation of the different stages of the new handshake. Furthermore, Figure 3.10 provides a detailed diagram of the entire process.

The first section of this new procedure is the **key exchange**, which simply involves each station sending their public ECDH key through broadcasted NRF24 packets. Here, it is worth mentioning that 2 NRF24 packets are needed to send each public key. The reason for this is that the eliptic curve used in the ECDH implementation chosen for this system uses 64-byte public keys. Therefore, since the maximum payload size of an NRF24 packet is 32 bytes, four packets are needed for this particular key exchange instead of two.

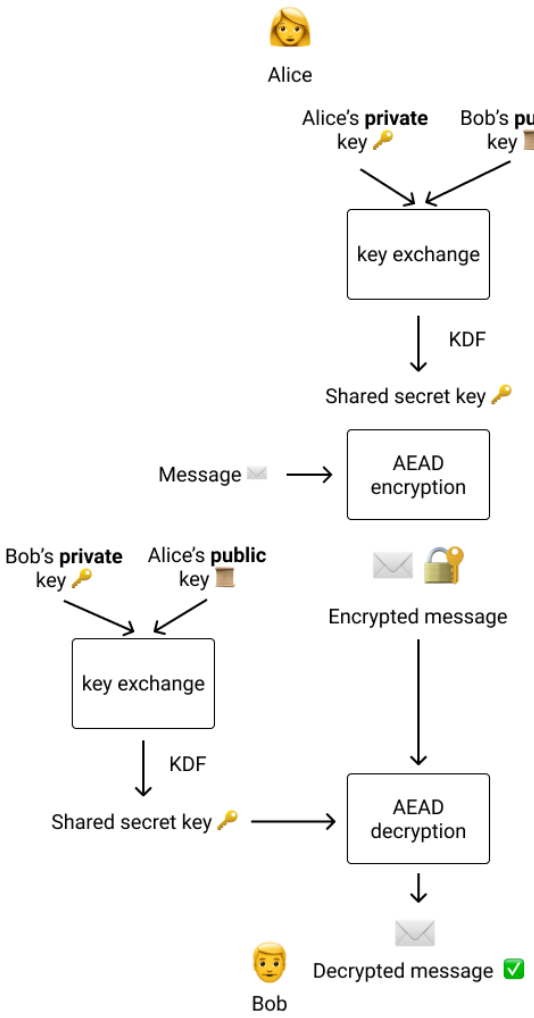


Figure 3.9: Example diagram illustrating the same interaction between ECDH, AES and the Key Derivation Function (KDF) as used in the system. Source: Sylvain Kerkour [37].

In addition, both public keys are sent in intertwined packets, like this:

1. First half of the wireless station ID.
2. First half of the central station ID.
3. Second half of the wireless station ID.
4. Second half of the central station ID.

This structure ensures smooth synchronization between stations' transmissions, and allows lost packets to be detected more easily than with multi-packet reception schemes.

The second main stage of the handshake is **initializing the AES encryption module**. To do this, each station involved in the handshake takes their own private key and the public key of the other station (joining the two packets received previously) and uses the ECDH module to generate a shared secret, i.e., an array of bytes. ECDH guarantees that both stations' secrets match, hence the name "shared". Following that step, the stations pass the shared secret to a Key Derivation Function, which uses the BLAKE2b and ChaCha20 algorithms internally to obtain a pseudo-random key. Since both pass the same shared secret, the key they generate is

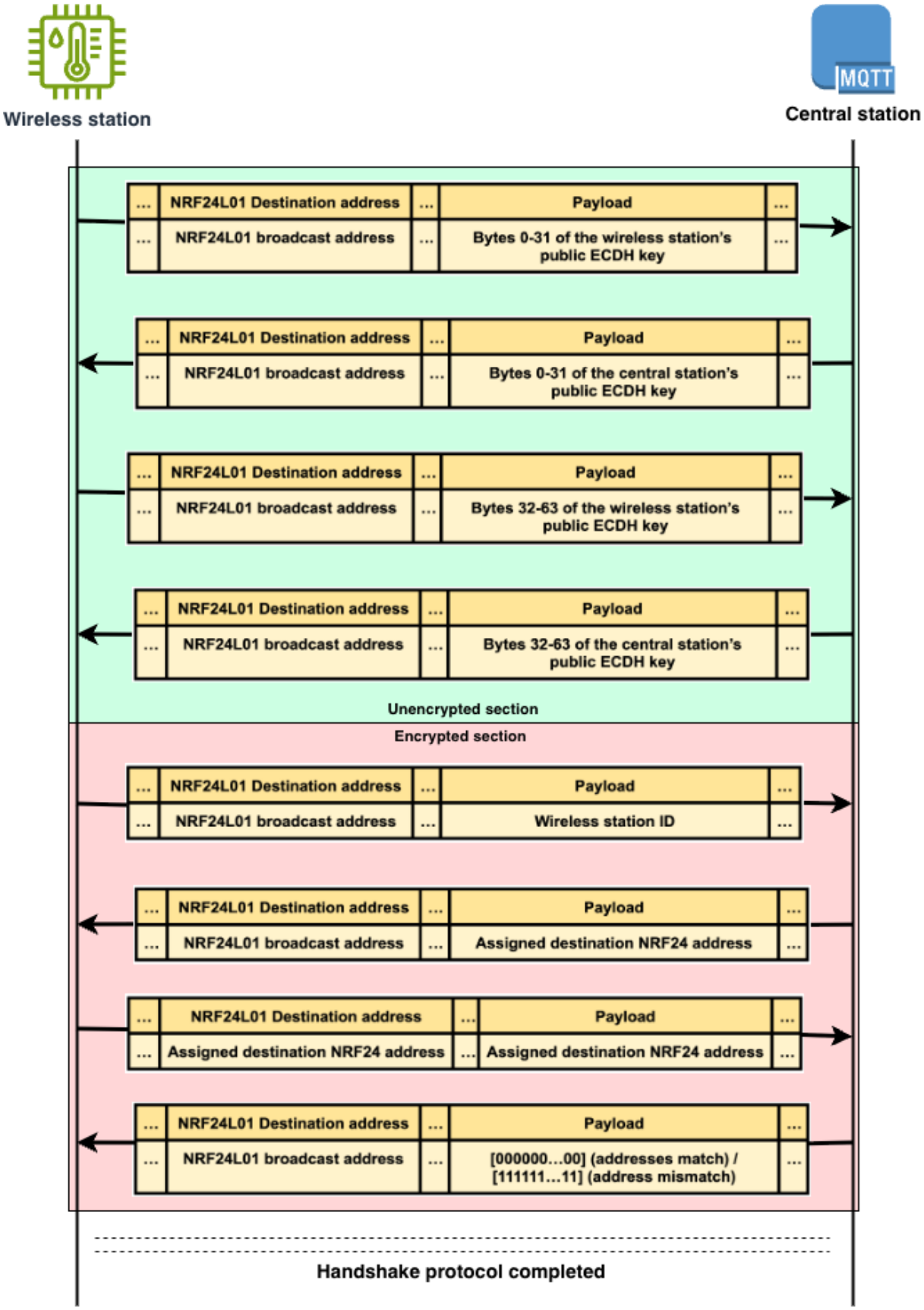


Figure 3.10: Advanced handshake protocol.

exactly the same. Finally, they use that key along with their Initialization Vector and a counter to initialize the AES encryption module. Each of the stations will use this structure to encrypt and decrypt all successive communications between them.

The third section of the protocol handles the **transmission of the wireless station ID and the destination NRF24 address** assigned to it. This step comprises the entire initial handshake, but in the current version of the procedure, the data exchanged is encrypted rather than transmitted in plain text.

Lastly, the fourth and final stage of the handshake is the **acknowledgment of the NRF24 destination address**. In it, the wireless station sends to the central one a packet containing the NRF24 address it received in the previous section. When the central station receives this packet, it obtains that address after decrypting the packet's payload and compares it to the one it had sent before. If they match, it sends the wireless station a packet with all zeros as its encrypted payload; if they do not match, the encrypted packet's payload is made up of all ones. Therefore, upon receiving this packet, the wireless station can verify whether it contains the address assigned by the central station or whether an error of some sort occurred and the handshake failed.

This revised handshake successfully eliminates most of the security vulnerabilities that were present in its initial version. If a malicious third party wanted to detect the ID or spoof the destination NRF24 address sent to the wireless station, it would have to reach the third stage of this protocol. However, even if it had impersonated a central station and successfully completed the key exchange protocol, it would still lack two essential items to complete the second stage:

a) the exact elliptic curve being used during the ECDH, necessary to generate the same shared secret as the wireless station, and

b) the AES Initialization Vector —only available in legitimate stations—, in order to successfully initialize the AES encryption module

Because the Initialization Vector is hardcoded in every station and is never transmitted by radio, the sheer computational power required to crack that vector through a brute force attack makes it virtually impossible for any third party to initialize the AES module with the same parameters as a legitimate station.

Of course, every system has its limitations. This encryption scheme is secure as long as the IV and the elliptic curve remain unknown, which could be considered a slightly naive approach. If they ever became public, any malicious party would be able to successfully initialize its AES encryption module during handshakes, effectively impersonating a legitimate station not only during the handshake, but also throughout the remainder of the communication with the other station.

Nevertheless, solving this issue would require implementing some type of authentication mechanism that stations could use to unequivocally validate other station's identities. One possible solution would be using certificates, similar to more advanced protocols like Transport Layer Security (TLS) [59]. In any case, implementing this feature falls far beyond the scope of the project, and it will therefore only be noted as future work.

The AES CTR counter

There is one additional subtlety that needs to be mentioned regarding how packets are encrypted in the system. As mentioned before, from the moment that both stations finish the second stage of the handshake and successfully initialize their AES modules with the same derived key, all their following communications are encrypted. This applies both to the remaining packets sent until the end of the handshake and to all subsequent packets containing the wireless station's readings (assuming the handshake is completed successfully). However, since the system uses AES CTR, each encryption and decryption operation requires the use of a counter. Moreover, decrypting a packet requires a station to know which was the value of the counter used to encrypt it, and herein lies the problem.

A pragmatic approach was used to resolve this issue. After the handshake's key exchange, all packets sent throughout the lifetime of the association between a pair of stations are well below the 32 byte NRF24 payload limit. Therefore, the system leverages this and, in every message, appends the four bytes containing the value of the counter to the trailing end of the encrypted data sent in the payload. After using it to encrypt one payload, the counter in the receiving station is increased by one. On the other end of the channel, the receiving station extracts the plain text counter from the trailing four bytes of the received packet's payload, and uses it along its AES module to decrypt the encrypted section of the payload where the packet's actual data is enclosed. See Figure 3.11 for a graphic overview of the different packet payload configurations allowed in the system, including those that use the trailing four bytes as the encryption counter.

An additional aspect of this mechanism must also be addressed. Since central stations can have several associated wireless stations, coherence with the mechanism described above indicates that they would have to store a different AES counter for each of their associations. This value is stored in the same buffer where central stations keep the mapping between each of their associated wireless stations' ID and NRF24 destination address.

3.4.4 Central station connectivity

Central stations are the backbone of the entire architecture, acting as relays between wireless stations and the rest of the system. Up to this point only the interactions between these two station types have been covered, and no description on how central stations interact with the other side of the system has been provided. For this reason, the current section offers a comprehensive guide to the higher-level architecture that allows the central stations to update the system database with all the sensor readings they have access to, both from their own sensors and from the readings sent to them by their associated stations.

Mongoose

One of the main features of the Pico 2 W board is its wireless connectivity capabilities through the onboard Infineon CYW43439 chip. This allows the Pico 2 W to connect to a WiFi network, as well as to connect to other devices via Bluetooth. The WiFi functionality is most relevant for the system, as central stations use it to access the Internet.

The interface that allows the Pico to use the Infineon chip is Mongoose [44]. This software provides a full range of network-related functionalities and it is designed specifically for embedded systems. Moreover, it greatly automates the handling of intermediate layers of the TCP/IP model, making it quite simple for low-level embedded systems to interact with higher layers of

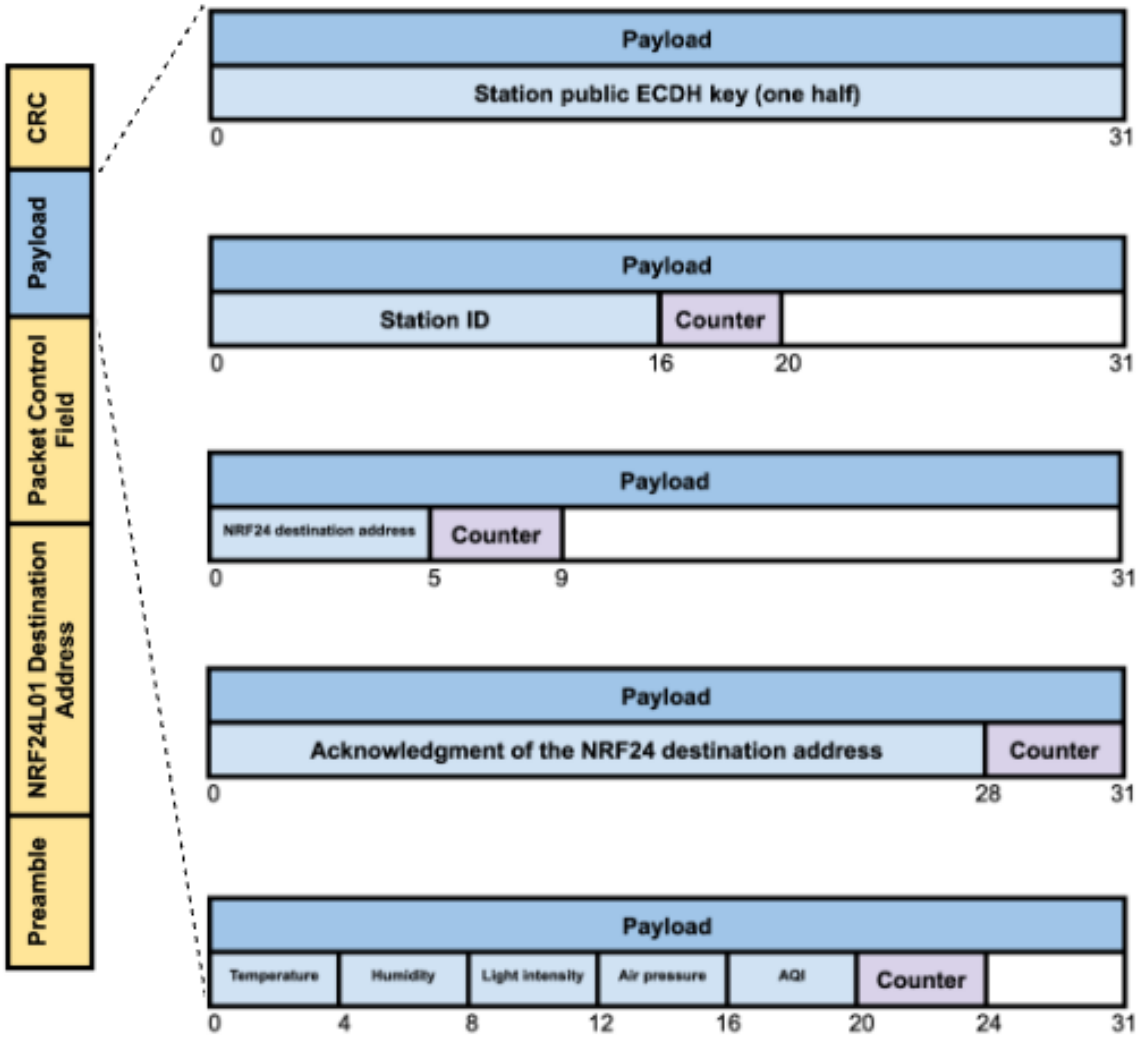


Figure 3.11: Set of the different packet payload configurations supported by the system, both encrypted (with the *Counter* field) and unencrypted.

that networking stack.

Central stations have two main use cases for the Mongoose firmware:

a) Connect to a local WiFi network.

b) Establish an connection with an MQTT broker and publish station readings to it.

Indeed, central stations use the **MQTT** protocol as the channel through which all data collected by the stations is uploaded to the system database. Its great compatibility with IoT projects and the tolerance to faulty client connections provided by its publish/subscribe architecture played a decisive role in the adoption of this protocol over other alternatives.

MQTT is conceptually simple. Devices publish data to a broker and other devices subscribe to a topic to receive data. In this system, stations serve as publishers, but there still needs to be a broker to whom they connect that receives the data they send.

Although there are many different ways to implement a broker, a cloud-based one was chosen for this system, mainly to avoid spending extra time configuring a self-hosted broker. In particular, the **ThingsBoard cloud** environment offered a very attractive free plan with a

vast number of useful features, such as customizable dashboards, rule chains and device profiles. This, combined with its straightforward onboarding and setup, were the key reasons why it was chosen as the MQTT broker.

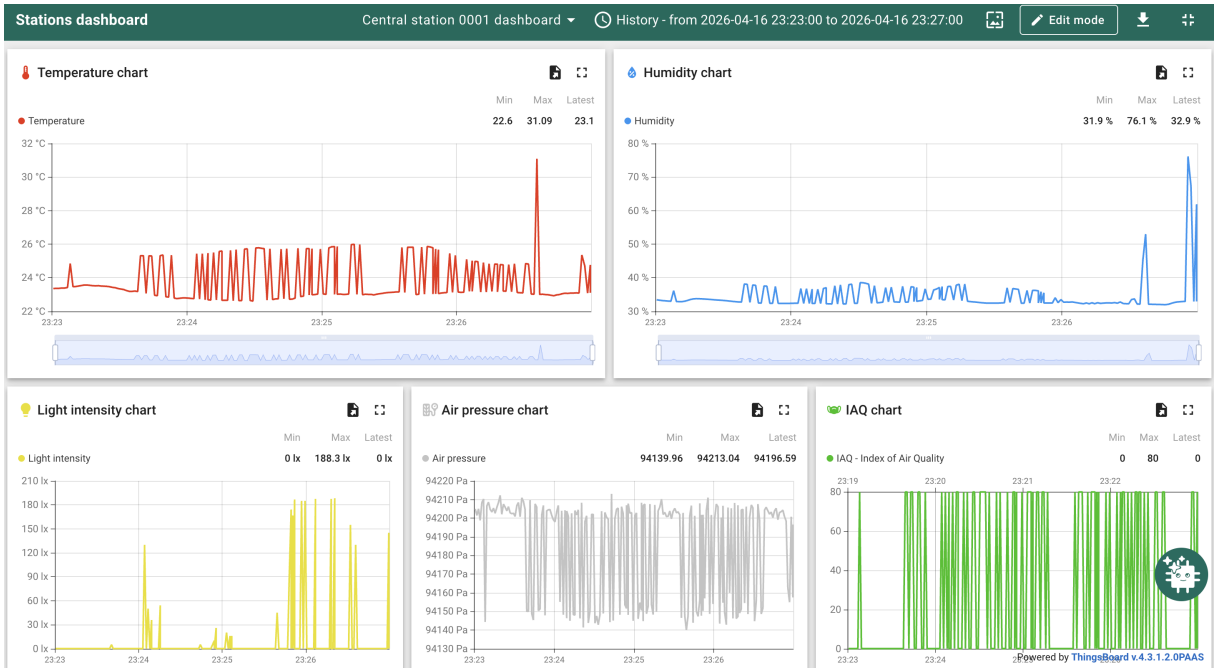


Figure 3.12: ThingsBoard customizable dashboard showing data plots from one central station.

One note about ThingsBoard must be added. ThingsBoard cloud node *acts* as an MQTT broker, thereby clients such as this system's central stations can indeed connect to them via MQTT. Nevertheless, it is not a traditional, general-purpose MQTT publisher/subscriber router, but rather an MQTT-compatible endpoint. Nonetheless, although it is not technically an MQTT broker, it effectively fulfills the same role and will therefore be treated as such within this scope of this project.

After configuring basic ThingsBoard-MQTT credentials for the central stations, the ThingsBoard cloud endpoint successfully ingests station readings, proof of which is the dashboard shown in Figure 3.12. However, all that data still has not reached its final destination, i.e. the system database. Fortunately, ThingsBoard offers a powerful feature called "Rule chains", with which users can create entire data management pipelines.

Using this utility, a new rule chain was implemented with the goal of uploading incoming environmental measurements to the central database (Figure 3.13). It takes the station readings in JSON format as input and invokes the endpoint of the system's API that is in charge of storing those readings in the database. By having an external API that contains all the database connection logic and its credentials, the system becomes more decoupled and adheres better to the principle of Separation of Concerns (SoC).

3.4.5 Internal system API

As mentioned before, an **Application Programming Interface (API)** was needed to manage access to the database and uploading of stations' environmental readings. Despite this being currently its only use case, this interface could easily be expanded with new endpoints if the system were to be extended, in which case it would be instrumental in scaling up the project. This is the underlying reason why considerable effort was put into developing such a small API.

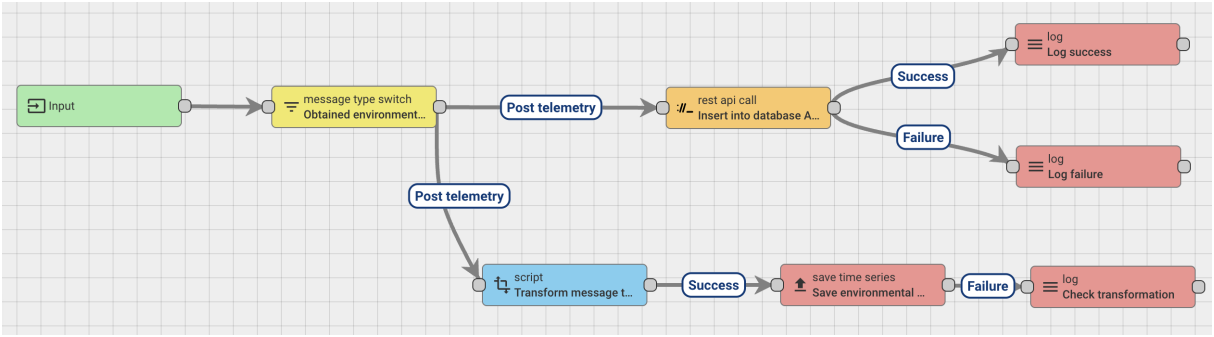


Figure 3.13: ThingsBoard rule chain for inserting data into the database.

Regarding its technical aspects, it is worth mentioning that it was developed using the **FastAPI** [29] framework and it is deployed on **Vercel** [76]. The API itself has only one endpoint that is used to insert sensor readings into the database. Additionally, not only does it store the necessary credentials to connect to the database, but it also stores an API key that filters incoming requests: those that do not include the exact same key are discarded, and the rest (i.e. the ThingsBoard client, which includes it in its endpoint calls) are accepted and the readings included in their payload are inserted into the database. Figure 3.14 shows the structure of the discussed API endpoint.

For a detailed description of the database, its entities and relations, see subsection 4.3.4 "Data architecture".

The code for this API is publicly available in the following GitHub repository: `https://github.com/RoiCorporation/tfg-api` [60]. The repository and all associated materials may be modified at the discretion of the author.

3.4.6 Tests and CI/CD

Any two Engineering projects, no matter how different the technical branches they belong to may be, have at least one thing in common: they need tests. Whether it is a new wing design for an airplane, a classic suspension bridge, or a revolutionary software application, all of them need proper testing mechanisms to ensure that they fit their supposed requirements and that their behavior is deterministic.

However, unlike other software development branches like frontend or backend development, testing embedded systems is not as straightforward as it may seem. The nature of firmware development is such that running a typical test script and checking that all tests pass does not really prove anything about the system itself. The Device Under Test (DUT) might have a shorted path in its circuitry, a component might be silently malfunctioning, a cable might be unplugged without there being any sign of it,... and the test suite might overlook any of these physical, tangible situations.

A compromise had to be made since the system actually needed tests, particularly for some of the logic-related methods. Therefore, it was decided to look for a testing framework that could be used to create and run tests for the purely algorithmic methods, whilst testing the sensors and the rest of the station's features would be performed visually.

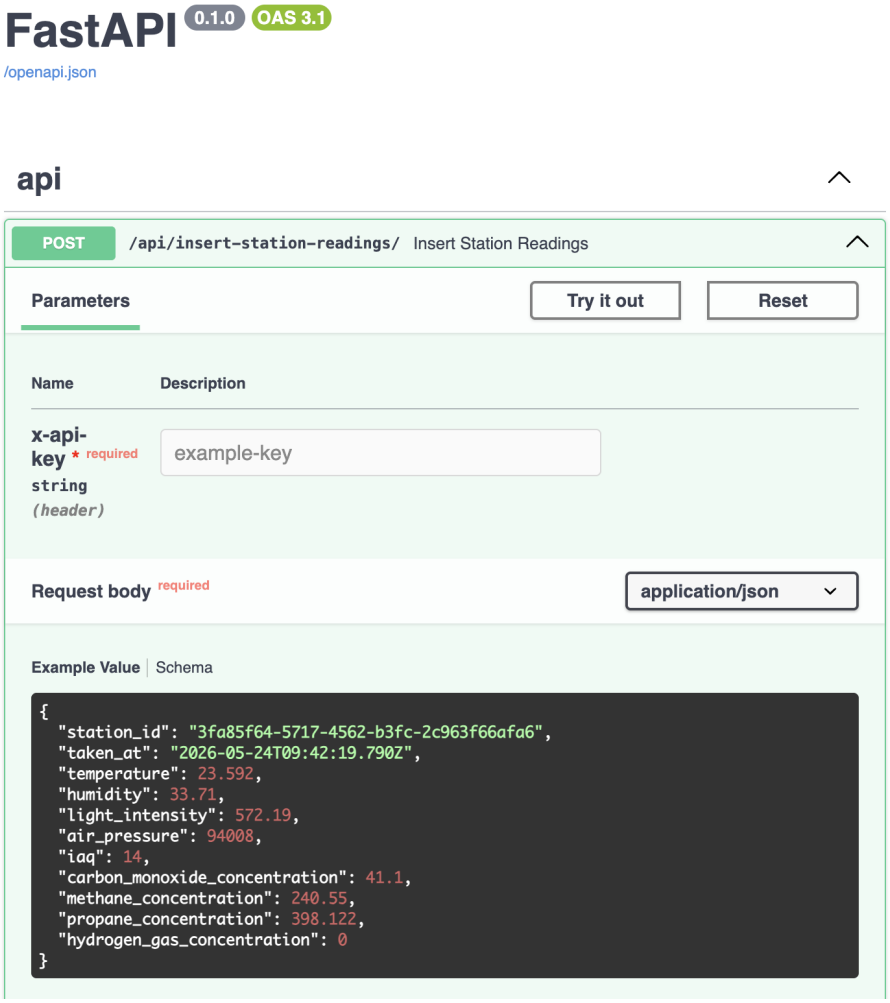


Figure 3.14: FastAPI's automatic documentation of the API's endpoint.

Ceedling

After looking at several different tools to create and run tests in C, a decision was taken to use **Ceedling** [16]. Ceedling is a testing tool that combines the Unity test framework and CMock's mock generation. Although running the tests takes only a couple of commands, it is worth noting that it took a considerable amount of time to set up this tool and configure it to run the first tests.

Nonetheless, credit is due to the developers of this tool. Mainly because of time constraints, only a limited number of Ceedling's features have been explored. There is a real possibility that some of the components it provides that were not used in this project could help with the limitations of testing in embedded systems described above. This hypothesis remains to be confirmed in future work.

Tests

As discussed previously, testing of the wireless station firmware has so far been restricted to logic-focused methods. In particular, the functions that calculate when an environmental hazard appears are those for which tests have been developed.

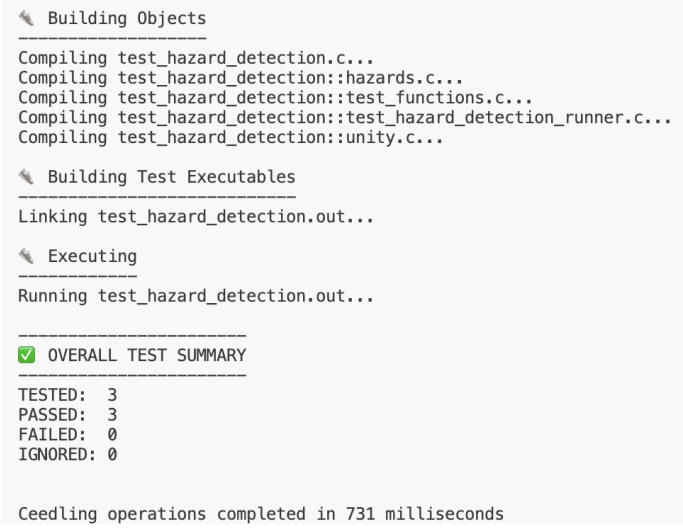


Figure 3.15: Results of running the test suite with Ceedling.

These methods are in charge of detecting hazards and their results are used by the system to trigger audible alarms, which is why they are arguably one of the most important features of the system. Because of that, it was considered ideal that, despite the limitations described above, tests should at least cover this feature.

Consequently, the file `test_hazard_detection.c` was created. It is located inside the `tests` directory and it includes various tests that check that the logic that triggers the aforementioned alarms works as expected. Figure 3.15 shows a successful run of these tests using Ceedling.

CI/CD

Continuous Integration/Continuous Delivery (CI/CD), is a set of practices and tools designed to improve the software development process by automating builds, testing, and delivery/deployment. The main reasons why it was decided to implement a CI/CD pipeline for the aforementioned firmware tests are the same as those for the web application. In addition, the platform, route and structure of the `.yaml` file and the general setup of the workflow are the same as for the web app's `.yaml` file.

A successful run of the CI/CD workflow can be seen in Figure 3.16.

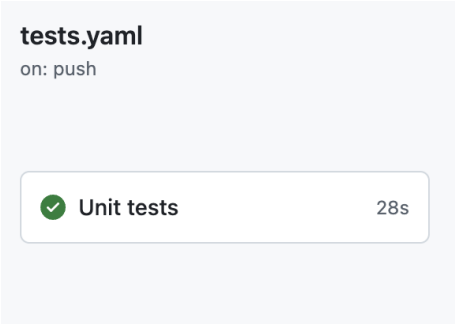


Figure 3.16: Successful run of the C-logic tests workflow.

For a more in-depth explanation of the reasons for designing a proper test suite and implementing a CI/CD pipeline, see sections 4.3.5 "Tests" and 4.3.6 "CI/CD".

Chapter 4

Web application

4.1 Overview

At this stage of the project, the user could easily access the environmental measurements taken by each station simply by looking at its display. Despite being ideal for a small-scale prototype, this approach faced two serious limitations when trying to scale it up to become a feasible commercial system, namely:

- Historical sensor readings could not be accessed through the screen as it only displays the last known value for each environmental variable.

- Sensor readings could not be read remotely, since the user had to physically be next to a central station and check its display to read the measurements.

Indeed, these two conditions greatly reduce the overall effectiveness and usefulness of sensor stations. With that in mind, it was deemed necessary to develop some sort of interface that allows the user to remotely access all readings taken by any of his stations.

After careful consideration, the type of interface that was selected was **a web application**. Not only did the author have previous experience building web applications (having both developed, deployed, and maintained such systems in the past), but also by choosing to develop a web application over alternatives such as a mobile phone app or a desktop app, any device with an Internet connection would be able to use the system, not just only phones or only PCs/laptops.

The following paragraphs contain a functional description of how this web application is built, as well as the particularities of its design and explanations of the various key decisions taken throughout its development that led to the final version.

The code for this application is publicly available in the following GitHub repository: `https://github.com/RoiCorporation/tfg-app` [61]. The repository and all associated materials may be modified at the discretion of the author.

4.2 Design

4.2.1 User management

One of the basic features that the app needs to fulfill its purpose is an effective means of user management. From within the app, any potential user must be able to create an account, sign in, check the readings of his stations, manage those stations, and update his personal information, all of which entail a proper system of user authentication and session handling.

Each of the aforementioned user actions is available inside the app through their respective pages. As such, there is one page for signing up, another for signing in, the Dashboard page to check sensor readings, the Stations Manager page to handle the user's stations, and an Account page (Figure 4.1) for the user to edit his personal information. Any anonymous user can access the sign up and sign in pages, but the rest of those that were just mentioned are only available to logged in users.

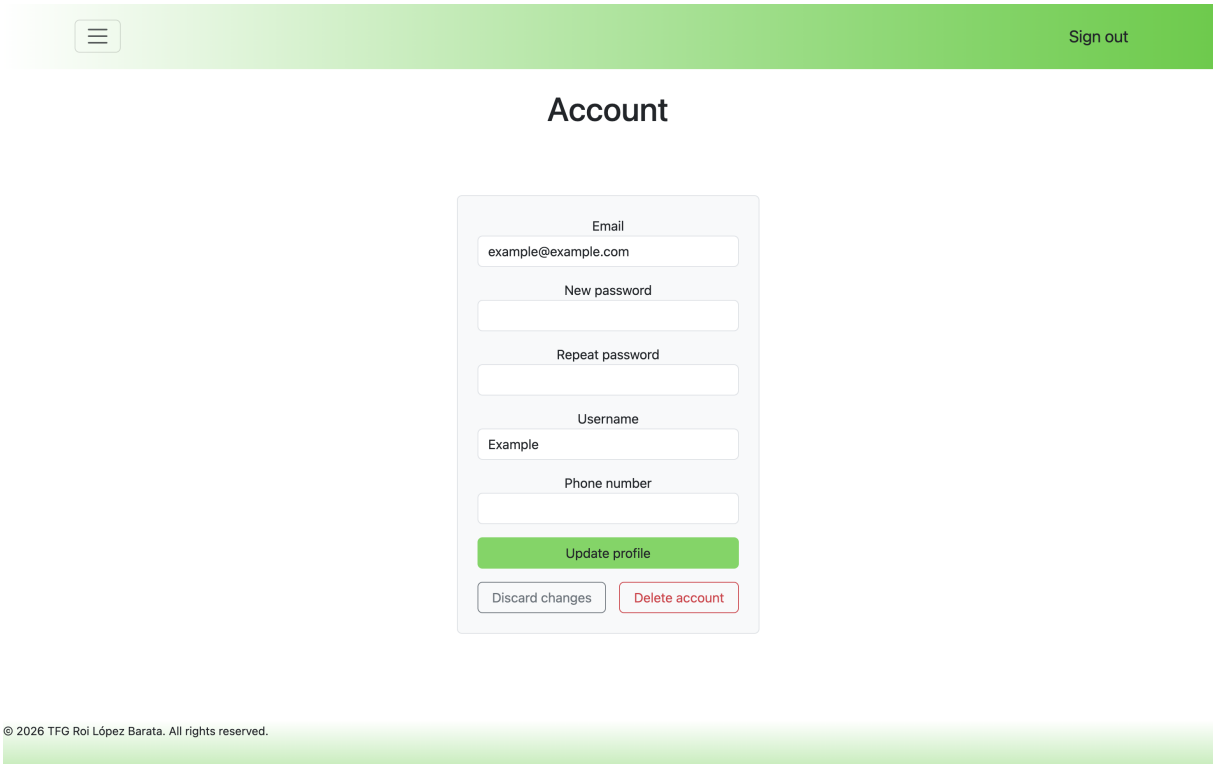


Figure 4.1: Account page where a user can check and update his personal data.

4.2.2 The Dashboard

As was previously explained, the need to create this application arose when considering the limitations that came with having one display in each station as the only means for the user to access the readings taken by each of his stations. For this reason, it is sensible that the main focus during the build of the application was on the data visualization subsystem.

The data taken by the stations are environmental parameters (e.g., temperature, carbon monoxide concentration), which are continuous variables by their very nature. Due to this essential characteristic, it was decided that the best way to represent such parameters would be using plots. Ideally, each environmental variable would be represented in a different plot, and

all of them would be displayed on the main page of the application. That is exactly the purpose of the Dashboard page (Figure 4.2).

Furthermore, since the user should be able to read historical data from such plots, there is one key feature that these graphs must also implement: when the user hovers the cursor/clicks on a point in the plot, that graph should display the value that the environmental variable it represents had at that point in time. In this way, the user can now determine exactly what the reading at that moment was without having to guess or estimate the magnitude based on the position of the cursor with respect to the graph axes.

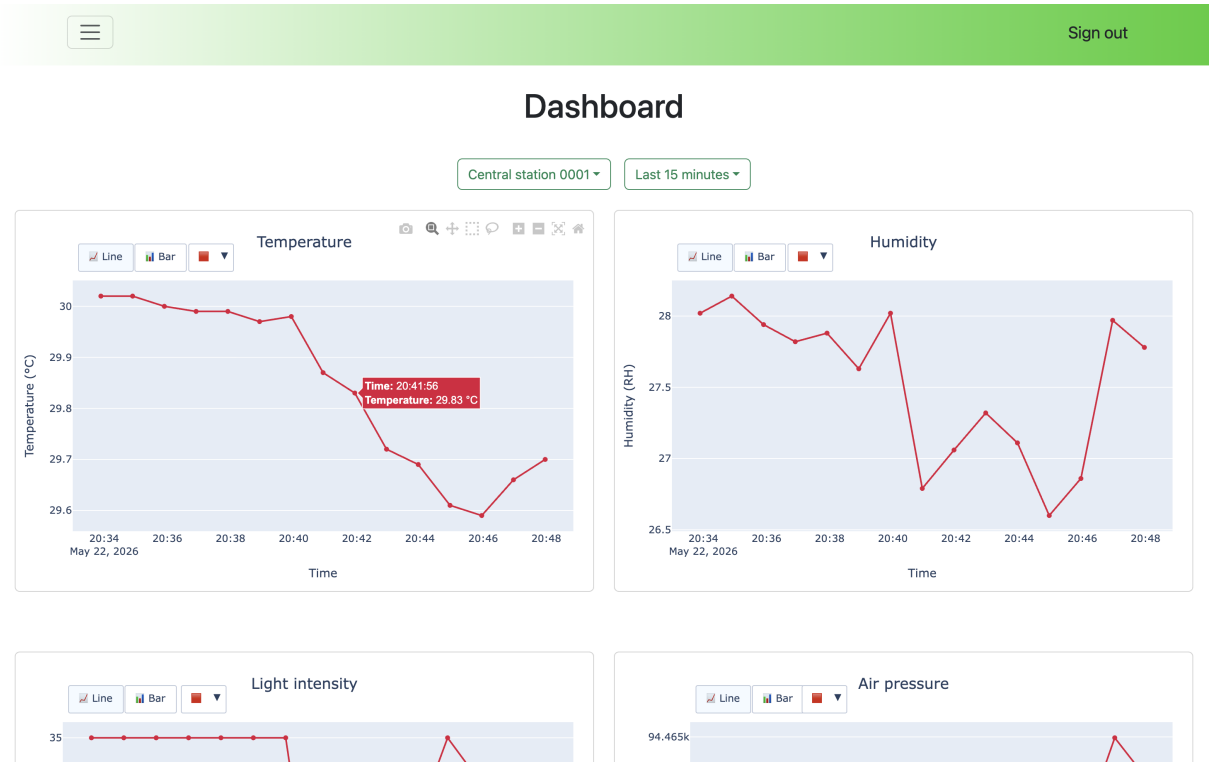


Figure 4.2: Dashboard with the interactive cursor hover feature being displayed.

4.2.3 The Stations Manager

The Stations Manager (Figure 4.3) was developed in order to ensure that all the user's stations can be managed from a centralized place inside the app. Through this page, users can connect a new station, associate a wireless station with one of their central stations, check the details of each of their stations, update the names of their stations, and delete any station they have linked to their account.

Although apparently simple, this page is a fundamental element of the whole app. Without its functionalities, users would never be able to register the physical stations as theirs, which means that no sensor readings would be shown on the Dashboard page, rendering the entire app useless.

4.2.4 Core app pages

Aside from the functional pages that were described in the previous sections, this system could not be considered a proper web application if it did not have some of the classic or *core* pages

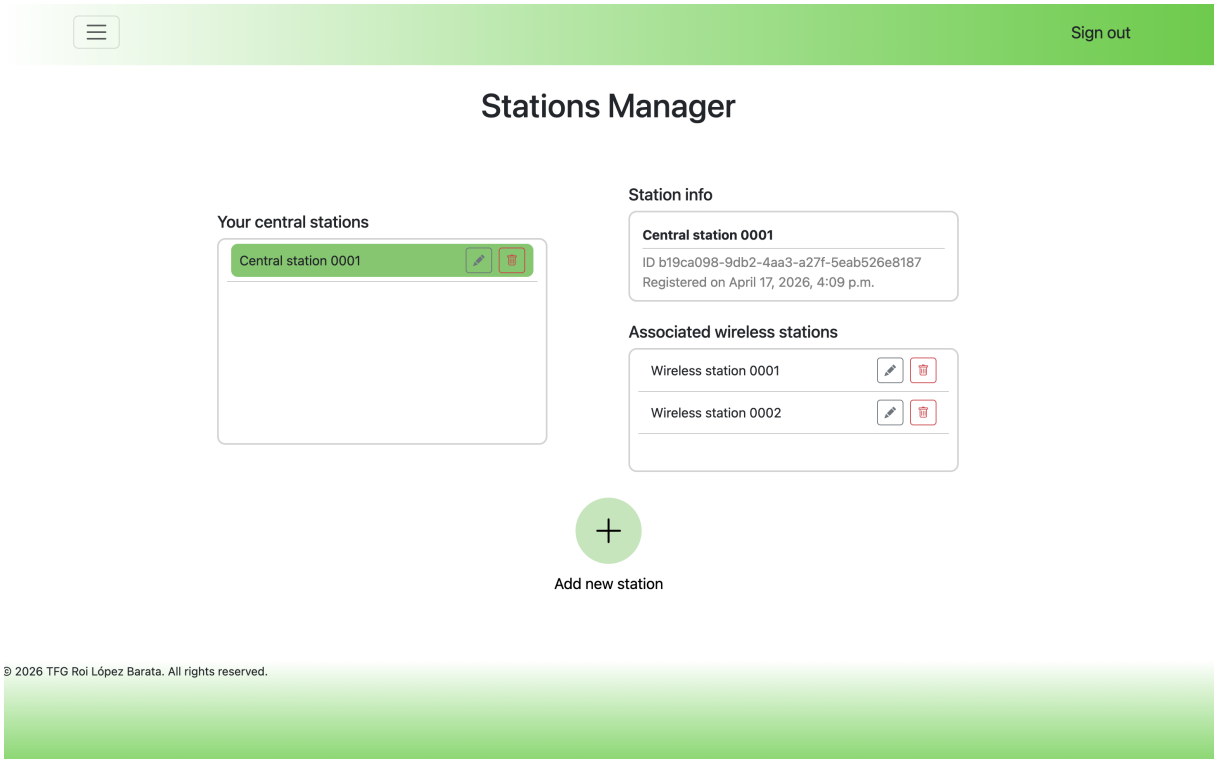
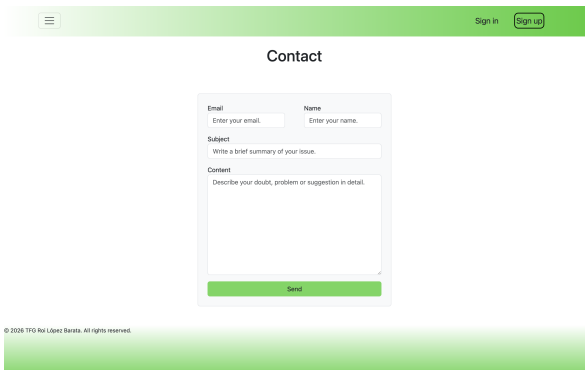


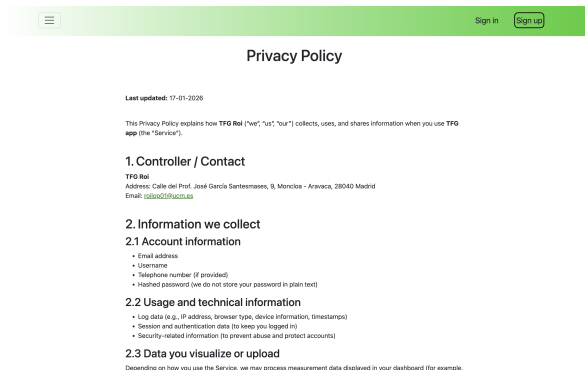
Figure 4.3: The Stations Manager page.

that are present on the vast majority of websites on the Internet. These are, as one could imagine, a landing page, a proper contact page and some legal content pages.

For this app, it was decided to develop a simple landing page that would contain a basic manual on how to use the application. For its part, the contact page (Figure 4.4a) ensures that users have a proper communication channel with the site manager. Regarding legal formalities, two separate pages were created, one to include a simple privacy policy (Figure 4.4b), and another to store the terms and conditions of use of the app.



(a) Contact page.



(b) Privacy policy page.

Figure 4.4: Examples of some of the core app pages.

4.3 Implementation

4.3.1 Technology stack

This application is built using several different technologies and frameworks. This section only lists each of them, although some particularly core ones are given more attention in the following sections.

The backend is implemented in **Python** [56] using the **Django** [23] framework, while the frontend and its tests are built and run with **Vite** [77]. The database is **PostgreSQL** [53] and **Psycopg** [55] is used as the Python adapter to integrate Django's Object-Relational Mapping (ORM) database system. Other non-core yet still relevant technologies used in the app include:

- **Plotly** [52], to generate interactive graphs for each environmental variable.
- **Playwright** [51], as the end-to-end test generation tool.

4.3.2 Backend

The choice of the programming language

Despite being first released more than 35 years ago, the Python programming language is still, to this day, and perhaps more than ever, one of the most popular programming languages among developers [50]. Among its many virtues are its straightforward syntax and a faster development process compared to lower level languages (by avoiding some of their stricter requirements). For these reasons, **it was decided to build this web application using a Python-based framework**. Selecting which, however, proved to be slightly harder than expected.

The choice of the web framework

There is indeed a large set of popular Python-based web frameworks. FastAPI, Django, or Flask provide an extensive set of highly reliable frameworks from which to choose. However, not all are ideal fits for developing this application. The widely considered lightweight nature of Flask makes it less suited for larger applications. Meanwhile, FastAPI's relatively steep learning curve, especially with no prior knowledge or experience, was also an important consideration (this app was developed prior to the API described in section 3.4.5 "Internal system API").

After weighing the advantages and disadvantages of the different options, **Django ended up being the chosen framework**. Not having to use a multitude of associated libraries for each of the app modules as would be the case with Flask, the built-in ORM with all its native features and the framework's acknowledged focus on scalability were the main reasons that led Django to be selected as the development framework of this app.

The backend structure

Once Django was selected as the web framework for the application, the next step was to divide the project into different apps. That may sound confusing, which is why a brief clarification is due.

When referring to a Django-based web application, it is very important to understand the difference between two terms: *project* and *application*. As stated in the official Django documentation [22], the term ***project*** describes a Django web application, whereas ***application*** describes a Python package that provides a set of features, such that applications may be reused in various projects. To avoid any misunderstanding, wherever the term *application* might be understood as either the whole web application being developed or the Django-defined Python package, the appropriate definition will be explicitly referred to.

With that semantic clarification done, it should be clear by now that the system has several different modules, each with their respective purposes. For example, the code in charge of managing users is much different from that tasked with providing core features, as they handle different ORM classes, render different sets of HTML templates, process different types of inputs, etc. This is exactly where Django applications —in the Python package sense— become useful (i.e. to divide the codebase logically into groups of components that provide different features).

Therefore, in order to properly structure the Django project, it was divided into three applications or Python packages, described below:

1. **Core**. This application renders the core templates (the Home, Contact, Privacy Policy, and Terms and Conditions pages) and provides the code for the contact page functionality. It does not contain any ORM class and does not make any requests to the database because none of its components accesses the data layer.

2. **Users**. This package contains the code that renders the pages that deal directly with the user (the Sign up, Sign in, and Account pages). Furthermore, it defines those ORM models that directly include a user instance. It also stores the logic to authenticate the user and access the database to perform Create-Read-Update-Delete (CRUD) operations on the users.

3. **Stations**. This application includes the necessary code to render the pages that explicitly handle stations (the Dashboard and Stations Manager pages). Similarly to the Users package, it describes the ORM classes that include instances of station entities and it also contains the logic to access the database to perform CRUD operations on the stations.

4.3.3 Frontend

This web application does not use a dedicated frontend framework like React or Angular. That may be surprising, especially considering how popular frontend-based approaches to web development have become in the past few years. There are basically two main reasons why such frameworks were discarded:

- Originally, there was no plan to develop this app as a core feature of the project. Instead, building an app was considered a rather interesting add-on for a later stage in the development of the entire system. Therefore, when the app was begun, the goal was to lighten the workload it required as much as possible, precisely because at the time it did not look like it would become as core to the project as it did. This leads to the second reason...

- The author had no previous experience with any frontend framework. Therefore, developing the app using any of those frameworks would require learning an entirely different development environment solely for frontend work, while already being on a tight schedule.

Indeed, instead of using a framework for that section of the project, a more "traditional" route was chosen: the classic combination of backend-generated HTML templates to build the actual pages and its content, and JavaScript files to handle the behavior of dynamic components, form field validations, and asynchronous server calls. As a side note, it should be mentioned that this part of the app uses Vite as the builder tool.

Each of the aforementioned Django applications that make up the website has their own set of JavaScript files, ensuring that the client-side logic remains self-contained. Furthermore, both the **Users** and **Stations** apps contain a test folder that stores unit tests for their respective client-side utility functions.

4.3.4 Data architecture

> *"Data!data!data!" he cried impatiently. "I can't make bricks without clay."*
>
> (Sir Arthur Conan Doyle [24])

Just as detective Holmes needed to obtain data to solve his cases, so does this system need to adequately store whatever data it generates. At first glance, a naive observer could think that the only information worthy of storing would be the environmental readings taken by the different stations. Nevertheless, that is not the only type of information that the app needs to function properly, since it also needs to take into account users' personal information, credentials, stations' identification details, etc.

In order to successfully integrate all these different data entities, a meticulous analysis of the interactions between them was conducted. The following items contain the results of that process, which became the rationale behind the app's data architecture.

- To create an account, the new user must provide an email, a username and a password. Optionally, he can also enter a phone number.
- When a registered user wants to log in, he only needs to use his email and password as credentials.
- Only registered users can add new stations to their account.
- When connecting a new station, the user must always include the station's ID, a name or alias for it and whether or not it is a central station. Additionally, if it is a wireless station, the user will also be asked to enter the ID of the central station with which the new wireless station will be paired.
- A registered user can rename and remove any of his associated stations.
- If the user deletes a central station from his list of stations, all the wireless ones that are associated with it will also be removed from the system.
- The environmental readings taken by any station will only be available to the user who has registered that particular station.

After considering these behaviors and restrictions, the data architecture for the application was designed. It is represented visually by the Entity-relantionship diagram shown in Figure 4.5, which was used as a template for the system database schema.

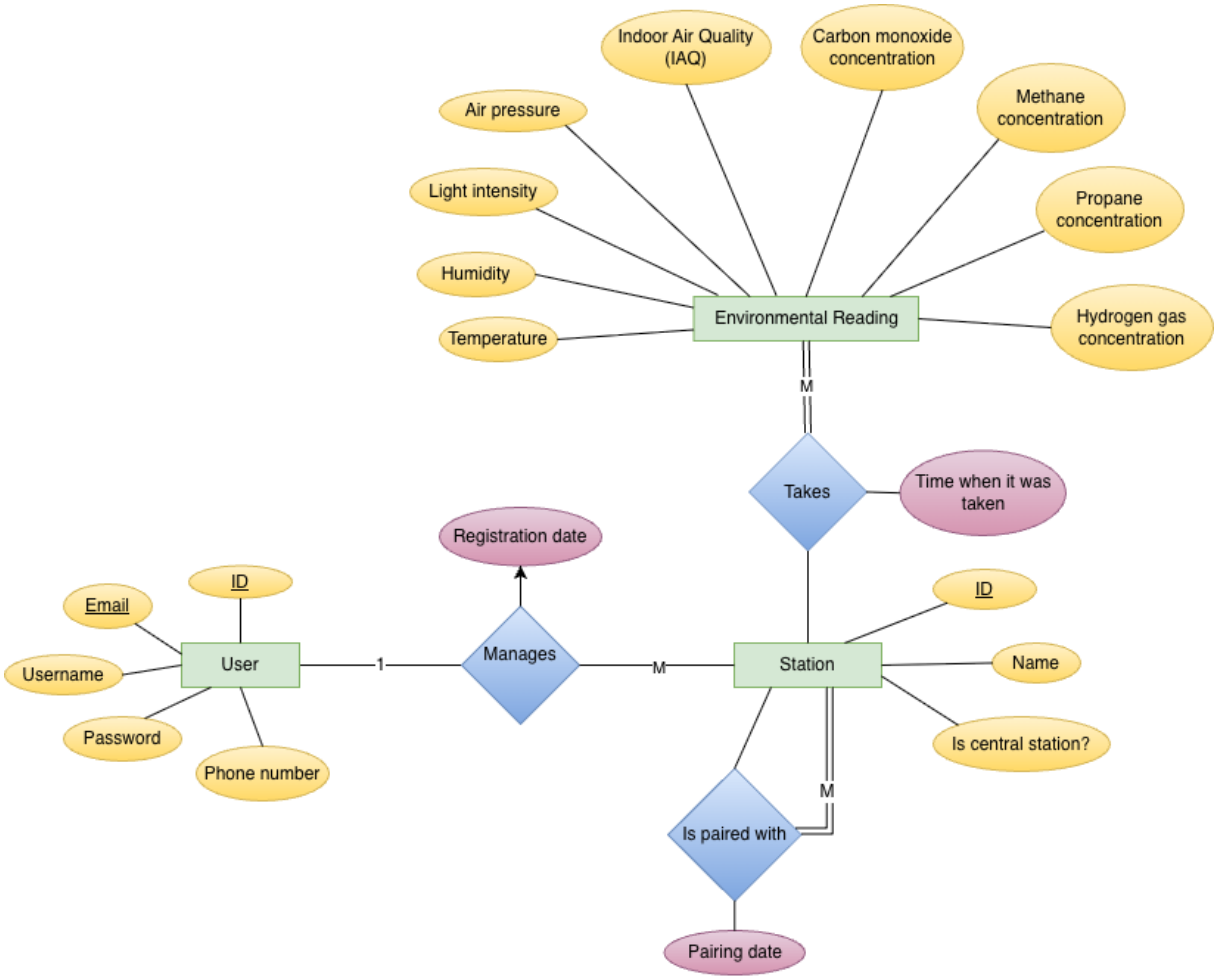


Figure 4.5: The Entity-relantionship diagram that models all the data entities in the the app.

Database provider

To finish off this section, there is one last aspect regarding how data generated by the system is stored that must be addressed. Since this application was developed to be used in a real-world production environment, its data storage mechanisms cannot be based on a local database running on someone's personal laptop. Alternatively, the most appropriate, scalable, and production-like approach is to use a remote database service. **Neon** [47] Postgres database was the chosen storage provider. Given its clear console, straightforward branch creation, and overall performance, having chosen this provider can be considered a good choice.

4.3.5 Tests

Motivation

As mentioned in subsection 3.4.6 "Tests and CI/CD", any Engineering project requires a well defined testing process. For the web app being described, it was clear from the beginning that it could not and would not be considered complete or finished without a proper test suite. Without a comprehensive set of well designed tests, there is no reliable way to guarantee that:

a) The current features of the app work as expected.

b) Any added new features do not break existing code.

With that in mind and with the strong conviction that a solid test framework is key to building high quality software, a thorough and appropriately structured set of tests was needed. This process took around half of the total time that had been allocated to develop the entire application, which is a testament to the importance given to the testing phase of the Software Development Life Cycle (SDLC).

The test suite

The test suite itself is made up of more than 250 different tests. They are structured following a layered approach, as shown in Figure 4.6.

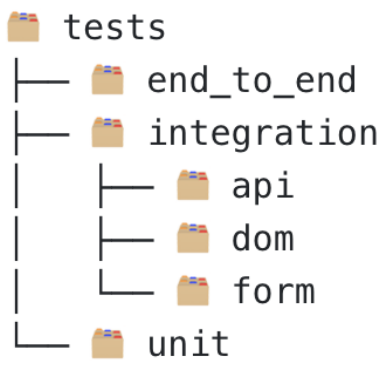


Figure 4.6: Structure of the web application's `tests` folder.

1. **Unit tests**. Stored inside the `tests/unit` folder, their scope is limited to ensuring that app-specific logic features yield the expected results for a given input. For example, they check that plots are generated correctly from lists of dates and environmental values or that the regex-based phone number validator correctly classifies numbers as valid or invalid.

2. **Integration tests**. These are stored inside the `tests/integration` folder and check that subsystems that interact with each other work as expected and are well integrated (hence the name). To better structure this set of tests, this directory is divided into three subdirectories:

   (a) **api**, which keeps tests for the API-like endpoints of both the **Users** and **Stations** Django apps.

   (b) **dom**, which contains tests to check that the content of the different pages is loaded correctly.

   (c) **form**, which includes tests to ensure that the fields of all forms in the app are validated and that descriptive error messages are shown when invalid input is entered into any field.

3. **End-to-end tests**. All tests in this category are stored in the `tests/end_to_end` directory. Each of the tests contains a set of actions that mirror real user interactions with the app. They were designed using the Playwright test framework and the rationale behind these tests is to ensure that all parts and modules of the app work as the end user would expect. By way of example, the workflow in one of these tests consists of going to the Sign up

page, creating a new account, arriving at the Dashboard page, navigating to the Stations Manager, and finally signing out, verifying at each step that the components on each page appear as expected.

4.3.6 CI/CD

Motivation

As was already explained in subsection 3.4.6 "Tests and CI/CD", **CI/CD** is a widely used approach to improve the software development process. Regarding this web application, there are several key reasons why it was decided to design a CI/CD pipeline.

- **Reliability**. Having processes that automatically run the entire test suite every time changes are pushed to the repository provides a systematic and effective way to detect errors and bugs.
- **Scalability**. As was explained before, the mindset while developing this web application was always centered around implementing the same techniques as any real life, commercial software product. Therefore, it is of utmost importance to have methods that ensure that the project is scalable and that its features can be enhanced on every release. This CI/CD workflow ensures that efficiency is not diminished when the project expands, by providing instant error detection and a standard integration pipeline for every new feature.

The CI/CD pipeline

One of the benefits of hosting the project repository in GitHub is the CI/CD tool that it offers: GitHub Actions [75]. This platform runs the CI/CD workflows that are defined in `.yaml` files stored inside the `.github/workflows` folder of the target repository.

The main CI/CD workflow for this app is defined inside the `tests.yaml` file. This file contains three different *jobs*, which is a series of steps that is executed on the same *runner* (basically, every job is a series of commands run on a short-lived virtual machine or container). **Backend tests** is the job that runs the tests for the app's backend, **Frontend tests** is the job that runs the frontend unit tests, and if both of them are successful —i.e. all their tests pass—, the **End-to-end tests** job runs all end-to-end tests. This structure can be seen in Figure 4.7.

This simple yet effective mechanism runs the full test suite after every push to any branch and after pull requests to the main branch, ensuring that bugs are detected instantly and do not remain hidden in the codebase.

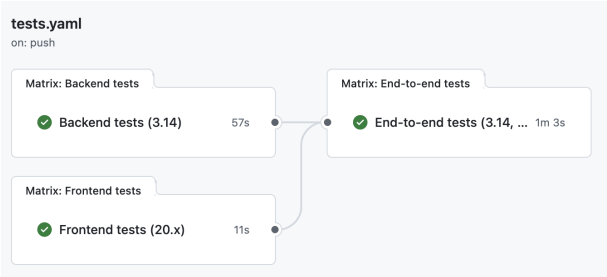


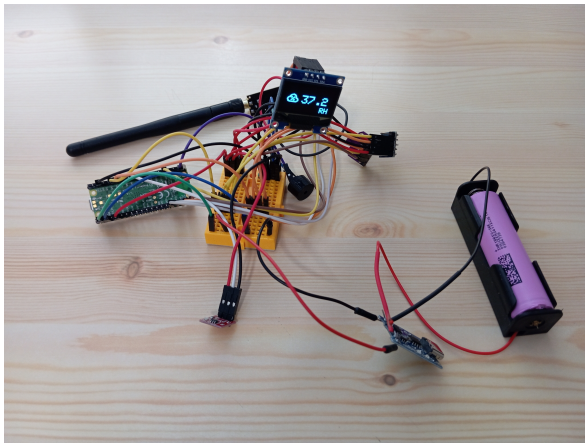
Figure 4.7: Successful run of the application's tests workflow.

Chapter 5

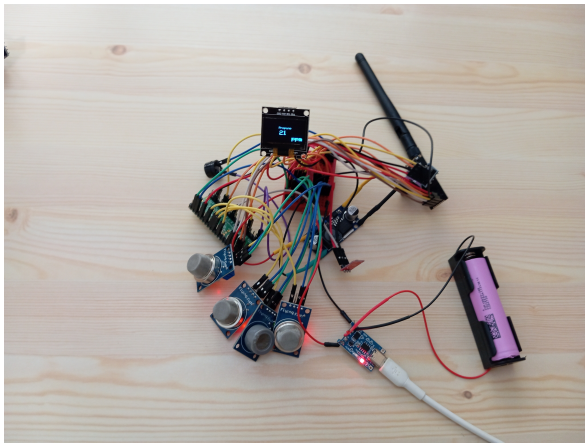
Results

5.1 Completed system

The complete hardware included in each station can be seen in Figure 5.1. The MQ-series sensors in Figure 5.1b represent the main visual difference between the electronics packages of the two station types.



(a) Wireless station hardware.



(b) Central station hardware.

Figure 5.1: Hardware inside wireless stations (left) and central ones (right).

Of course, once the hardware was complete and fully functional, proper enclosures were needed to improve the stations' usability and to reinforce their role as prototypes of a commercially viable product. Furthermore, each station needed somewhere to display its ID, since this is essential for managing these devices in the web application as explained in section 4.3.4 "Data architecture".

For these reasons, 3D casings were designed to enclose each station's hardware, and their ID was engraved on top of the casings. Figure 5.2 shows the two types of casings designed.

5.2 Battery life

One of the key objectives of this project was to ensure that wireless stations had a battery life large enough to ensure continuous operation over several months. Throughout the development

(a) Wireless station 3D casing.

(b) Central station 3D casing.

Figure 5.2: 3D casings for wireless stations (left) and central ones (right).

of their firmware, special attention has been put into making the implementation as energy efficient as possible, in order to lengthen their battery life as much as possible.

To check whether that objective has been achieved and to what extent, it was necessary to monitor the power consumption of a wireless station over several minutes. This would provide enough data to calculate the total time it would take for the batteries to run out of power, by factoring in the station's current draw in both low power and normal modes of operation.

5.2.1 Power profiles

To measure the station's current draw and obtain a power profile, its battery was connected to a Nordic Semiconductor Power Profiler Kit II [65]. In addition, the Power Profiler app included in Nordic Semiconductor's nRF Connect for Desktop [65] framework was the software used to generate and analyze the power profiles.

Figure 5.3 shows the power consumption profile of a wireless station obtained through several station cycles. Two main consumption levels can be identified from that profile. The lowest one corresponds to the station's power draw while in low power mode. The highest level (clearly represented by the short spikes and the central elevation) corresponds to the periods where the station momentarily wakes up from its "hibernation" state (i.e. low power mode), initializes all sensors and components, takes the sensor readings, transmits them via radio and goes back to hibernation.

The station's current draw when in hibernation has been measured at around 550 – 740 µA, depending on which "valley" of the power profile is considered. When in normal mode, the current draw varied from 13.02 to 15.49 mA. These quantities have been obtained by selecting specific segments of the profile shown in Figure 5.3, which are included in Appendix B "Segments of a wireless station power profile".

Before proceeding to estimate a wireless station's battery life based on this data, there are a few important clarifications that must be made with regards to the monitoring process. The first one is that during such process, the period between station readings when the device is put to low power mode was reduced from the standard 1 minute to just 15 seconds. This was done to create more low power cycles in a shorter time, thereby obtaining more power consumption data without actually modifying the underlying behavior of the station.

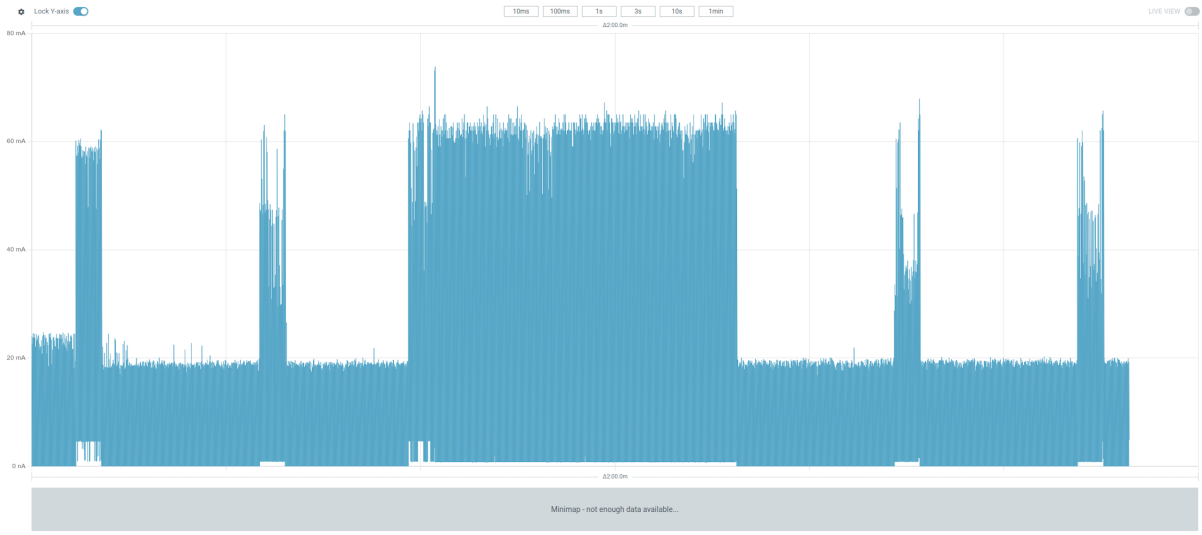


Figure 5.3: Overall power profile of a wireless station after several low power cycles.

The second item that must be addressed is the fact that all the power profiles and subprofiles in this document (both in the current section and in Appendix B "Segments of a wireless station power profile") are plotted in a way that, at least apparently, could confuse the reader into thinking that the actual current draw is much larger than what is stated here. Indeed, by looking at the y-axis of Figure 5.3, one could believe that the current used in low power mode is actually closer to 20 mA, and the current used by the station in active mode is much closer to the 60 – 70 mA range than to the aforesaid 13.02 – 15.49 mA one.

Nevertheless, this effect is simply an illusion. The aforementioned graph must compress in very little width an extremely large amount of data points, due to the ultra high sampling rate of the Power Profiler Kit (100000 samples per second). As shown in Figure 5.4, even in hibernation, some synchronous components in the Pico 2 W are activated approximately once every millisecond, which causes a very brief but noticeable spike in the current. The combination of thousands of these spikes causes the graph to show apparently higher current consumptions than the real values.

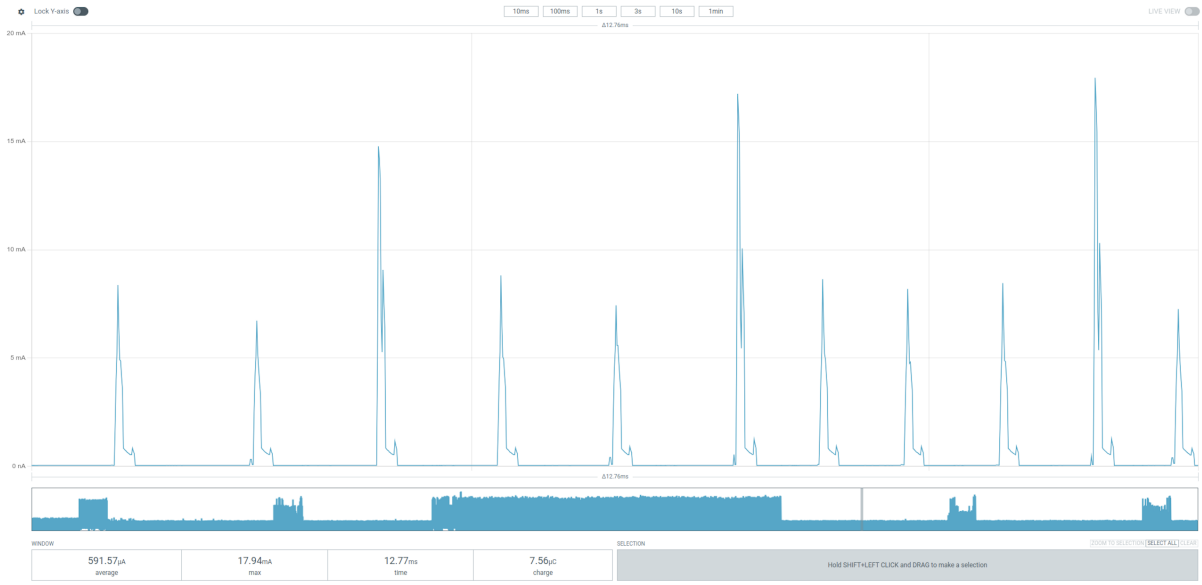


Figure 5.4: Zoomed-in detail of the power profile that shows the periodic current spikes.

The final clarification concerns the unusual hump at the center of Figure 5.3. Although it is only slightly higher that the rest of the non-hibernation regions, it lasts far longer than the latter. Both effects can be explained by the fact that this region of active state was not triggered by the board's wake up timer, but by a short press on the capacitive touch button. As explained in section 3.4.2 "Sensors and components", this action wakes up and initializes the station, takes sensor measurements and transmits them via radio and, crucially, starts the OLED display measurement-showing sequence.

This procedure cycles twice through the entire set of measurements taken by the sensors and displays each on the OLED screen for a few seconds. The sequence lasts around 30 seconds, during which the display —and therefore also the Pico 2 W— is turned on and consuming a few extra milliamperes. These two conditions perfectly match the behavior observed in the central section of the aforementioned figure, thereby explaining the increased duration and current consumption.

Because this sequence is triggered by a human action and occurs very rarely, it is not considered part of the station's normal operating regime. Therefore, this interval will not be taken into account when estimating the station's battery life, as it does not represent a significant portion of the overall operating time.

5.2.2 Estimations

In order to estimate the battery life of a wireless station, it is necessary to first know the average current it draws, I_{avg}. To calculate it, one must consider the ratios of time spent in hibernation and normal or "active" mode, as well as the current drawn in each one. Equation (5.1) defines the mathematical relationship between all these variables.

$$I_{\text{avg}} = \frac{T_{\text{hibernation}}}{P_{\text{monitoring}}} I_{\text{hibernation}} + \frac{T_{\text{active}}}{P_{\text{monitoring}}} I_{\text{active}} = \frac{T_{\text{hibernation}} I_{\text{hibernation}} + T_{\text{active}} I_{\text{active}}}{P_{\text{monitoring}}} \tag{5.1}$$

, where

- $T_{\text{hibernation}}$ is the time spent in hibernation during one cycle, measured in seconds,
- T_{active} is the time spent in active mode during one cycle, measured in seconds,
- $I_{\text{hibernation}}$ is the current draw during hibernation, measured in mA,
- I_{active} is the current draw during active mode, measured in mA,
- $P_{\text{monitoring}}$ is the duration of a full cycle, measured in seconds, and
- I_{avg} is the average current consumption in mA.

Since $T_{\text{active}} = P_{\text{monitoring}} - T_{\text{hibernation}}$, then the equation can be rewritten as follows.

$$I_{\text{avg}} = \frac{T_{\text{hibernation}} I_{\text{hibernation}} + (P_{\text{monitoring}} - T_{\text{hibernation}})\, I_{\text{active}}}{P_{\text{monitoring}}} \tag{5.2}$$

The battery life of a wireless station can be calculated with Equation (5.3).

$$L_{\text{battery}} = \frac{C_{\text{battery}}}{I_{\text{avg}} \cdot 24} \tag{5.3}$$

, where

C_{battery} is the battery capacity, measured in mAh, and

L_{battery} is the estimated battery life in days.

Now that Equation (5.2) and Equation (5.3) have been properly defined, the only thing left to do to obtain an estimate of battery life is replace the variables for actual values. In order to give a conservative estimate, each mode's current draw will be taken from the upper bound of the aforementioned ranges, and the battery capacity will be assumed to be below the nominal 3500 mAh to account for potential discrepancies. Consequently, the values used for each variable are as follows.

$$\begin{aligned} T_{\text{hibernation}} &= 58 \text{ s} \\ I_{\text{hibernation}} &= 0.7 \text{ mA} \\ I_{\text{active}} &= 15 \text{ mA} \\ P_{\text{monitoring}} &= 60 \text{ s} \\ C_{\text{battery}} &= 3400 \text{ mAh} \end{aligned}$$

Based on these values, the average current consumption and estimated battery life of a wireless station are:

$$I_{\text{avg}} = \frac{58 \cdot 0.7 + (60 - 58) \cdot 15}{60} = 1.176 \text{ mA} \tag{5.4}$$

$$L_{\text{battery}} = \frac{3400}{1.176 \cdot 24} = 120.464 \text{ days} \approx 4 \text{ months} \tag{5.5}$$

As proven by the result of Equation (5.5), the estimated battery life of a wireless station is well above the 3 month requirement set in section 3.2 "Requirements". Therefore, it can be concluded that, in terms of battery life and energy management, the project has achieved its goals, as it significantly exceeded the target battery life.

5.3 Insufficient Analogue-to-Digital Converter channels

There is an issue with respect to the MQ-8 sensor and how it is used in the system to monitor hydrogen gas levels that needs to be addressed. Shortly before the project's deadline, it was discovered that the Pico 2 W board only has three Analogue-to-Digital Converter (ADC) channels. Since the four MQ sensors needed to monitor gas concentrations used analog outputs, four ADC channels would be required to allow simultaneous readings of all of them.

Normally, after discovering this issue, the most logical solution would be to connect an external ADC module to the Pico and connect the MQ sensors to that external device. However, this option had to be discarded because the problem appeared too late in the development timeline. Consequently, including this device and the logic necessary to control it added a small but noticeable complexity layer that simply could not be overcome in the limited time available between the discovery of the problem and the final deadline.

Instead, a reasonable compromise was made. Since hydrogen gas is the most unlikely chemical component to appear in a domestic environment, the system uses the MQ-8's digital output,

instead of the analog one. Therefore, the sensor's output still signals when the concentration of that gas exceeds a certain threshold, while eliminating the need for an external ADC. This simplifies the design and allows the remaining MQ sensors to operate normally without complex multiplexing schemes to accommodate four analog outputs using only three ADC channels.

5.4 Contribution to the NRF24L01 driver repository

One of the most important libraries used throughout this project was the "Pico nrf24 driver" library. Publicly available from its GitHub repository [4], it has been instrumental in integrating the NRF24L01 radio module with the Pico 2 W.

Despite its overall good code quality and documentation, since the year it was developed, some issues had been brought up to the author using the repository's Issues page. Most of them had been solved either directly by the original author or by other contributors, once the changes had been approved by the former. However, some unresolved issues remained by the time the driver was used in this project for the first time.

One such open issue was a problem with the conditional logic in one of the methods used to initialize the NRF24 radio transceiver. Due to a small bug inside the `initialise()` method, whenever a user tried to set the transceiver with the default configuration by passing NULL as the argument to that function, the method would fail to configure the device properly.

This problem was a known bug that was first reported in November 2022 [15]. Unfortunately, it seems to have drawn very little attention from the repository's official contributors, since there were no other comments on the thread and no follow-up from the issue's author. Nevertheless, it was important to solve this error, as it was a major flaw in the design of one of the most relevant functions in the entire library.

For this reason, a new post was added to the original Issue thread, detailing the problem, the most effective solution and several unrelated documentation inconsistencies that should also be addressed (Figure 5.5).

Fortunately, after a few days, the driver's author acknowledged and responded to this message. Shortly after, the modifications proposed in the aforementioned post were applied and merged to the main branch of the repository.

One last note regarding this event needs to be mentioned. It is deeply fulfilling to see how the effort put into this project has also led to a meaningful contribution to an open source codebase. Indeed, knowing that this input has helped improve the driver, even slightly, is both a source of satisfaction and an inspiration to continue contributing to Open Source Software (OSS).

5.5 Problems with the NRF24L01 module

During the planning stage of this project, it was decided that communication between stations would occur via Long Range (LoRa) radio modules, particularly with a breakout board based on the SX1278 chip [67]. Among the benefits of these devices were their extremely long range (up to hundreds of kilometers under ideal conditions) and low power consumption [66], which would be very valuable for this project.

Nevertheless, when the time came to implement the communication scheme, using the avail-

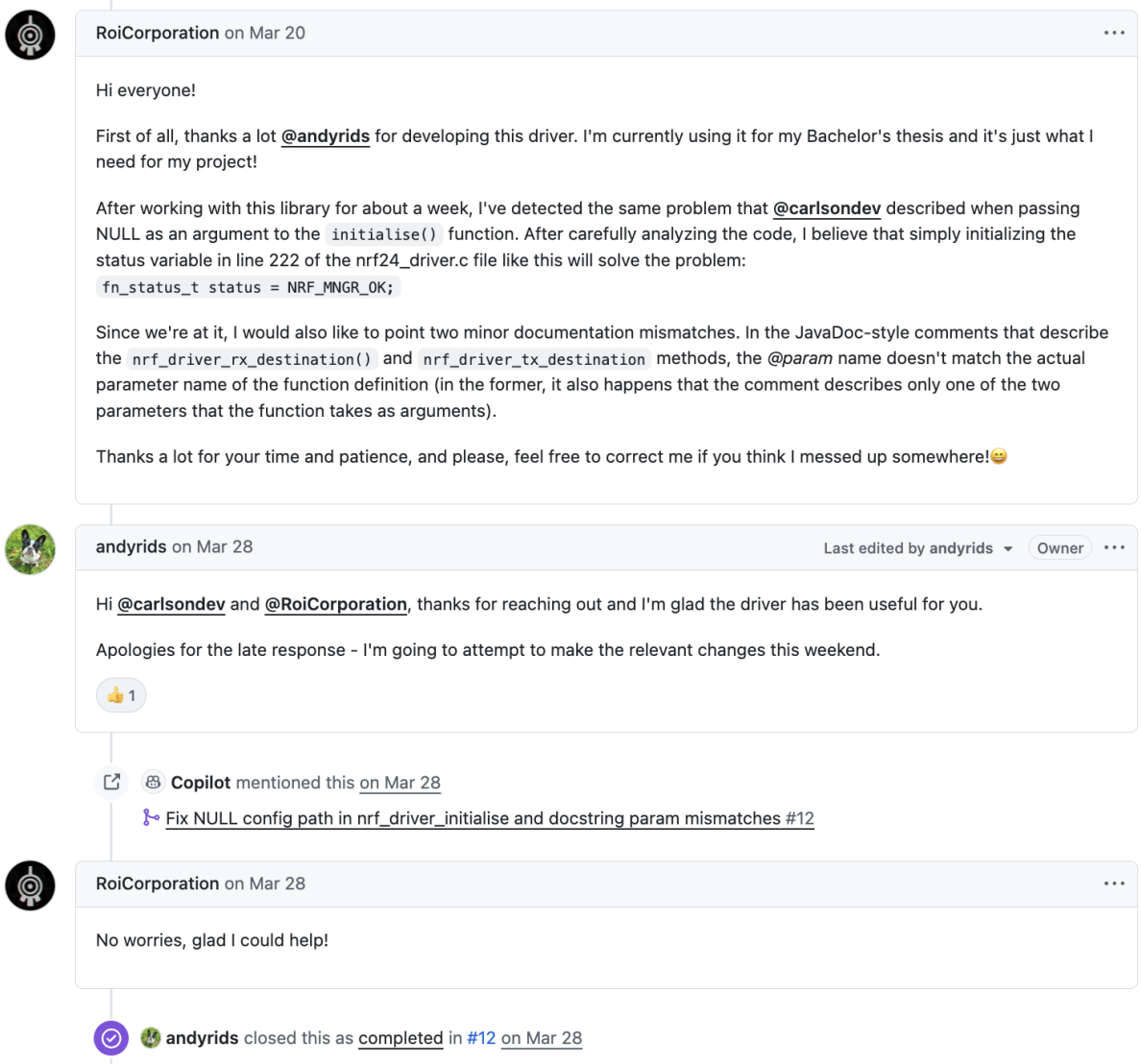


Figure 5.5: Section of the GitHub Issues page showing the aforementioned post, the response of the repository's owner and the subsequent merge of the changes into the main branch. Source: andyrids [2].[1]

able LoRa breakout board with the Pico 2 W proved extremely difficult. There was no apparent way to make the board work correctly with the Pico controller, and after many failed attempts, a different radio module was chosen: the NRF24L01.

The NRF24L01 module, as shown throughout this document, became a cornerstone of the project. It is the component responsible for providing cross-station wireless connectivity, and although it lacks the range and low power capabilities inherent to LoRa devices, it is still a reliable and well supported radio platform.

However, despite its many virtues, this module has one feature that makes it suboptimal for this system: its operating frequency of 2.4 GHz, which is the same frequency band used by Wi-Fi protocols, as specified in the IEEE 802.11 family of standards [34].

At first, this did not pose any apparent problem. The original handshake protocol was simple enough that the two messages exchanged in it were usually transmitted correctly. Nevertheless, when this protocol was replaced by the advanced handshake, the number of successful handshakes decreased significantly. The main hypothesis for this phenomenon is that being indoors

and surrounded by devices that used Wi-Fi caused considerable interference around the 2.4 GHz bandwidth, which corrupted some of the packets transmitted during the handshake and caused the protocol to fail. The difference in the number of packets exchanged in the advanced handshake compared to the previous version of the protocol (8 packets versus 2 packets) could explain the reduced handshake success rate, as the former requires more packets to be transmitted, which increases the probability of a packet being corrupted by interference, causing the procedure to fail.

Fortunately, the NRF24L01 module can detect corrupted packets by checking their Cyclic Redundancy Check (CRC). Moreover, this device already features auto acknowledgment and auto retransmit included in the Enhanced ShockBurst™ data link layer that it uses [64]. These features would be ideal to ensure lost packets were retransmitted, which makes them the most promising potential solution to the aforementioned packet corruption issues.

In theory, the NRF24 driver used in the project includes those two mechanisms, auto acknowledgment and auto retransmissions, inside the `send_packet()` method. However, two hints signal these functionalities are not well implemented (if at all) in the driver.

a) The fact that the `README.md` file describing the driver has the item "Implement auto-acknowledgement with payloads" as an unmarked element in the "TODO" list [5].

b) It was observed during the system's operation that often times, when a wireless station transmitted a packet with environmental data, the `send_packet()` function would return an error, and yet the associated central station would receive the packet successfully with the `read_packet()` method. This is particularly significant in light of the `send_packet()` method documentation:

   A return value of ERROR (0) indicates that either, the packet transmission failed, or no auto-acknowledgement was received before max retransmissions count was reached. ([3])

These two hints strongly suggest that the driver does not currently support auto acknowledgment and auto retransmissions. Since there is no automatic mechanism to validate packets, the only alternative would be to manually analyze CRCs and force the receiving station to send some sort of "forced retransmission" message to indicate the transmitter to resend the packet if the previous one had been corrupted. However, this again is not possible because the NRF24 driver does not provide access to the lower level logic that disassembles packets and checks CRCs.

After thoroughly studying the NRF24L01 specifications and the driver implementation, in combination with observations on the module's behavior taken throughout the project's development, it can be concluded that this device is not the best choice for the system's use case. Its limited payload size —only 32 bytes—, the potential interferences caused by sharing the same frequency as Wi-Fi protocols and the incomplete implementation of the driver used to integrate the Pico 2 W with it are all reasons that make it suboptimal for this system.

5.6 End-to-end web test fails

Section 4.3.5 "Tests" described in great detail the motivation and structure of the web application's test suite. As explained in those paragraphs, this comprehensive set of tests has been a fundamental part of the app's development.

[1] ***RoiCorporation*** is the GitHub username of the author of this thesis, Roi López Barata.

Although designing them was a rather laborious task, once they were implemented and the issues they revealed in the app were fixed, most tests would pass successfully. In particular, unit and integration tests passed consistently in every test run.

However, end-to-end tests were more troublesome. In some test runs, all end-to-end tests would pass, while in others, some would randomly fail. This behavior was consistent across environments, occurring both when running the test suite locally and when executing the GitHub Actions test workflow.

At first glance, one could think that this was somehow related to the database. After all, a faulty connection to the test database could explain the occasional failures. However, this explanation was soon discarded because a new ephemeral local test database is created and populated specifically for end-to-end tests with each run.

After analyzing the logs of failed tests, it was determined that the failures were caused by timeouts occurring when finding certain page elements (for example, error messages expected to appear after a user entered invalid data into a form field). Nevertheless, outside the end-to-end test environment, those same elements appeared correctly when required, as attested by the fact that all integration tests covering the same routines passed successfully. It would seem as if this was a problem of how Playwright was running these end-to-end tests, since even when timeouts were increased, Playwright would still fail to locate the aforementioned elements.

This behavior affects the usefulness and practicality of the test suite, as the random outcome of end-to-end tests makes them essentially useless. Although considerable effort has been devoted to resolving this issue, no explanation for this unexpected behavior of the end-to-end tests within the Playwright environment has yet been found. More work is needed to clarify the causes of this phenomenon and restore a fully functional test suite.

Chapter 6

Conclusions and future work

6.1 Conclusions

The system developed for this thesis provides an effective means for automatic indoor monitoring of environmental conditions. These conditions include temperature, humidity, carbon monoxide concentration and an Air Quality Index (AQI), among others. Two types of sensor stations equipped with the devices required for this monitoring have been developed: central stations and wireless stations. Both take measurements of basic environmental conditions and central stations also monitor the concentrations of some gases (carbon monoxide, propane, etc.). Moreover, the system is organized into networks of stations, where wireless stations are associated with central ones in a star topology and communications between them happen through encrypted links. Stations use this structure to connect to the system database and upload their readings to it.

Additionally, a web application has also been developed as part of this project. This application is connected to the system database, allowing users to visually monitor both current and historical readings of all environmental parameters taken by any of their stations.

Overall, the system provides the functionalities that were originally proposed. Furthermore, the system has exceeded expectations in ensuring adequate battery life for wireless stations, and a complex, multi-layered encryption scheme has been developed to ensure secure communications between stations. In addition, commercial viability is a realistic prospect, especially given the enhanced usability provided by the web application and the low manufacturing cost of each station (approximately 45 € for each wireless station and 60 € for each central station).

6.2 Future work

There are a handful of features that could be used to extend the system's functionality. This section includes a list of those identified during the development of the project that could not be completed due to time constraints or because they fell outside the scope and objectives of this thesis.

- **Alternative radio modules.** As explained in section 5.5 "Problems with the NRF24L01 module", **the NRF24L01 transceivers used in this project were not the ideal radio modules for the system's use case**. Therefore, any further work on this project would necessarily need to consider using other devices for radio communications, ideally

with a larger payload size and a different operating frequency than the NRF24L01.

- **Over-The-Air (OTA) updates.** This feature would allow the system administrator to **remotely update stations with new firmware**. At the current stage of the project, implementing this feature in central stations would not require large structural changes, as ThingsBoard's client already has a built-in OTA endpoint. Wireless stations, however, would indeed require further work to be able to use this feature, since they are not connected to the rest of the system via MQTT.

- **Custom Printed Circuit Board (PCB).** Although this feature was considered in earlier stages of the project, it was discarded because it was outside the scope of this thesis. Nevertheless, significant benefits could be achieved by developing a custom PCB integrating all station components. Not only would the device be made much more compact, but the Pico 2 W could also be stripped of unused GPIO pins and features, thereby simplifying the schematic and further reducing current consumption.

- **Authentication certificates.** One of the vulnerabilities of the encryption system implemented in this project is the absence of an authentication mechanism to validate participating stations. As described in section 3.4.3 "Communication between stations", a malicious third party that knows the IV and the elliptic curve used in the system can initiate or participate in handshakes. This represents an important flaw in the encryption scheme and would require immediate addressing if further work on this project were to take place. **Digital certificates** are proposed as a potential authentication solution, but other approaches should be considered.

- **Sensor calibration.** MQ sensors are not very accurate and require calibration to indicate proper readings. This project has used approximate values to adjust the sensitivity of these devices, but further work is required to calibrate them using more appropriate and professional tools.

- **Web alarms.** Just as buzzer alarms provide audible signals that people can hear when they are near a station, a similar alert system could be implemented within the web application. For example, warning emails could be sent to a station owner if any of his stations detects a potential hazard. This functionality would improve even more the system's usability, which is why it should be considered as a potential addition in future work on the project.

Acronyms

ADC Analogue-to-Digital Converter. v, 49, 50

AES Advanced Encryption Standard. 24–26, 28, 29

API Application Programming Interface. iv, vi, 14, 31–33, 39, 43

AQI Air Quality Index. 14, 16, 18, 20, 54

CI/CD Continuous Integration/Continuous Delivery. iv, v, 32, 34, 42, 44

CRC Cyclic Redundancy Check. 52

CRUD Create-Read-Update-Delete. 40

DUT Device Under Test. 32

ECDH Elliptic Curve Diffie-Hellman. 24–26, 28

GPIO General Purpose Input/Output. 16, 55

HTML HyperText Markdown Language. 40, 41

I²C Inter-Integrated Circuit. 12, 16, 18

IEEE Institute of Electrical and Electronics Engineers. 51

IoT Internet of Things. iii, 8

IV Initialization Vector. 28, 55

KDF Key Derivation Function. 25, 26

LNG Liquified Natural Gas. 5

LoRa Long Range. 50, 51

MCU Microcontroller Unit. 12, 16–18, 21

MQTT Message Queuing Telemetry Transport. 14, 30, 31

ORM Object-Relational Mapping. 39, 40

OSS Open Source Software. 50

OTA Over-The-Air. 55

PCB Printed Circuit Board. 55

SDLC Software Development Life Cycle. 43

SoC Separation of Concerns. 31

SPI Serial Peripheral Interface. 12, 16, 20

TLS Transport Layer Security. 28

UUID Universally Unique IDentifier. 14

VOC Volatile Organic Compounds. 5, 18

Glossary

anonymous user user of the web application that has not created an account yet . 36

data layer set of software entities (methods, variables, etc.) of the web application that directly interact with the system database . 40

django application Python package within the Django web framework that provides some set of features and may be reused in various projects . 40, 41

Enhanced ShockBurst™ Nordic Semiconductor wireless protocol for reliable low-power data transmission . vi, 22, 52

entity-relantionship diagram graph that represents all the data objects and their relations in a given system . vi, 41, 42

transceiver device that can act as both transmitter and receiver . 22, 50, 54

Bibliography

[1] U.S. Enegry Information Administration. *Electricity use is becoming more common for residential heating.* URL: https://www.eia.gov/todayinenergy/detail.php?id=66324.

[2] andyrids. Accessed: 21-05-2026. 2026. URL: https://github.com/andyrids/pico-nrf24-driver/issues/2.

[3] andyrids. *nrf24_driver.c.* URL: https://github.com/andyrids/pico-nrf24-driver/blob/main/lib/nrf24l01/nrf24_driver.c.

[4] andyrids. *Pico NRF24 driver.* URL: https://github.com/andyrids/pico-nrf24-driver.

[5] andyrids. *README.md.* URL: https://github.com/andyrids/pico-nrf24-driver/blob/main/README.md.

[6] Arduino. Accessed: 25-05-2026. URL: https://store.arduino.cc/collections/nano-family/products/arduino-nano-33-iot.

[7] Arduino. Accessed: 25-05-2026. URL: https://store.arduino.cc/collections/nano-family/products/arduino-nano-rp2040-connect.

[8] Arduino. Accessed: 25-05-2026. URL: https://store.arduino.cc/collections/nano-family/products/nano-esp32.

[9] *Arduino - Home.* URL: https://www.arduino.cc/.

[10] Jean-Philippe Aumasson et al. "Blake2". In: *The Hash Function BLAKE*. Springer, 2014, pp. 165–183.

[11] The Amazing Adventures of Ben Franklin. *The Pennsylvanian Fireplace.* Accessed: 21-05-2026. URL: https://content.time.com/time/specials/packages/article/0,28804,2012113_2011976_2011968,00.html.

[12] Daniel J Bernstein et al. "ChaCha, a variant of Salsa20". In: *Workshop record of SASC*. Vol. 8. 1. Lausanne, Switzerland. 2008, pp. 3–5.

[13] John Shaw Billings. *Ventilation and heating.* Engineering record, 1893.

[14] Matthias Braubach et al. "Mortality associated with exposure to carbon monoxide in WHO E uropean M ember S tates". In: *Indoor air* 23.2 (2013), pp. 115–125.

[15] carlsondev. *NULL config returns error in $nrf_driver_initialise$.* URL: https://github.com/andyrids/pico-nrf24-driver/issues/2.

[16] *Ceedling.* URL: https://www.throwtheswitch.org/ceedling.

[17] Lada Hensen Centnerová. "On the history of indoor environment and it's relation to health and wellbeing". In: *REHVA Journal* (2018).

[18] R Cushen et al. "An unusual incident: carbon monoxide poisoning risk in 540 homes due to faulty wood burner installations". In: *Public health* 173 (2019), pp. 17–20.

[19] Joan Daemen and Vincent Rijmen. "AES proposal: Rijndael". In: (1999).

[20] Kyzer R Davis, Brad Peabody, and P Leach. "Universally unique identifiers (UUIDs)". In: *IETF Datatracker* (2024).

[21] Dfaguimba. Accessed: 21-05-2026. 2025. URL: https://www.facebook.com/61554977615841/photos/a-detailed-architectural-cross-section-of-a-hypocaust-system-an-ancient-roman-me/1222528689381659 20/.

[22] Django. *Applications.* URL: https://docs.djangoproject.com/en/6.0/ref/applications/.

[23] *Django.* URL: https://www.djangoproject.com/.

[24] Arthur Conan Doyle. *The Adventure of the Copper Beeches.* The Strand Magazine, 1892.

[25] U.S. Department of Energy. *History of Air Conditioning.* URL: https://www.energy.gov/articles/history-air-conditioning.

[26] *Espressif.* URL: https://www.espressif.com/.

[27] Eurostat. *Energy consumption in households.* URL: https://ec.europa.eu/eurostat/statistics-explained/SEPDF/cache/58200.pdf.

[28] Garrett G. Fagan. "Sergius Orata: Inventor of the Hypocaust?" In: *Phoenix* (1996).

[29] *FastAPI.* URL: https://fastapi.tiangolo.com/.

[30] HermannSW. *"Performance C vs MicroPython"' thread in "Raspberry Pi Pico > General".* URL: https://forums.raspberrypi.com/viewtopic.php?t=303456.

[31] hippy. *My Pico Benchmarking.* URL: https://forums.raspberrypi.com/viewtopic.php?t=303458.

[32] U.S. Department of State Office of the Historian. *Oil Embargo, 1973–1974.* URL: https://history.state.gov/milestones/1969-1976/oil-embargo.

[33] *Hydrogen chemical profile.* URL: https://pubchem.ncbi.nlm.nih.gov/compound/hydrogen.

[34] IEEE. *IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications.* URL: https://standards.ieee.org/ieee/802.11/1163/.

[35] John E Janssen. "The history of ventilation and temperature control: The first century of air conditioning". In: *Ashrae Journal* 41.10 (1999), pp. 47–52.

[36] Jeremy Johnson. "New directions in cryptography". In: *IEEE Trans. Inf. Theory* 22.6 (1976), pp. 644–654.

[37] Sylvain Kerkour. Accessed: 21-05-2026. 2022. URL: https://kerkour.com/end-to-end-encryption-key-exchange-cryptography-rust.

[38] Neal Koblitz. "Elliptic curve cryptosystems". In: *Mathematics of computation* 48.177 (1987), pp. 203–209.

[39] Eric Lavigne et al. “Mortality and hospital admission rates for unintentional nonfire-related carbon monoxide poisoning across Canada: a trend analysis”. In: *CMAJ open* 3.2 (2015), E223–E230.

[40] Hannes Lehar. “The Roman Hypocaust Heating System. Calculations and thoughts about construction, performance and function”. In: *17th International Conference on Cultural Heritage and New Technologies*. 2012.

[41] Wolfgang Gerhard Locher. “Max von Pettenkofer (1818–1901) as a pioneer of modern hygiene and preventive medicine”. In: *Environmental health and preventive medicine* 12.6 (2007), pp. 238–245.

[42] Leroy Merlin. *Caldera de condensación de gas natural MANAUT myto plus 25kw*. Accessed: 21-05-2026. URL: https://www.leroymerlin.es/productos/caldera-de-condensacion-de-gas-natural-manaut-myto-plus-25kw-97066125.html.

[43] *Methane chemical profile*. URL: https://pubchem.ncbi.nlm.nih.gov/compound/Methane.

[44] *Mongoose*. URL: https://mongoose.ws/.

[45] *Monocypher*. URL: https://monocypher.org/.

[46] Tony Mormino. Accessed: 21-05-2026. 2025. URL: https://www.facebook.com/100009129420767/posts/if-you-had-this-packaged-ac-unit-in-your-home-you-are-officially-old-in-1933-the/4121578088156465/.

[47] *Neon*. URL: https://neon.com/.

[48] Florence Nightingale. *Notes on nursing: What it is, and what it is not*. Lippincott Williams & Wilkins, 1992.

[49] *Open Fires, Chimneys and Flues*. URL: https://historicengland.org.uk/advice/your-home/improvement/open-fires-chimneys-and-flues/.

[50] Stack Overflow. *Most popular technologies*. URL: https://survey.stackoverflow.co/2025/technology/.

[51] *Playwright*. URL: https://playwright.dev/.

[52] *Plotly*. URL: https://plotly.com/.

[53] *PostgreSQL*. URL: https://www.postgresql.org/.

[54] *Propane chemical profile*. URL: https://pubchem.ncbi.nlm.nih.gov/compound/propane.

[55] *Psycopg*. URL: https://www.psycopg.org/.

[56] *Python*. URL: https://www.python.org/.

[57] Herminia Buchelli Ramirez et al. “Niveles elevados de carboxihemoglobina: fuentes de exposición a monóxido de carbono”. In: *Archivos de bronconeumología* 50.11 (2014), pp. 465–468.

[58] *Raspberry Pi*. URL: https://www.raspberrypi.com/.

[59] Eric Rescorla. *The transport layer security (TLS) protocol version 1.3*. Tech. rep. 2018.

[60] RoiCorporation. *tfg-api*. URL: https://github.com/RoiCorporation/tfg-api.

[61] RoiCorporation. *tfg-app*. URL: https://github.com/RoiCorporation/tfg-app.

[62] RoiCorporation. *tfg-firmware*. URL: https://github.com/RoiCorporation/tfg-firmware.

[63] Satkit. Accessed: 24-05-2026. URL: https://satkit.com/sensor-bme680-arduino-temperatura-humedad-presion-gas.

[64] Nordic Semiconductor. *nRF24L01 Product Specification V2.0*. URL: https://docs.nordicsemi.com/bundle/nRF24L01P_PS_v1.0/resource/nRF24L01P_PS_v1.0.pdf.

[65] Nordic Semiconductor. *Power Profiler Kit II*. URL: https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2.

[66] Semtech. *LoRa (PHY)*. URL: https://www.semtech.com/lora/what-is-lora.

[67] Semtech. *SX1278*. URL: https://www.semtech.com/products/wireless-rf/lora-connect/sx1278.

[68] Mikyong Shin et al. “Morbidity and mortality of unintentional carbon monoxide poisoning: United States 2005 to 2018”. In: *Annals of emergency medicine* 81.3 (2023), pp. 309–317.

[69] RV Simha. “Willis H carrier: Father of air conditioning”. In: *Resonance* 17.2 (2012), pp. 117–138.

[70] ESPRESSIF Official Store. Accessed: 25-05-2026. URL: https://es.aliexpress.com/item/1005004439243833.html.

[71] *The History of Using Natural Gas for Heating Homes*. URL: https://www.progasllc.com/the-history-of-using-natural-gas-for-heating-homes/.

[72] tiendatec. Accessed: 25-05-2026. URL: https://www.tiendatec.es/raspberry-pi-pico/2423-1396-raspberry-pi-pico-2-w.html#/249-version-sin_pines.

[73] tiendatec. Accessed: 24-05-2026. URL: https://www.tiendatec.es/maker-zone/modulos/2580-modulo-sensor-de-intensidad-de-luz.html.

[74] tiendatec. Accessed: 24-05-2026. URL: https://www.tiendatec.es/maker-zone/modulos/1565-sensor-mq-9-detector-gas-monoxido-carbono-8472496020886.html.

[75] *Understanding GitHub Actions*. URL: https://docs.github.com/en/actions/get-started/understand-github-actions.

[76] *Vercel*. URL: https://vercel.com/.

[77] *Vite*. URL: https://vite.dev/.

[78] Lindell K Weaver. “Carbon monoxide poisoning”. In: *New England Journal of Medicine* 360.12 (2009), pp. 1217–1225.

[79] Pablo Young et al. “Florence Nightingale (1820-1910), a 101 años de su fallecimiento”. In: *Revista médica de Chile* 139.6 (2011), pp. 807–813.

[80] Victor Zarnowitz and Geoffrey H Moore. “The recession and recovery of 1973-1976”. In: *Explorations in Economic Research, Volume 4, number 4*. NBER, 1977, pp. 1–87.

Appendix A

Comparison of Indoor Air Monitoring Alternatives

URL	Device name	Battery Life (hours)	Price	Temperature	Humidity	Light intensity	Air pressure	VOCs - AQI	Carbon monoxide	Methane	Propane	Hydrogen gas	Amount of relevant parameters monitored
https://www.amazon.com/Professional-Temperatur	9-in-1 Professional Indoor Air Quality Monitor Indoo	16	81,22 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://www.sisco.com/home-air-quality-monitor	Home Air Quality Monitor, PM2.5/CO2/TVOC/Tempe		270,53 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://sperdirect.com/es/products/indoor-air-quali	Indoor Air Quality Monitor		84,77 €	☑	☑	☐	☐	☐	☐	☐	☐	☐	2
https://sperdirect.com/es/products/multi-function-c	Multi-Function Indoor Air Quality Monitor with Optio		167,41 €	☑	☑	☐	☑	☑	☐	☐	☐	☐	4
https://www.temtop.co.uk/es/products/temtop-s1-c	Temtop S1-up Indoor Air Quality Meter Temperature	1440	28,95 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://products.shawcity.co.uk/products/honeywel	Honeywell HTRAM Indoor Air Quality Monitor		158,43 €	☑	☑	☐	☐	☐	☐	☐	☐	☐	2
https://www.iqair.com/products/air-quality-monitor	AirVisual Pro Indoor Monitor	4	319,00 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://www.carrierathome.com/products/air-monit	Carrier Air Quality Monitor		102,76 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://www.acurite.com/products/8-in-1-smart-ind	8-in-1 Smart Indoor Air Quality Monitor with Wi-Fi		85,62 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://www.test-meter.co.uk/tpi-1010a-co-and-co2	TPI 1010A CO and CO2 Indoor Air Quality Monitor	40	880,30 €	☑	☑	☐	☐	☐	☑	☐	☐	☐	3
https://ionmax.com.au/products/ionmax-q10-air-qu	Ionmax Q10 Air Quality Monitor		127,59 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://metone.com/products/aerocet-532-handhel	Aerocet 532 Handheld Particle Monitor	8		☑	☑	☐	☐	☐	☐	☐	☐	☐	2
https://www.jaycar.co.nz/eve-room-indoor-air-qualit	Eve Room - Indoor Air Quality Monitor (Thread)		187,53 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://huma-i.eu/en/indoor-air-quality-monitors/2-h	Huma-i smart (HI-300)	1	179,00 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://aranet.com/en/home/products/aranet4-hom	CO2 Monitor for Indoor Air Quality at Home and Wo	35040	215,38 €	☑	☑	☐	☑	☐	☐	☐	☐	☐	3
https://www.amazon.es/dp/B0CZ85RT6C	Qingping Air Quality Monitor Gen 2, Air Quality Moni		159,99 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://www2.purpleair.com/products/purpleair-zen	PurpleAir Zen - Air Quality Monitor	0	256,04 €	☑	☑	☐	☑	☑	☐	☐	☐	☐	4
https://www.walmart.com/ip/AIRKNIGHT-9-In-1-Ind	AIRKNIGHT 9-in-1 Indoor Air Quality Monitor, Portab		82,86 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://ruuvi.com/air/	Ruuvi Air – Indoor Air Quality Monitor	0	149,00 €	☑	☑	☐	☑	☑	☐	☐	☐	☐	4
https://www.airthings.com/en/view-plus	View Plus	17520	299,00 €	☑	☑	☐	☑	☑	☐	☐	☐	☐	4
https://www.airgradient.com/indoor/	Indoor Air Quality Monitor	0	118,17 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://www.amazon.es/amazon-smart-air-quality-n	Amazon Smart Air Quality Monitor - Amazon Smart		79,99 €	☑	☑	☐	☐	☑	☑	☐	☐	☐	4
https://sensibo.com/	Sensibo Air PRO		411,00 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://www.labmate.com/indoor-air-quality-monito	Indoor Air Quality Monitor LMAQM-A100		6.184,84 €	☑	☑	☐	☐	☑	☐	☐	☐	☐	3
https://www.ikea.com/us/en/p/alpstuga-air-quality-	ALPSTUGA	0	25,68 €	☑	☑	☐	☐	☐	☐	☐	☐	☐	2
https://www.amazon.com/Qingping-Monitor-Compa	Qingping Air Monitor Lite, Apple HomeKit Compatib	7	64,98 €	☑	☑	☐	☐	☐	☐	☐	☐	☐	2
https://shop.getuhoo.com/products/uhoo-sense	uHoo Sense	0	173,92 €	☑	☑	☐	☑	☑	☑	☐	☐	☐	5

Figure A.1: Comparison of 27 commercially available indoor air monitoring systems.

URL	Device name	Battery Life (hours)	Price	Amount of relevant parameters monitored
https://www.amazon.com/Professional-Temperature-For	9-in-1 Professional Indoor Air Quality Monitor Indoor Port	16	81.22 €	3
https://www.sisco.com/home-air-quality-monitor	Home Air Quality Monitor, PM2.5/CO2/TVOC/Temperatur		270.53 €	3
https://sperdirect.com/es/products/indoor-air-quality-mo	Indoor Air Quality Monitor		84.77 €	2
https://sperdirect.com/es/products/multi-function-deskt	Multi-Function Indoor Air Quality Monitor with Optional W		167.41 €	4
https://www.temtop.co.uk/es/products/temtop-s1-calida	Temtop S1-up Indoor Air Quality Meter Temperature & Hu	1440	28.95 €	3
https://products.shawcity.co.uk/products/honeywell-tran	Honeywell HTRAM Indoor Air Quality Monitor		158.43 €	2
https://www.iqair.com/products/air-quality-monitors/airv	AirVisual Pro Indoor Monitor	4	319.00 €	3
https://www.carrierathome.com/products/air-monitor	Carrier Air Quality Monitor		102.76 €	3
https://www.acurite.com/products/8-in-1-smart-indoor-a	8-in-1 Smart Indoor Air Quality Monitor with Wi-Fi		85.62 €	3
https://www.test-meter.co.uk/tpi-1010a-co-and-co2-indo	TPI 1010A CO and CO2 Indoor Air Quality Monitor	40	880.30 €	3
https://ionmax.com.au/products/ionmax-q10-air-quality-	Ionmax Q10 Air Quality Monitor		127.59 €	3
https://metone.com/products/aerocet-532-handheld-par	Aerocet 532 Handheld Particle Monitor	8		2
https://www.jaycar.co.nz/eve-room-indoor-air-quality-mo	Eve Room - Indoor Air Quality Monitor (Thread)		187.53 €	3
https://huma-i.eu/en/indoor-air-quality-monitors/2-huma	Huma-i smart (HI-300)	1	179.00 €	3
https://aranet.com/en/home/products/aranet4-home	CO2 Monitor for Indoor Air Quality at Home and Work	35040	215.38 €	3
https://www.amazon.es/dp/B0CZ85RT6C	Qingping Air Quality Monitor Gen 2, Air Quality Monitor In		159.99 €	3
https://www2.purpleair.com/products/purpleair-zen	PurpleAir Zen - Air Quality Monitor	0	256.04 €	4
https://www.walmart.com/ip/AIRKNIGHT-9-In-1-Indoor-A	AIRKNIGHT 9-in-1 Indoor Air Quality Monitor, Portable Ind		82.86 €	3
https://ruuvi.com/air/	Ruuvi Air – Indoor Air Quality Monitor	0	149.00 €	4
https://www.airthings.com/en/view-plus	View Plus	17520	299.00 €	4
https://www.airgradient.com/indoor/	Indoor Air Quality Monitor	0	118.17 €	3
https://www.amazon.es/amazon-smart-air-quality-monit	Amazon Smart Air Quality Monitor - Amazon Smart Air Qu		79.99 €	4
https://sensibo.com/	Sensibo Air PRO		411.00 €	3
https://www.labmate.com/indoor-air-quality-monitor/lma	Indoor Air Quality Monitor LMAQM-A100		6,184.84 €	3
https://www.ikea.com/us/en/p/alpstuga-air-quality-sens	ALPSTUGA	0	25.68 €	2
https://www.amazon.com/Qingping-Monitor-Compatible-	Qingping Air Monitor Lite, Apple HomeKit Compatible Wi-	7	64.98 €	2
https://shop.getuhoo.com/products/uhoo-sense	uHoo Sense	0	173.92 €	5

Figure A.2: A compact version of the comparison shown in Figure A.1.[1]

[1]For products whose prices were not originally listed in Euros, conversions from the original currencies to Euros were calculated using Google Chrome's currency conversion tool on May 14, 15, and 16, 2026.

Appendix B

Segments of a wireless station power profile

B.1 Low power mode (“hibernation”)

Figure B.1: Low power segment of the wireless station power profile (segment 1 of 2).

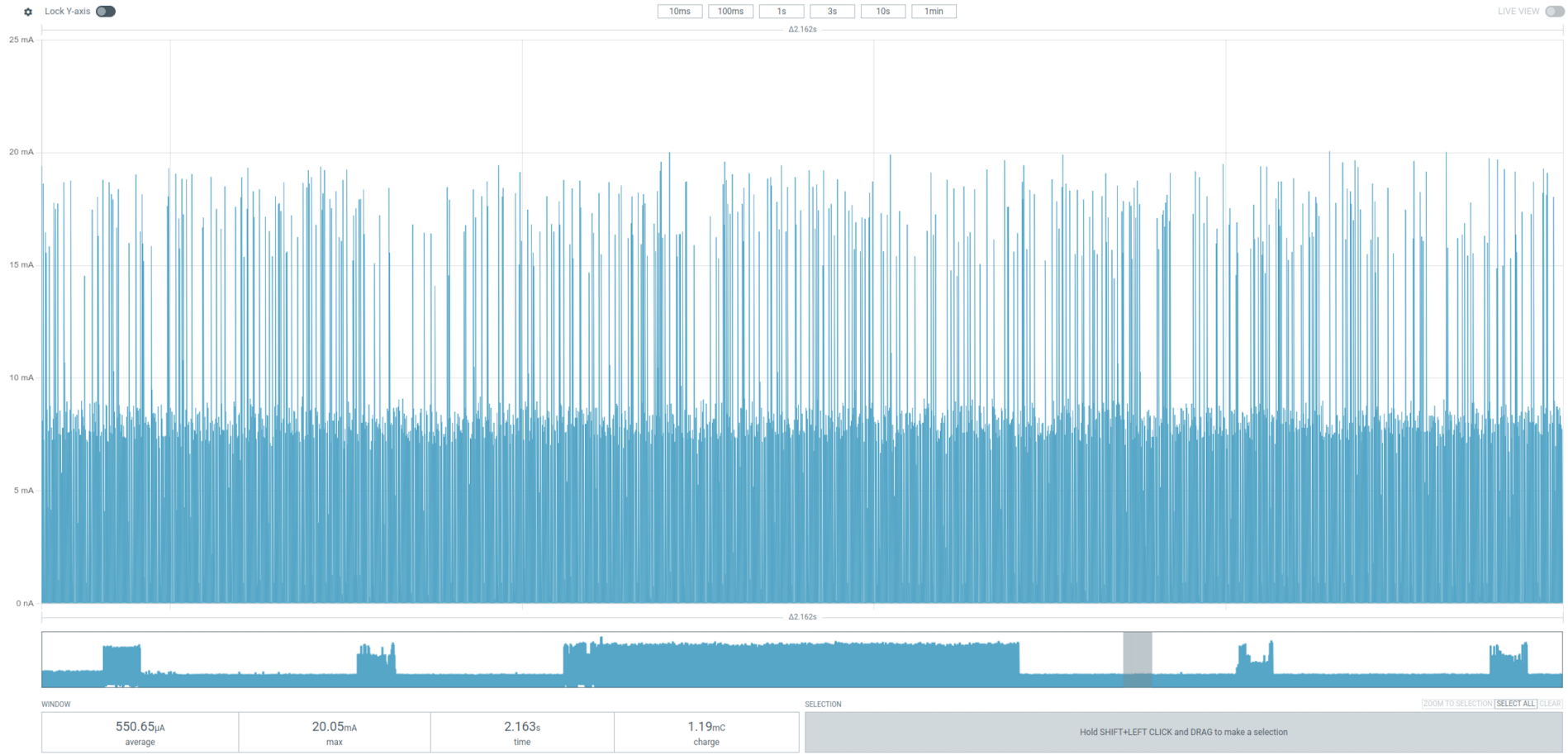


Figure B.2: Low power segment of the wireless station power profile (segment 2 of 2).

B.2 Normal operation (station actively working)

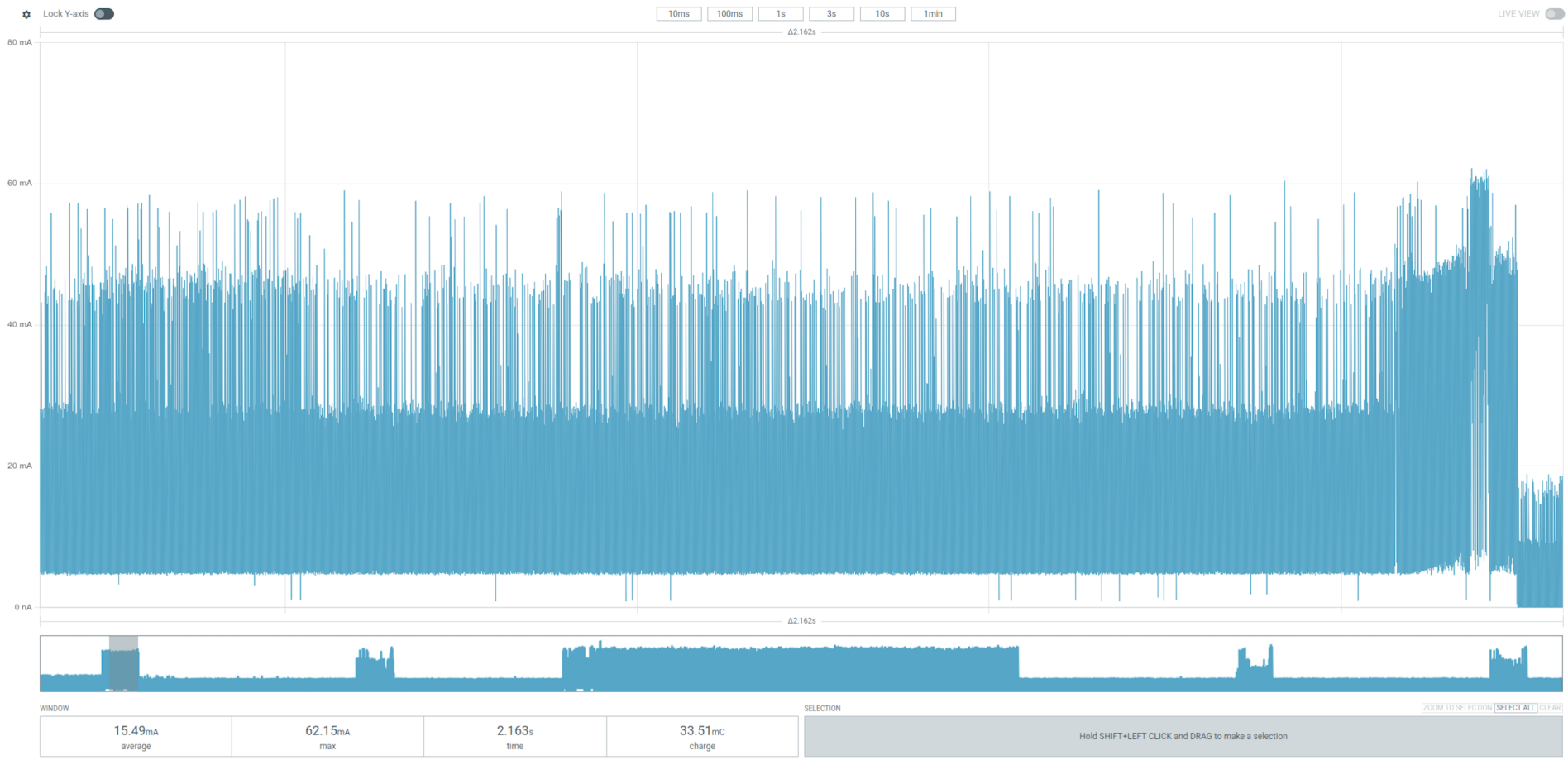


Figure B.3: Normal operation (active) segment of the wireless station power profile (segment 1 of 3).

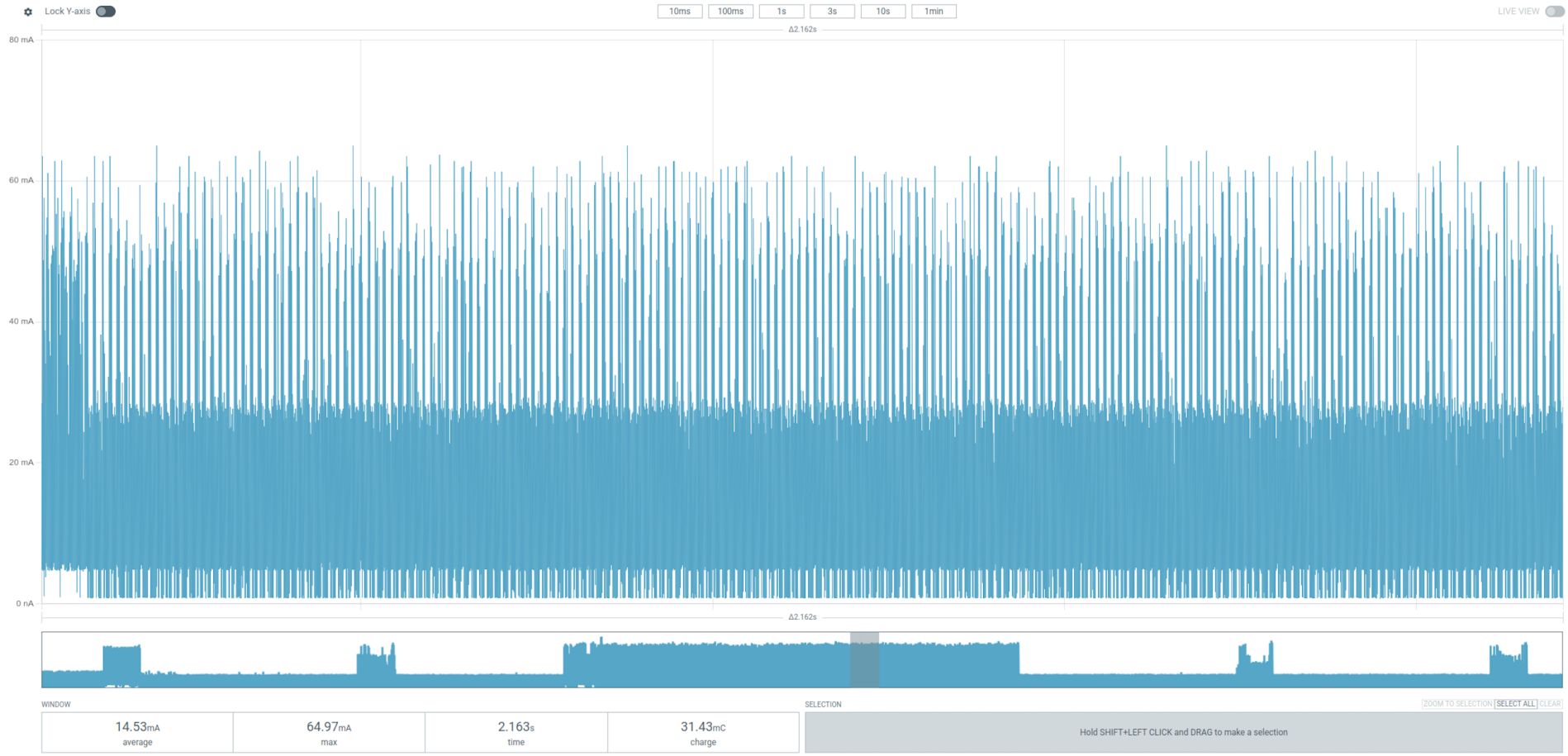


Figure B.4: Normal operation (active) segment of the wireless station power profile (segment 2 of 3).

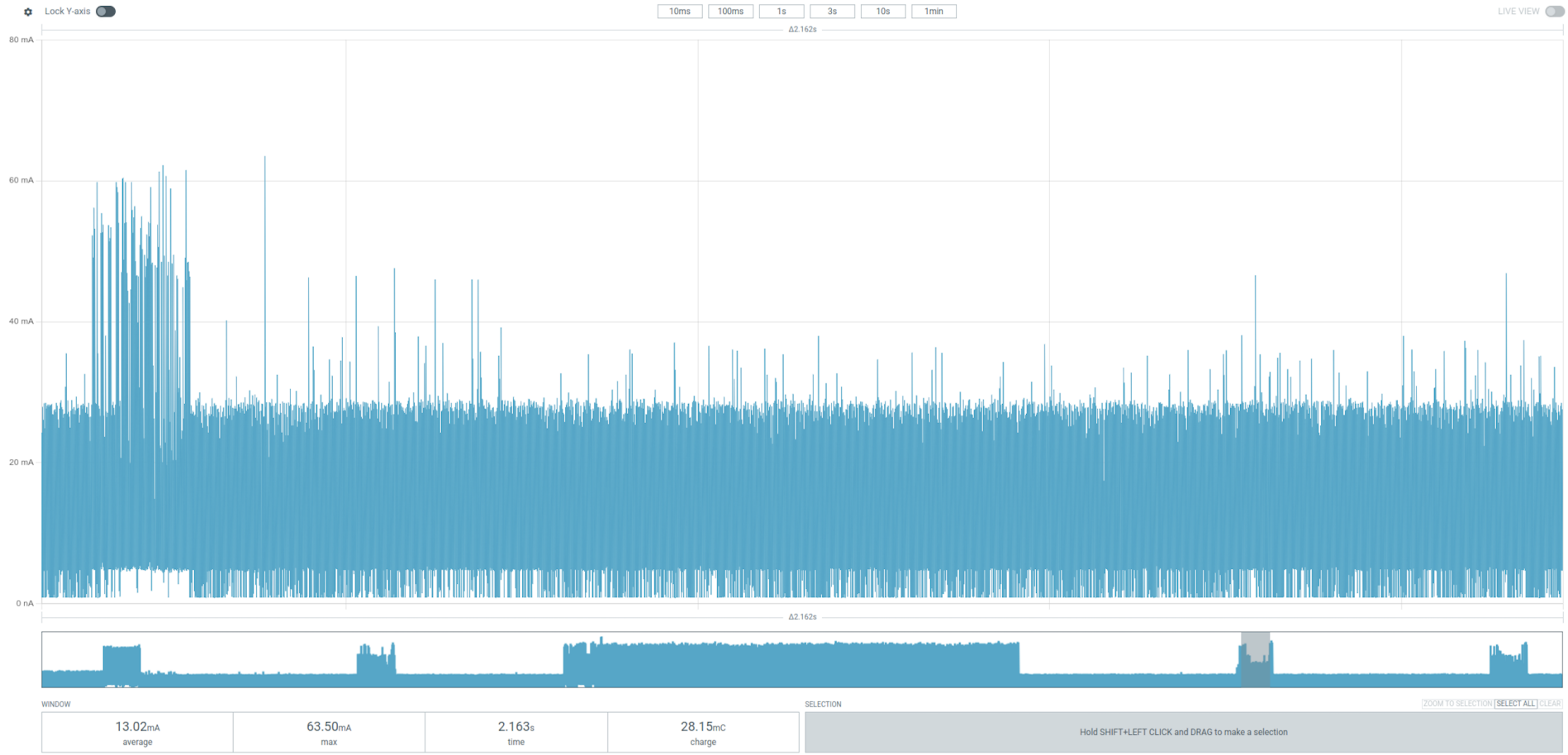


Figure B.5: Normal operation (active) segment of the wireless station power profile (segment 3 of 3).