PM

Low Cost IoT Monitoring Solution for Increased Student Awareness on Campus

MASTER'S DEGREE PROJECT

Leonel Dinarte Camacho Freitas

MASTER IN INFORMATICS ENGINEERING

UNIVERSIDADE da MADEIRA

A Nossa Universidade

www.uma.pt

May | 2021

Low Cost IoT Monitoring Solution for Increased Student Awareness on Campus

MASTER'S DEGREE PROJECT

Leonel Dinarte Camacho Freitas

MASTER IN INFORMATICS ENGINEERING

ORIENTATION

Karolina Baras

CO-ORIENTATION

Lina Maria Pestana Leão de Brito

I want to start, and acknowledge, my family, friends, former colleagues, my thesis supervisors Karolina and Lina, and everyone that has shaped my life positively.

It was a very tough year to the world, and for me personally, one of the worst times in recent years. Writing this thesis and developing this project in very adverse conditions meant forfeiting a lot of my life, both emotionally and economically. I had to return to my home island and pause my professional career. This was a tough challenge, but I know I would eventually regret not finishing my master's degree.

I want to appreciate and dedicate all my work entirely to my closest family, especially to my mom who insisted me to finish my master's degree. Without her support and motivation, I would not be able to complete this path.

The end of this journey, for me, means more than just an academic achievement, it's the end of an era, and I feel my personal path of redemption is complete. I only started my master's to prove myself that I was able to succeed after falling miserably during my first years of bachelor. I did not want to study or be at university at that time, in fact I had no direction. In the first two years I passed two classes, which can be seen as a laughable "miracle", having in consideration that I barely attended any class or studied anything at all, still I was able to eventually pick myself up in the following years. Now I am thankful and pleased to say that things are way different in a positive way and those murky times are long gone.

My journey in the last 4 years shaped profoundly how I see the world, having lived, studied or worked in more regions than some people have had jobs, a gift for which I am grateful. I hope to whomever reads this introduction might gather some inspiration from it, and muster the perseverance to reach their goals, and that failure is part of the process as long as we learn from our mistakes and forgive ourselves.

To all, much love and always stay grateful and humble.

Leonel.

“Experience is merely the name men gave to their mistakes.”

Oscar Wilde, The Picture of Dorian Gray

Resumo

O preço do equipamento eletrónico tem descido acentuadamente nas últimas décadas. Este decréscimo favoreceu um crescimento exponencial de todo o tipo de dispositivos eletrónicos, a ubiquidade destes dispositivos torna-os quase indissociáveis dos seus utilizadores. Uma das áreas a beneficiar e igualmente a potenciar esse crescimento é a Internet das Coisas, IoT, ou seja, a possibilidade de uma ligação constante e de partilha de dados entre os mais diversos dispositivos. Em paralelo com o uso de soluções de índole comercial, tem surgido dispositivos de custo reduzido como o Arduino, que permitem criar pequenos projetos IoT de forma económica. Para além destes, outros produtos tais como as boards da Espressif Esp8266 e ESP32 já com ligação Wi-Fi incluída tem ganho notoriedade em anos recentes. O objetivo deste projeto foi precisamente o desenvolvimento de um sistema de recolha de dados de baixo custo e autónomo, que tenha utilidade para os seus utilizadores, especialmente num contexto académico utilizando o ESP32 da Espressif como base. Este sistema visa recolher dados ambientais tais como temperatura, humidade, nível de ruído, e através do uso dos sinais Wi-Fi emitidos pelos dispositivos móveis, tentar estabelecer a estimativa da ocupação de uma determinada localização. Esta informação recolhida deverá ser disponibilizada à comunidade académica tornando possível, por exemplo, aos alunos selecionar a melhor sala de estudo baseando-se na sua ocupação, ou nível de ruido. Os principais componentes destes projetos são os ESP32s e os respetivos sensores, denominados de SensingBox, e a componente de software e infraestrutura associada para processar os dados recolhidos, interagir com as SensingBox e permitir a visualização dos dados recolhidos.

Além deste objetivo primário é importante fazer uma análise exploratória das potencialidades desta plataforma em projetos de baixo custo assim como as metodologias de desenvolvimento para a mesma. De certa forma este projeto visa em parte servir de guia para quem esteja a desenvolver um projeto de natureza similar. O sistema desenvolvido foi testado em dois contextos, um para aprimorar o sistema num contexto doméstico, e outro no contexto para o qual havia sido idealizado, ou seja, no Campus da Universidade da Madeira.

Palavras Chave: IoT, Microcontrolador ESP32 da Espressif, Sistemas de Monitorização de baixo custo, Captura de Pacotes de Redes Wi-Fi;FreeRTOS

Abstract

In the last decades there has been a steep decrease in the price of electronics. This decrease has favored an exponential increase of all types of consumer electronics. The ubiquity of these devices makes them almost inseparable from their users. One area benefiting and at the same time further increasing this ubiquity is the Internet of Things, IoT, in other words the possibility of having a constant connection and data sharing amongst the most diverse devices. In parallel with the development of IoT solutions designed for Industrial use, some other low-cost devices such as the Arduino have appeared, which enable the development of small IoT projects economically. Besides the Arduino other products such as the ESP8266 and ESP32 boards by Espressif have been gaining popularity in recent years. The aim of this project is the development of a low cost, autonomous monitoring solution that is of use for the academic community. This system should collect environmental data such as temperature, humidity, noise pollution and make an estimation of the occupancy of a certain zone through the ubiquity of mobile devices by capturing their Wi-Fi request probes. The information collected will be made available for the academic community, giving the possibility of students to select the best study room based on the occupancy or noise level of the different rooms. The main constituents of the project are the ESP32s with the associated sensors named Sensing Boxes, and the necessary software component and infrastructure to process the data collected and provide it to interested parties.

In addition to this main objective, it is important to understand and document the potentialities of the ESP32 in low-cost projects as its development methodologies available. In a sense provide a guideline of the most important topics about this board and some sensors for anyone developing a project of similar nature. The system was tested in two contexts, one of domestic nature, and in order to fine tune and debug the system, the other in the context to which it was originally idealized, the campus of University of Madeira.

Keywords: IoT: ESP32 Microcontroller from Espressif; Low Cost monitoring System, Wi-Fi packet capture; FreeRTOS

Index

Figure List

Table List

Acronyms

ACID - Atomicity, Consistency, Isolation, Durability

ADC – Analog Digital Converter

API - Application Programming Interface

ATT - Attribute Protocol

AWS - Amazon Webservices

BLE - Bluetooth Low Energy

CD - Continuous Delivery

CI - Continuous Integration

CLI – Command Line Interface

COAP - Constrained Application Protocol

CORS – Cross-Origin Resource Sharing

DMA – Direct Memory Access

EC2 - Elastic Compute Cloud

GATT - Generic Attribute Profile

HTTP - Hypertext Transfer Protocol

I2S - Inter-IC Sound

IDF - Integrated Development Framework

IOT - Internet of Things

IR - Infrared

ISR - Interrupt Service Routine

MQTT - MQ Telemetry Transport

NFC - Near Field Communication

NVS- Non-Volatile Storage

NPM- Node Package Manager

OTA - Over the Air

RDS - Relational Database Service

REST - Representational State Transfer

RFID -radio frequency identification

RTC - Real Time Clock

SD - Secure Digital

SDK- Software Development Kit

SPA - Single Page Application

SRS - Software Requirements Specification

UART - Universal Asynchronous Receiver/Transmitter

ULP - Ultra Low Power

ZB - Zettabytes

1 INTRODUCTION.

It is estimated that by 2025 over 41.6 billion devices or things will be connected to the Internet.[1]. These devices generate a significant amount of information, with current projections showing that in 2025, there will be 5 IoT for every human, generating 79.4 (ZB) of information.[2]. While more information is available, is necessary to take advantage of this information in useful ways in areas as diverse and distinct as agriculture or healthcare. This is one of the objectives of IoT, generate value from information and apply it to different industries. While there are several examples of IoT projects applied in different industries, a significant number of projects are developed deprived of commercial intent using low-cost devices and sensors, this way providing an entry door to the IoT world.

One annoyance faced by some students is the selection of the appropriate room to study. This project proposes a low-cost monitoring system that will monitor environmental conditions, and occupancy of different rooms throughout campus, providing this information to students. In recent years, the growth of IoT has contributed to an increased number of projects and information online, especially related to low-cost devices. While this information is certainly useful, many of the projects or examples are of limited scope or applied in very basic domestic scenarios. This is another important aspect of this project, serve as a guideline in terms of development aspects to have in consideration when developing IoT solutions, especially when using the ESP32 or another microcontroller from this family.

This project encompasses different technologies and was also developed using common practices in industry today, such as the use of containers, testing, documentation, while these concepts are primarily applied in a professional and more collaborative contexts, than a master thesis project, they still hold their value even in an academic context.

1.1 MOTIVATION

Studying is an integral part of any student's life. The location in which studying takes place can be a key to a better academic performance.

According to a survey made to university students in 2015, the choice of place to study at University of Madeira (UMa) Campus, is conditioned mostly by the high occupancy of study rooms, as the related level of noise. The behavior of some students is also a problem. The survey had 12 questions and obtained a total of 83 responses with most participations being engineering students.

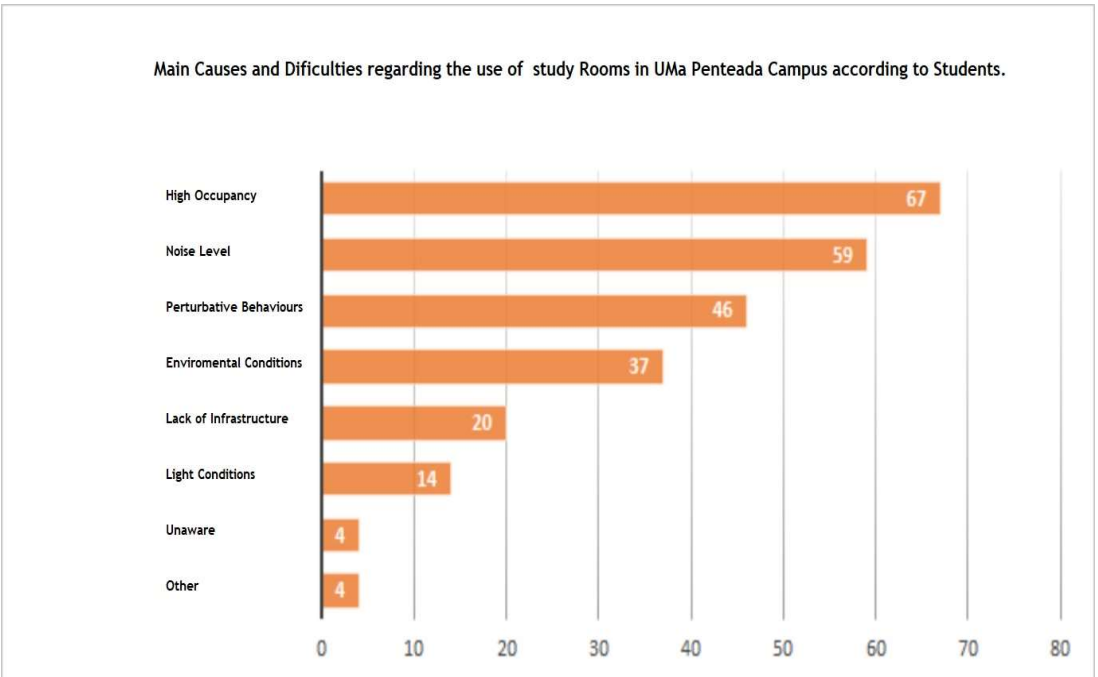


Figure 1 Main Causes and Difficulties while studying at Uma

Based on this survey, the most important parameters to measure are ambient noise and occupancy level of the different rooms. By measuring these parameters with a "Sensing Box" and providing this information to the academic community, this may increase student's awareness for non-optimal study conditions or better distribute them across different rooms.

To reach this goal in a cost-efficient manner, it is important to select the right devices and sensors, i.e., the right "things". Sometimes this can be a difficult undertaking, especially for developers with limited electronic knowledge, so providing a review of the current development boards for IoT and their potentialities can be a good starting point.

1.2 OBJECTIVES

The objectives of this project are based on the previous motivation and can be defined as the following:

- Develop a low-cost system that can monitor environmental conditions in different university locations and provide that information to students.
- Assess feasibility of Wi-Fi as a means of counting people.
- Create a system that should be applicable to other contexts that require a low-cost data collection device.
- Explore some of the most popular IoT development boards on the market such as the ESP32 by Espressif and key concepts related to IoT projects.

1.3 STRUCTURE

This thesis follows the following structure:

- Chapter 1: Introduction, and project objectives are presented.
- Chapter 2: Corresponds to the literature review focusing on smart campus initiatives using IoT concepts.
- Chapter 3: Focuses on the technological review and design consideration for the project with special regards to the communication protocols, microcontroller boards, sensors, and battery options.
- Chapter 4: Refers to system implementation, from the low-level implementation in the ESP32 board, to the deployment of the overall system.
- Chapter 5: The tests, and results are presented and analyzed.
- Chapter 6: Presents the conclusion, limitations, and future work.

2 LITERATURE REVIEW

In this chapter, the current trends of IoT uses in universities are explored. A significative number of initiatives aimed at integrating IoT in university's campus are currently taking place. On one hand, "smart campus", tend to provide contextual awareness information to their community by using location tracking and services information. On the other hand, some projects try to leverage IoT concepts into eLearning applications. In the following sections some examples of these two approaches are provided, starting with smart campus initiatives, and moving onto e-learning and sensor-based learning examples.

2.1 SMART CAMPUS

Smart Campus or connected campus can be seen as a subset of smart cities. Smart campus make use of IoT to automatically control and monitor facilities and provide a higher quality service to students and staff. This leads to increased efficiency and responsiveness of the campus, thus providing better decision making, space utilization, and students experience.[3].

Muhamad et al. [4], conducted a survey of published research to identify the general state of smart campus initiatives. The research questions proposed can be found bellow.

ID	Research Question	Motivation
		smart campus
RQ2	What kinds of technology support smart campus implementation?	Identify the suitable technology used to develop smart campus and its specific usage
RQ3	Is there the standard model for the smart campus?	Identify the generic model that could be adopted for smart campus development
RQ4	What are the applications implemented in smart campus?	Identify the variety of smart campus application and research challenges on it
RQ5	What are smart campus contributions?	Identify the contribution of smart campus that motivates university management to develop it

Figure 2 Smary Campus Initiatives [4]

After filtering and narrowing down the search to 29 different research publications, a distinction can be established between Smart Campus and Digital Campus. Smart Campus can be seen as a campus that features connectivity and applies IoT concepts to provide useful information to its users. A digital campus can be seen as an online campus or platform for eLearning. In terms of technology, the most referenced technologies in smart campus were IoT and its associated technologies such RFID, NFC, general sensors (motion, temperature …) and cloud computing.

2.1.1 DEAKING UNIVERSITY

At Deakin University, several systems were deployed and are used by part of the academic community. Students can see the occupancy of the library in a heat map at the library´s entrance. Additionally, it is possible to request assistance from the student´s current position and a librarian will go to their encounter. For the location-based tracking, Cisco Mobility Service Engine is used. This system can track the physical location of connected devices.[5] Another employed solution at Deakin Universitiy is a virtual assistant named Deankie Genie (Figure 3) that can help students in their daily academic tasks such as navigation around campus, or submitting a report. Deankie Genie makes uses of IBM´s Watson [6], a cloud based AI.

Figure 3 Deankie Genie Interface

2.1.2 MICROSOFT PROJECT GARÇON.

Project Garçon developed by Microsoft and implemented in one of Microsoft campus uses IoT and Artificial Intelligence to provide a novel form of interaction to its users, in opposition to a traditional web-based system. It comprises a voice assistance that is placed in a wall near a particular place of interest. In the demo, it was placed near a bathroom, and the main goal was to provide an easier way of reporting problems or other occurrences, instead of a traditional approach of logging the data in a web or mobile app and opening a ticket.[7]

2.1.3 IN-ROOM FEEDBACK

A group of researchers from Warsaw University of Technology [8] developed a system to explore a hint based feedback system, where the system hints actions that the user might want to perform. This type of technique was defined as soft actuation contrasting with proactive techniques where the system acts on the user behalf. With soft actuation

the user is hinted for actions that he could perform. Using a small box with a 3x3 LED display grid, where the LED pattern corresponds to an action that the user might want to perform. The system was tested with 14 participants. Most of the participants liked the concept of soft actuation, although the system was considered as too subtle with many hints going unnoticed.

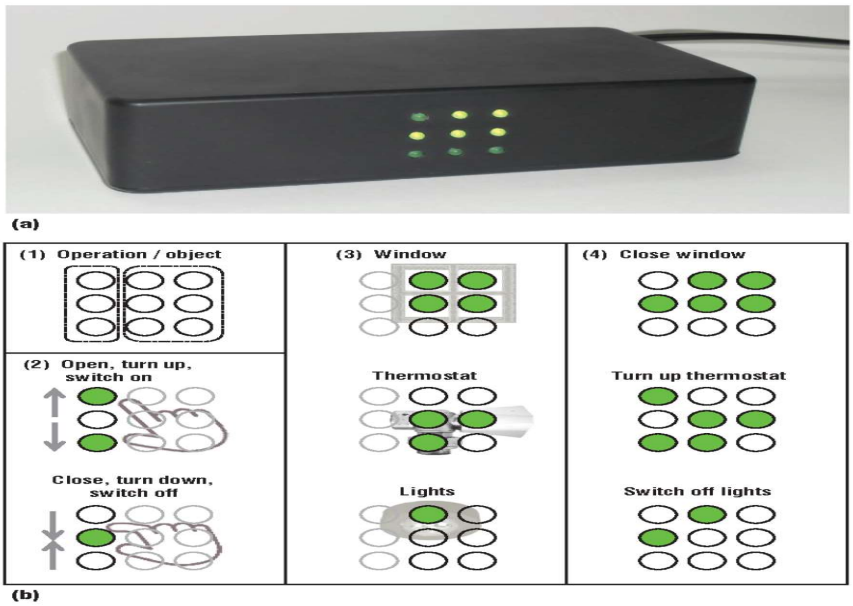


Figure 4 Soft Actuation Techniques

2.2 SENSOR BASED LEARNING

Sensors can also be used to support learning, gathering data from real users and using this data to create more realistic use cases. In a survey performed by Jan Schneide et al. [9], 82 prototypes found in literature studies were analyzed to identify the state of the art of sensor-based learning in different domains. The prototypes were classified according to their Learning Domain, Formative assessment, and Feedback. Formative Assessment goal is to check student learning continuously and provide feedback that can improve their performance. Most well-known applications that use sensors to track and give feedback are related to the fields of sports and fitness such as Digifit or Xbox fitness. The survey concluded that still there is not enough research in sensor-based platforms, so that they can become effective learning tools.

2.2.1 IOTFLIP

Flipped Learning is a pedagogical approach in which the conventional notion of classroom-based learning is inverted, so that students are introduced to the learning materials before class with classroom time then being used to deepen understanding through discussion with peers and problem-solving activities facilitated by the teacher[10]. Such activities might include solving problems together, quiz games, and other forms of interactivity. Adding IoT concepts to this pedagogical approach has been proposed. In the platform named IoTFlip (Figure 5) designed for medical students, real

data from patients is used, such as heart rate, blood pressure or glucose level. This data is extracted through wearable medical devices and stored in the IotFlip platform. Then following a Case Based Learning approach (CBL) [11], students are asked to solve a patient's case, getting feedback from their instructor.

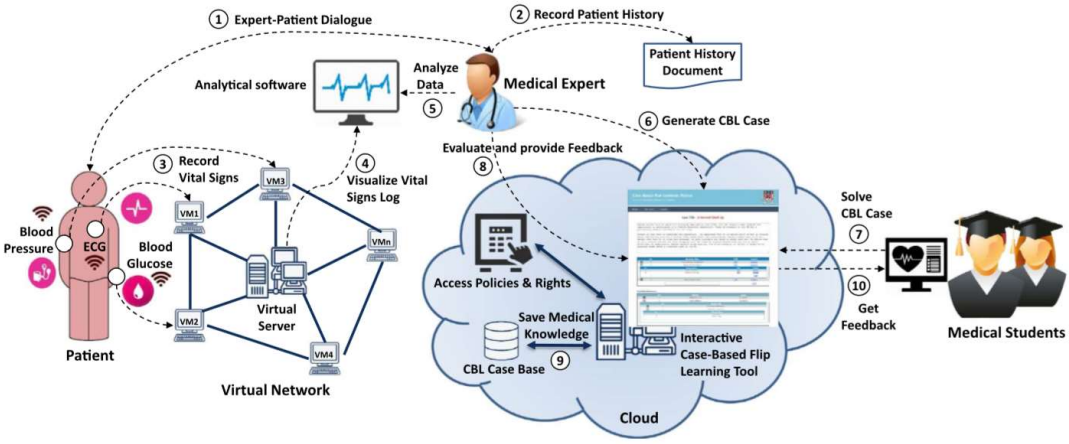


Figure 5 Iot Flip sensor Based learning

2.2.2 SUMMARY

Based on the previous research, smart campus initiatives could be sub-divided in two different perspectives. One regarding the building itself, and its management, and the other as the interactions and goals of its occupants. In terms of the building itself, the main concerns are with sustainability, such as reducing energy consumption or having a better water management system, this corresponds to the vision of a smart building. Smart buildings are defined as buildings that are technologically aware, sustainable, healthy, meet the needs of occupants and business, flexible and adaptable to deal with change. Several examples of smart buildings management systems, have been proposed and are in use in the present day, such as Intel´s smart building energy management solutions.[12] This project is more focused on the interaction of the users of the building in this case university students and how can they benefit from having an "intelligent building". Regarding sensors in a learning environment the main domains of research and commercial applications are in the realm of medical or fitness applications.

2.3 MONITORING

With the decrease average price of sensors from roughly 1.30$ in 2004 to a predicted 0.38$ in 2020 [13], it is now easier to monitor a set of environmental parameters with a relatively set of low cost devices. Although some parameters such as temperature, and humidity are easily monitored, other conditions such as occupancy of a certain room or in-door location provide a harder challenge.

2.3.1 DIFFERENT METHODS FOR PEOPLE ESTIMATION.

Several methods for estimating and counting people have been proposed or implemented, varying in the degree of accuracy and cost. In this section, some of the most usual methods are explored and compared.

2.3.1.1 COMPUTER VISION

Artificial Intelligence through the analysis of images and applying pattern recognition algorithms can provide ways to detect and count people. According to C. Raghavachar et al. in a comparative survey of human detection methods [14] one of the primary challenges in camera based approaches is the accurate detection of humans in different scenarios. Another identified challenge, transversal to most IoT application is privacy. Today there are several commercial products such as Density [15] (Figure 6) or V-Count that use this computer vision based approach. These products are mostly used in shopping centers and stores.

Figure 6 Density a Computer Vision People Tracker [15]

2.3.1.2 RFID and Bluetooth Low Energy

Another approach (Figure 7) to estimate the number of persons in a room is using wearable sensors equipped with RFID Tags or Bluetooth low Energy (BLE). An access card could also be seen as a way to have an indication of the amount of people in a determined space.

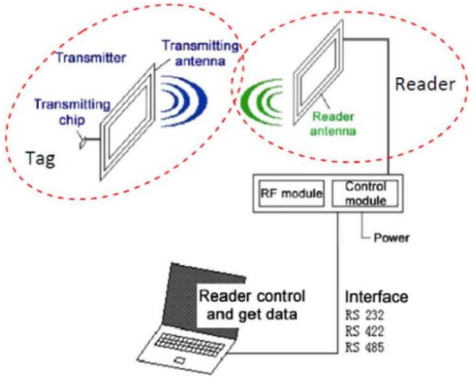


Figure 7 RFID working Principle [16]

2.3.1.3 MOTION SENSORS

Microwaves based sensors such as motion sensors can be used to count people although most frequently are used for detecting people. This type of sensors emits pulses of microwave frequencies that reflect from nearby object. If a person passes through, the signal will be disrupted. Typically, this type of sensor is used in automatic doors, or in automatic lighting. Jeong Woo Choi[17] et al conducted a proof of concept using two microwave sensors with promising results. The system was tested for one week in a metro station platform entrance (Figure 8) and had an absolute error of less than 10%. This type of approach preserves anonymity and privacy, since no private information can be extracted from people passing by and is a relatively cost-effective solution. The main problem arises when there are multiple locations that require multiple sensors, or there is not a defined pathway for entrance and exit and people can move in different directions. For this type of cases, it is harder to apply this sort of solution.

Figure 8 Sensor Module used on the study by Jeon Woo Choi

2.3.1.4 WI-FI

Using Wi-Fi signals to track people's movements has been used in different research projects. The main idea when using Wi-Fi sniffing is to capture the Wi-Fi probes or more specifically request probes. These probes are network packages that smartphones emit periodically to detect nearby Access Points (AP). This Probes contain the MAC-address of the broadcaster device. By combining this information with the Received Signal Strength Indication (RSSI) it could be possible to estimate the location of a certain device. Following this idea, a case study aimed at gathering information about spatial-temporal patterns in the movement of people in touristic destinations such as Madeira Island was conducted by Nunes et al.[18] The study concluded that passive Wi-Fi Tracking is an accurate method for estimating flows of people in Points of Interest(POI). Locals and tourists can be differentiated, and tracking information can be used to detect meaningful events such as arrivals, departures, and gatherings of people. Another study conducted by George Pipelidis et al. [19] called Human, used several beacons placed in different locations and a smartphone that broadcasted several probe requests to generate a radio map that could then be used to track differences devices. Wi-Fi sniffing might raise some

privacy concern since users have no way to opt-out, and most are not aware that their locations can be tracked this way.

2.3.2 SUMMARY

Wi-Fi was the chosen method for occupancy tracking. While the other methods have their merits and disadvantages, they all require dedicated sensors. Wi-Fi and Wi-Fi enabled devices have become ubiquitous, thus this method of people counting might yield satisfactory results, with a low entry barrier. The previous research papers are more concerned with movement tracking rather than people counting, so this project might contribute to the body of knowledge applying the same Wi-Fi tracking principle for other context using a low-cost system.

3 TECHNOLOGICAL REVIEW

In this chapter a general overview of the different design aspects concerning the development of the project are made. This chapter focuses on the different communication protocols than can be used for IoT projects, different development methodologies that the Espressif Boards can accommodate, and a description and comparison of the most common battery types available for this type of system. The different low-cost sensor options and their operating principles are also discussed.

3.1 CIRCUIT

The data collection unit, SensingBox, is a key element of this project. To find the most suitable device for this project, is necessary to analyze some of the different boards in the market today. Another important aspect is the sensors. There are a multitude of sensors that can be used, some, although widely available and used in other projects, don´t have the degree of accuracy necessary, or are too expensive. At last, the battery operation is also a significant aspect of the project, it greatly increases the portability of the SensingBox, this way the SensingBox does not have to be near a power outlet, that often is not the best location, from a data gathering standpoint. Using batteries although a good asset, increases the complexity of the project substantially, it is necessary to have in consideration when to gather data, when to put the device in “sleep” mode, how to schedule updates, amongst other details, and of course, depending on the battery, there are some safety issues that must be addressed such as the risk of explosion or fire.

3.2 SYSTEM ON A CHIP

A system on a chip (Soc) is a device that combines the requirements of several computer components integrated in a single chip [20]. The ESP32 by Espressif is one of those chips, it is the successor of the ESP8266, but features a new microprocessor, and supports Bluetooth Low Energy (BLE). Another important electronic prototyping board is the Arduino Uno, although it does not feature wireless connectivity out of the box, is an important device even from a historical standpoint and still used extensively to this day.

In the following sections these prototyping boards are presented in temporal order from first one to be launched (Arduino Uno), to the more recent ESP8266 and ESP32 from Espressif.

3.2.1 ARDUINO UNO

Born at the Ivrea Interaction institute as an easy tool for fast prototyping, aimed at students without a background in electronics, this board quickly gained popularity amongst electronic enthusiast and students.[21] The most used Arduino Boards are the Arduino Uno (Figure 9), featuring the ATmega328 microcontroller, and the Arduino Mega (Figure 10) with the AtMega2560. Developing for the Arduino boards is made through the Arduino IDE, which accepts C or C++ code and has Arduino specific functionalities.[22]

The primary advantages of using the Arduino and its ecosystem could be summarized as the following:

- Inexpensive: Compared to other microcontroller the Arduino is relatively inexpensive, costing less €20 for an original Arduino Uno and €35 for an Arduino Mega.[23]
- Cross-Platform IDE: The Arduino IDE runs on Windows, Macintosh OSX, and Linux Operating systems.
- Simplicity: The Arduino was intended to be used by students without a background in electronics or programing, this makes it easy to develop and work with.
- Extensive Community: Being the most used prototyping board, it possesses a vast community, with diverse libraries available. The same applies to hardware extensions, such as sensors.
- Code reuse: code developed for one board can easy be migrated to another board from the Arduino family.

Disadvantages.

Connectivity: Wi-Fi and Bluetooth connectivity is not present as standard in the Arduino Uno or Mega boards. Although Wi-Fi capabilities can be added using different unofficial Wi-fi shields that cost less than $10.

Power-Consumption: The Arduino microcontrollers are not well suited for battery operation due to the lack of sleep capabilities.

Computational Power: Compared to competitors especially from Espressif the Arduino boards feature less computational power.

Figure 9 Arduino Uno

Figure 10 Arduino Mega

3.2.1.1 ADITIONAL ARDUINO BOARDS

Arduino has been releasing additional boards in recent years, such as the Arduino MKR Wi-Fi 1010 better suited for IoT projects due to its Wi-Fi capabilities, MKR FOX 1200 designed for low-power applications or the Nano 33 IoT. While these boards are more suitable for IoT projects than the original Arduino Uno and Mega, the price difference is still substantially higher than its competitors. In the Arduino Store it is possible to buy the Environmental Monitor Bundle composed by the MKR Wi-Fi 1010 and the Arduino MKR ENV Shield that has the capabilities to gather Atmospheric Pressure, Temperature, Humidity, Light Intensity and UV levels. This bundle is advertised at €45 as of April 2021.

3.2.2 Espressif Family

The ESP8266 is a low-cost microchip, with Wi-Fi capabilities. It was the first microchip by Espressif to gain significant traction in IoT related projects. It was superseded by the ESP32 but due to its still lesser price compared to the EPS32, it remains a viable solution for IoT projects that do not need the additional features of the ESP32.

The ESP32 (Figure 12), is the successor of the ESP8266. The main differences when comparing the Esp8266 are the use of a dual core microprocessor and the addition of Bluetooth Low Energy (BLE). From a security standpoint, is also a more robust device featuring additional safety features that were lacking in the ESP8266 such as secure boot and flash encryption.

The ESP32-S2 is a variant of the ESP32 running on a single core with overall less features than the ESP32, but due to this it could be a potential solution for projects that have several energy constraints. Additionally, when using the ESP32-S2 in Light Sleep it is possible to stay connected to Wi-Fi which is beneficial for saving battery since a significant amount of power is used when connecting to Wi-Fi after a sleep cycle. The power consumption in Light Sleep is down 44% to 450uA when compared to the ESP32. The ESP32-S2 was scheduled to launch at the end of 2019 beginning of 2020, but due to the Corona Virus pandemic, it is still difficult to get a ESP32-S2 development board from different manufactures, only the ESP32 S2 Saola 1M and the ESP32-S2-Kaluga-1 are available as of June 2020.

Figure 12 ESP32 DevKit LOLIND32

3.2.3 COMPARISON OF DIFFERENT MICROCONTROLLERS.

Specifications	Arduino Uno Atmega2560	ESP8266	ESP32	ESP32-S2
Microcontroller	AtMega2560P	Single Core Xtensa CPU Up to 160Mhz Clock	Tensilica Xtensa LX6 32 bit Dual-Core at 160/240 MHz	Xtensa® single-core 32-bit LX7 microprocessor, up to 240 MHz
Internal Memory	8KB SRAM,4K EEPROM	160KB	520KB	320 KB SRAM
Connectivity	-	Wi-Fi 802.11 b/g/n (HT20)	Wi-Fi 802.11 b/g/n (HT20) Bluetooth,BLE	Wi-Fi 802.11 b/g/n (HT20)
External Memory	256 KB	Up to 16MB SPI Flash, Up to 4MB SPI RAM	Up to 16MB SPI Flash, Up to 4MB SPI RAM	Up to 16MB SPI Flash, Up to 4MB SPI RAM
Peripherals Support	GPIO(14),ADC(6)	GPIO(17),ADC(1)SPI(2),UART(1),I2C(1)	GPIO(34),ADC(12),DAC(8),SPI(4),I2S(2),I2C(2),UART(3),Temperature Sensor	GPIO(43),ADC(2),DAC(8),SPI(4),I2S(1),I2C(2),UART(2),Temperature Sensor
Security Features	-	-	Secure boot Flash encryption 1024-bit OTP, up to 768-bit for customers AES, Hash (SHA-2)	Secure boot, Flash encryption, 4096-bit OTP, up to 1792 bits for users, AES-128/192/256 RSA Digital signature
Operating Voltage	5V	3.0 to 3.6V	3.0 to 3.6V	3.0 to 3.6V
Average Power Consumption	45 mA	80mA	80mA	TBD
Programing Language	Arduino (C/C++)	C/C++(Arduino Core,IDF-ISP), Micropython	C/C++(Arduino Core,IDF-ISP), Microphyton	C/C++(Arduino Core,IDF-ISP), Micropython
Price(aprox as of June 2020 €)	20€ (several 3rd party option for less	4€	6€	7€
Launch Date	2005	Q4 2014	2016	Q4 2019

Table 1 Comparison of the Different Microcontrollers

In Table 1 it can be seen that there are quite substantial differences between the Arduino and the ESP32 boards. Arduino was intended as a fast and inexpensive way to prototype and not as an IoT device. Over the years several Arduinos have appeared with different characteristics, but to this day the most popular is still the Arduino Uno [24]. Compared to the boards by Espressif, Arduino is significantly underpowered in computational terms, and in the author's opinion not really suitable for IoT projects especially due to the lack of native Wi-Fi connectivity. Still, it is a valued tool for teaching basic electronics and a quick way to prototype.

The ESP8266 originally was a microchip with Wi-Fi capabilities named ESP-01. It could be added to a microcontroller such as the Arduino providing wireless functionalities and internet connection to such microcontroller. This caught the attention of the IoT hobbyist or maker community, especially considering the price of 5 USD. At the end of 2014 Espressif released an SDK allowing the chip to be programmed directly, removing the need of an external microcontroller.[25] The ESP32 is the more powerful and capable of the SoCs compared. It is more efficient, has additional GPIO and ADC pins and more security features such as flash encryption that were lacking in the ESP8266. The ESP32-S2 can be seen as a lower-powered version of the ESP32. It uses a single core MCU and has no Bluetooth capabilities.

Although Espressif boards have been gaining traction in recent years, Arduino is still the most popular ecosystem, albeit being on the decline, as a quick search in google trends can demonstrate.(Figure 13)

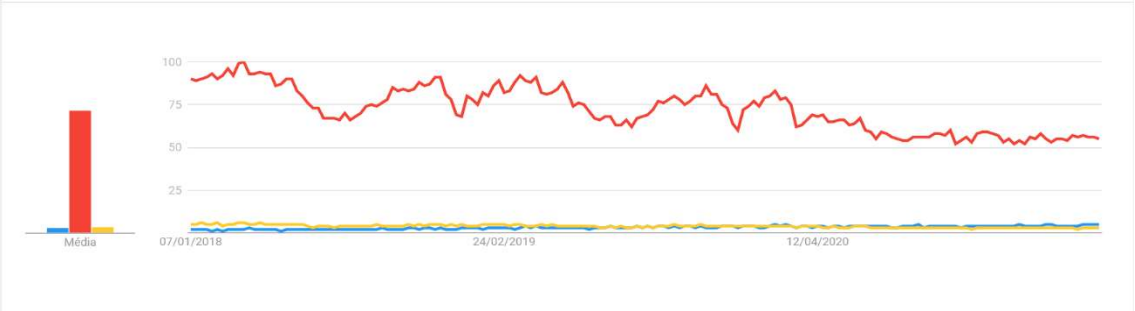


Figure 13 Google Trends Search From 2018 to 2021, Red Line Arduino, Blue ESP32 and Yellow ESP8266

3.3 DEVELOPMENT METHODOLOGIES FOR THE ESP32

Developing for the ESP32 can be achieved using two main different approaches. One is using the Espressif IoT Development Framework (ESP-IDF). The other is using the Arduino Core for ESP32 and development using in an “Arduino-like way.” The ESP-IDF makes use of the FreeRTOS [26] Kernel which is an open source real time operating system for micro-controllers. A Real Time Operating System (RTOS) is an operating system that is optimized for use in embedded/real-time applications. Their primary objective is to ensure a timely and deterministic response to events. An event can be external, like a switch being hit, or internal like a character being received.

Using a real time operating system allows a software application to be written as a set of independent tasks. Each task is assigned a priority, and it is the responsibility of the

RTOS to ensure that the task with the highest priority that is able to run, is the task that is running. Examples of when a task may not be able to run include when waiting for an external event to occur, or when a task is waiting for a fixed time period [20]. Because FreeRTOS is designed for single core used, ESP-IDF uses a customized version of FreeRTOS in order to accommodate the dual core nature of the ESP32 [27]. Another third development option is using Micropython. In the following sections these different approaches are compared, starting from the most popular ESP32 Arduino Core to Micropython.

3.3.1 ESP-32 ARDUINO CORE.

ESP32 Arduino Core is a set of libraries that act as a "wrapper" for the ESP-IDF APIs, and enable the use of several Arduino libraries, this is one of the main advantages of using this approach, another advantage is related to code portability, developing using the Arduino core, simplifies the migration of code to other platforms that share the Arduino ecosystem. As there is a significant community of makers around the Arduino platform, this method benefits from the extra libraries already available. The major drawbacks could be seen as having to use the Arduino Structures based on setup and loop. The lack of debugging options is another issue, as the overall lack of features of the Arduino IDE compared to full-fledged IDEs like Eclipse or Visual Studio Code, although it is possible to develop for the Arduino in Visual Studio Code, by using external plugins.

3.3.2 MICROPYTHON

Micropython is an implementation of Python 3 programming language that includes a small subset of the Python standard library and is optimized to run on microcontrollers and in constrained environments. As it happens with the Arduino ESP32Core Micropython is implemented on top of ESP-IDF. The main advantage of using this approach is the reduced time to develop, which is good for prototyping. The drawbacks are that usually, the code execution is slower, and more resources are used than in an implementation on ESP-IDF or Arduino Core, additionally there seems to be a smaller maker community using Micropython on the ESP32 when compared with Arduino ESP32 core and ESP-IDF.

3.3.3 DEVELOPMENT STRATEGIES CONSIDERATIONS

Initially for this project ESP Arduino core was used, and the first test prototype developed for the ESP8266 used this method. For the ESP32 development, the ESP-IDF was chosen. One of the reasons for this change was the difficulty on using the non-volatile storage library (NVS) in conjunction with the Wi-Fi Library, additionally the author wanted to experience the two approaches and delve more into FreeRTOS to gain a more profound knowledge of the ESP32. The Arduino core in the author's opinion hides and abstracts what is happening, which can be an issue when working with low level code. The author would argue that a bigger sense of control and a better understanding of embedded system development is attained when programing using the

ESP-IDF, although for small projects or quick prototypes the Arduino still proves to be a very solid development option with a massive community of makers.

3.3.4 ESP32 DEVELOPMENT BOARDS

There are several companies that provide development boards for the ESP32, the ESP32 module itself is just the microchip. It is necessary to have additionally components such as an USB to UART converter to interact with the board through USB, a voltage regulator might be needed depending on the power source used, amongst other components. Development boards provide these features. There are many different development boards that differ in the components and connectors that they provide. In this project, as power consumption is somewhat important, one electronic component present in most development boards that can have a big impact on the power consumption is the voltage regulator. A voltage regulator generates a fixed output voltage of a preset magnitude that remains constant regardless of changes to its input voltage or load conditions[28]. The recommended input voltage for the ESP32 is 3.3 Volts. When connected by USB, the 5V from the USB interface are reduced to 3.3V by the voltage regulator. A voltage regulator has a quiescent current (current necessary to power the voltage regulator), and the power dissipation, (the current that is dissipated). Different boards can have different voltage regulators with different types and operation modes. Therefore, their quiescent current and power dissipation current can vary substantially.

According to power consumption test made by Andreas Spiess [29], and another comparison made by Holger Fleischmann [30], the most economical and one of the most efficient development boards available is the D32 By Wemos, now rebranded to LOLIN. These boards use a voltage regulator from the LN1334 series, compared to the typical AMS1117 in other boards. By using this voltage regulator, the efficiency of the board while on sleep mode is greatly increased. Additionally, this board features a PH-2 2.0mm battery power connector which is also useful for this project.

3.4 BATTERY

There are several types of power sources available on the market than can power the ESP32, the operation voltage of the ESP32 ranges from 2.3V to 3.6V. When using a single power supply a 3.3V input is recommended and a minimum of 500mAh.

Typically batteries follow the same operation principle based on oxidation and reduction of an electrolyte with metals, thus generating a flow of electrons that flow from the anode(negative) to the cathode(positive),consequently powering a device connected between this two poles[31] (Figure 14). The different types of battery technologies distinguish themselves mainly on the chemical composition of their electrodes (anode and cathode) and the electrolytes. In this section the most common battery types are briefly explored and compared, to find the most suitable candidate to power the ESP32.The most important requirement to have in consideration in ensuring a minimum 3.3V input voltage and 500mAh current to power the ESP32 and the associated sensors. In addition to these requirements the battery selected should have a reduced cost per mAh of capacity and be

safe based on its electrochemical proprieties or have additional safeguards that ensure a secure operation.

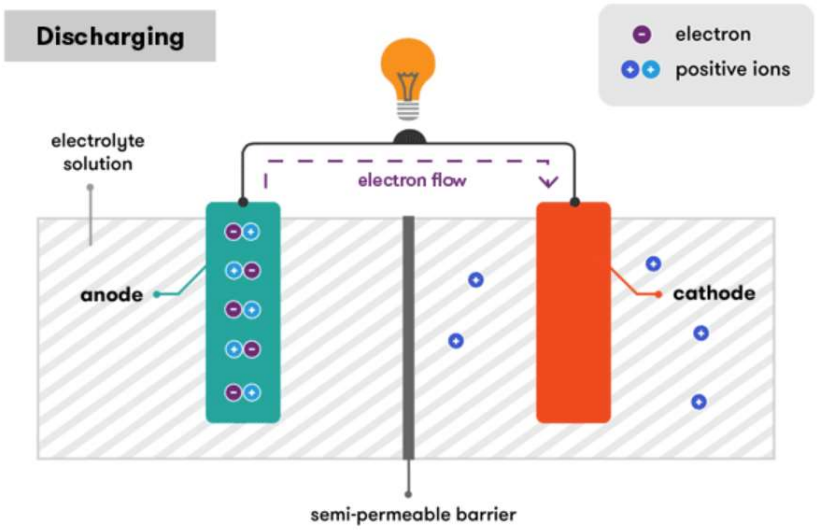


Figure 14 Battery Working Principle [32]

3.4.1 LITHIUM-ION

Lithium-ion batteries commonly referred as Li-ion are a rechargeable battery solution that use lithium ions as their key component.[33] Currently, they are used in a wide number of applications from consumer electronics to the automobile sector.

Currently, there are several lithium-ion cell designs in the market cylindrical, prismatic, button and pouch lithium-ion cells (Figure 15). One of the most commonly found formats of Li-ion battery is the 18650-cell format, which has a cylindrical shape and an output of 3.7V. It is importance to note that Lithium-ion batteries should be protected for overcharge, heat, or "over" discharge. Typically, this protection is done by a small circuit that connects the two battery poles. Not using protection circuits greatly increases the risk of fire or explosion, although the use of a circuit does not eliminate all the risks. Several incidents have happened across multiple industries, related to lithium ion batteries, even though the batteries used in these occurrences had to comply to several standards and possessed several certifications by different authorities. [34]

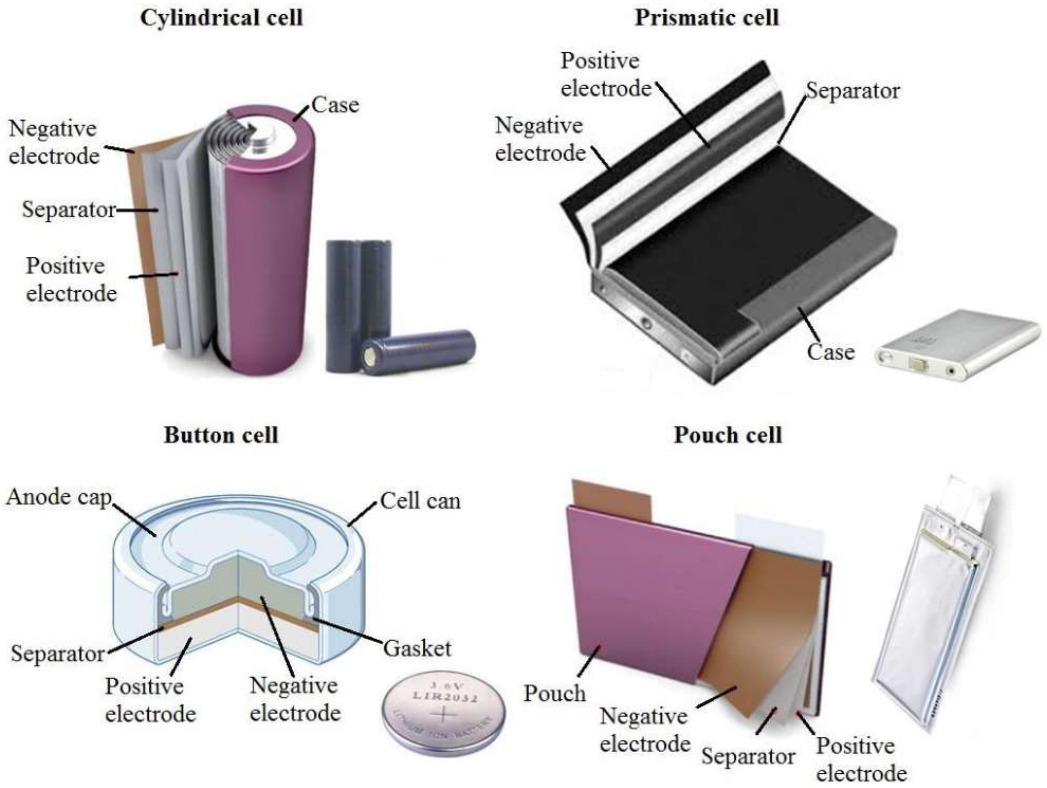


Figure 15 Different Types of Li-Ion cells [35]

3.4.2 LITHIUM-POLYMER

Lithium-Polymer or LiPo is another type of Lithium based battery technology but in contrast to the Li-ion batteries where a liquid electrolyte is used between the cathode and anode, a solid, porous or gel like chemical component is placed. Physically, most LiPo battery have a rectangular format, and typically provide a nominal voltage output of 3.7V. They can suffer from the same problems of Li-Ion batteries, being susceptible to explosion and fire if overcharged, "over" discharged, or ruptured. They are also present in a variety of industries and everyday objects such as mobile phones and power banks [31].

3.4.3 LITHIUM IRON PHOSPHATE

The lithium iron phosphate battery, typically called LiFePo4 or LiFe is a type of rechargeable battery based on the original lithium-ion chemistry but uses Iron(Fe) as a cathode material. Compared with other Lithium based batteries,LiFePO4 cells have a higher discharge current, do not explode under extreme conditions and weigh less but have lower voltage and energy density than normal Li-ion cells.[36] The nominal output voltage of these batteries is around 3.2V.These batteries are significantly pricier than other alternatives.

3.4.4 NICKEL-CADMIUM

Nickel-Cadmium, Nicd is a type of battery using Nickel(NI) and Cadmium(CD) as the electrodes, and a 30% potassium hydroxide (KOH) in distilled water [37]. Appearing firstly in 1950 they were quickly adopted until around the 1990s where they began losing space for NiHm batteries and Li-Ion due to these battery technologies being more environmentally friendly and having a better energy density and capacity. The nominal output of a Nickel Cadmium battery is typically 1.2V [38].

3.4.5 NIHM BATTERY

Nihm or Ni-Hm is a rechargeable battery type similar to NiCd where the fundamental difference resides in the use of a hydrogen absorbing alloy for the negative electrode instead of cadmium. Typically, these batteries have two to three times more capacity than a similar NiCd battery and almost as much energy density as a Li-ion Cell. As with NiCd the typical voltage output will be of 1.2V.

3.4.6 ALKALINE BATTERY

Alkaline batteries are usually disposable batteries although rechargeable variants exist, zinc and manganese dioxide are used as electrodes. The alkaline electrolyte used is either potassium or sodium hydroxide[39]. Although recharging is not recommended due to the risk of acid spillage, fire or explosion, it is effectively possible to charge these batteries, albeit for very few recharge cycles. Alkaline batteries have a nominal output of 1.5V.

3.4.7 COMPARISON OF DIFFERENT BATTERY TYPES

Battery Type	Li-ion 18650	LiPo	LifePo4	NiCd	NiHm	Alkaline
Output Voltage	3.6V	3.6V	3.2V	1.2V	1.2V	1.5V
Safety Hazards	Fire or Explosion	Fire or Explosion	-	Spillage	Spillage	Spillage
Recharge Cycles	300-500	<300	2000	2000	180–2000	-
Self Discharge Rate per month	1–2%	5%	3%	10%	10%	0.3%

Table 2 Comparison of Different Battery Types

From Table 2 the Li-Ion and LiPo batteries are the only batteries with enough output voltage to power the ESP32. The output voltage is within the ESP32 3.0 to 3.6V operation range. Although the other batteries do not meet the necessary voltage output to power the ESP32 using a single cell, they could be run in series thus reaching the required voltage. Running in series has several disadvantages as opposed to a single cell. As the capacity of the battery depletes, so does the voltage output. With multiple cells this means that a significant amount of the capacity will be wasted because the combined voltage output will not be sufficient to power the circuit even if there still some capacity left in each individual cell. Another aspect to have in consideration is the self-discharge rate of the battery, this corresponds to the amount of capacity that the battery will lose over time without being connected to anything. This can also be referred as shelf life of a battery. For this implementation, the Li-Ion battery and Li-Po battery were chosen due to their cost, voltage, and amperage output. These batteries have additional safety concerns, all of them will be used only with protection circuits that reduce the risk of a hazardous incidents. Figure 16 depicts the typical discharge rate of a Li-ion cell. Even when the battery is almost empty, it remains within the minimum voltage input range of the ESP32.

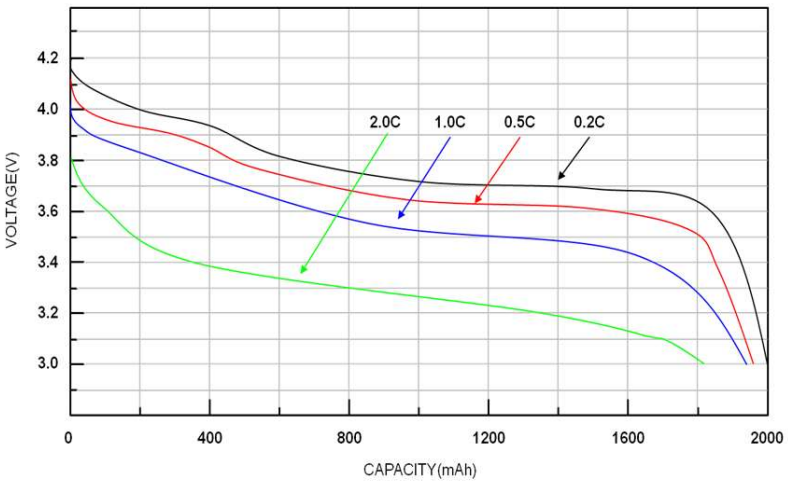


Figure 16 Discharge Rate of a Li-Ion battery [40]

3.5 SENSORS

The Selection of sensors for this project is based on Figure 1 student's preferences, availability, and cost. The initial parameters selected to be measured were sound level, humidity, temperature, air quality, and occupancy. After initial testing it was noted that the air quality sensor, the MQ135, had a heating element which in turn translates into a higher power consumption of the overall circuit. For this reason, its use was discarded.

3.5.1 SOUND SENSORS

Sound can be defined as a form of energy that is transmitted by pressure variations.[41] The difference between pressure waves called sound wave can be sensed by microphones. A sound wave is a continuous analog signal. To transform this signal into a digital signal that can be processed by an electronic device such as the ESP32 is necessary to use an ADC. The ADC will perform two steps, sampling and quantization. Sampling is the process of recording an analog signal at regular discrete moments of time. The sampling rate *fs* is the number of samples per second. The time interval between samples is called the sampling interval $Ts=1/fs$. Quantization is the process of converting the samples values to a finite set of discrete values. Figure 17 depicts these two steps. Values are sampled and replaced with an approximation of the real value with a single decimal place.

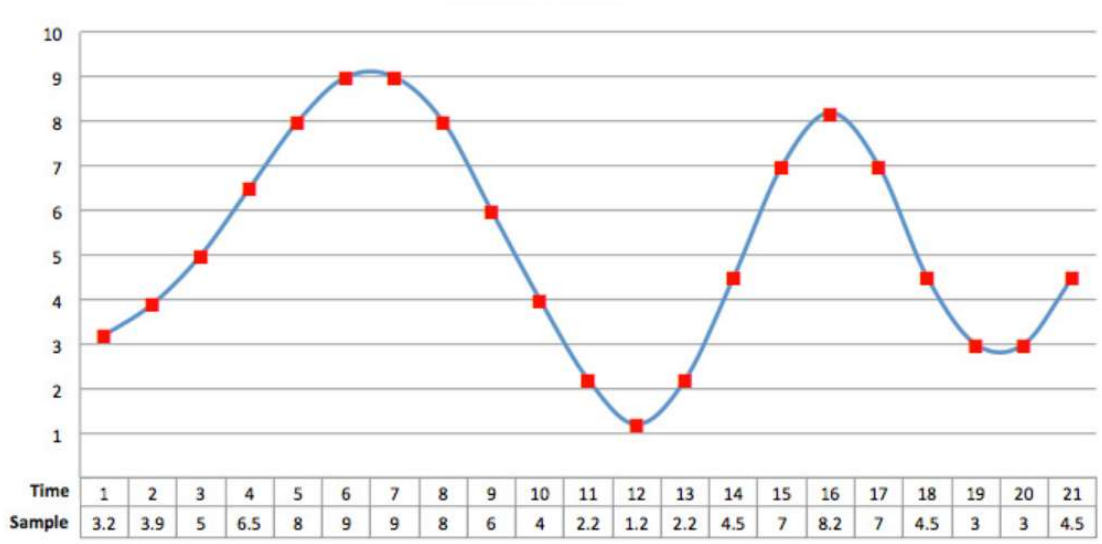


Figure 17 Sound Wave Sampling [42]

After the ADC conversion, the collected values can be converted to decibel using the formula at 1.

1. $dB = 20Log(V_{out}\%V_{ref})$

V_{out}: Voltage Output

V_{ref}: Baseline Voltage

The objective is to measure the noise level. A quick search for a low-cost sound sensor online will usually return three results the most. The KY-038, KY-037 sensor and the LM393 sensor. Such sensors are designed to detect if a sound passes a certain threshold that can be adjusted according to a potentiometer but are not suitable for measuring the sound level in decibels inside a room, due to their low sensitivity and not being pre-amplificated. The signal detected by a microphone is usually too weak to be used, therefore an amplifier commonly known as preamp is necessary to amplify the signal and detect the changes in the waves signal.

Naively a significant amount of hobbyist in the field of electronics and even some papers regarding the use of Low-cost IoT sensors try to use sensors that are not suitable for measuring sound but only for detecting it. When tested, the KY-038 with the potentiometer set to around 2.8v, for a quiet room, a clap would bring the sound output to around 3.2v on the ESP32 ADC, with some of the readings fluctuating wildly. This is in part due to the nature of the ADC in the ESP32 that has a significant amount of noise even when using a 10uf capacitor connect to the ADC input as recommended by Espressif.

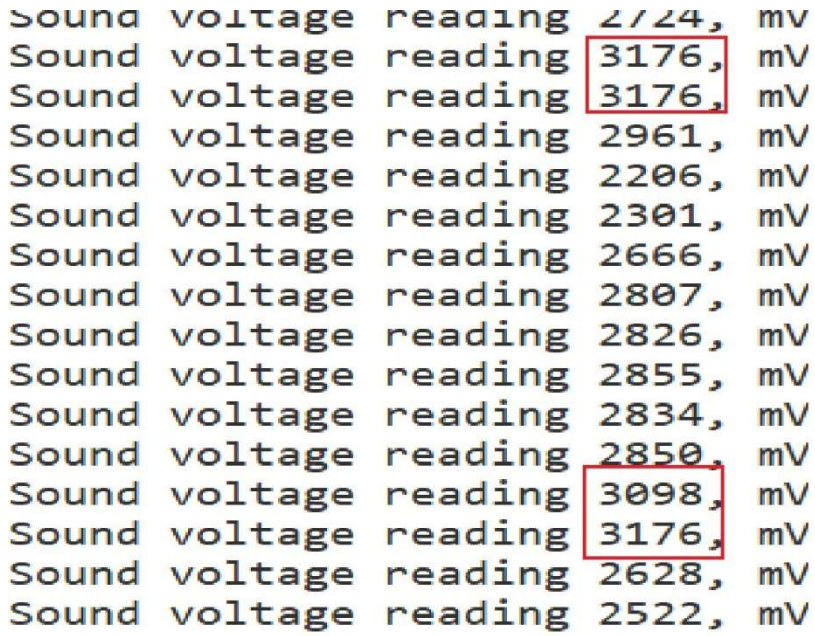


Figure 18 Sound Detection Tests with Ky-038

The LM393 sensor does not have an analog output, only a digital pin is present, thus the digital signal is either Low or High depending if a sound was detected.

After realizing that these sensors are not adequate for measuring the sound level in a room, a new query was made for a more suitable sensor that had a built-in amplifier.

Instead of finding new sensors, an alternative approach could be a relative comparison where the same sound detection threshold is set in all the sensors and the number of detections that exceed the defined value are counted. A sensor that had detected more sounds over the threshold would belong to a noisier environment. This approach would probably be excessively flawed, a certain location could have more detection of a sound that barely passes the threshold, while other could have a more sporadic detections but far more intrusive and perturbing sounds, for this reason this approach was discarded.

The most affordable sensor found with a built in amplifier and already used in some other projects [43] is the MAX4466. This sensor features an amplifier with a gain of x25 to x125, thus can output signals amplified by up to one Volt.

Given in consideration than this sensor has a sensibility of -46dB, it is necessary to subtract this value to the result found using the formula (1).

3.6 TEMPERATURE AND HUMIDTY

One of the most common examples related to IoT is building a thermometer or a hygrometer. Temperature and Humidity gathering is one of the easiest ways to introduce someone to a basic IoT device use case. The most common and less expensive sensor is the DHT11 from the DHT sensor family. This sensor can measure temperature 0-50 °C ±2°C range and 20-90% ±5% RH. Other sensors such as the DHT22 or the SHT10 have

better accuracy and range but at a higher cost. For a project where there is not a necessity to measure negative temperatures or have a very accurate reading, a DHT11 is adequate.

3.7 ESTIMATING OCCUPANCY

As referred in section 2.3.1.4 from the multiple ways of detecting and counting people a Wi-Fi based approach was selected, where the probes of a user devices are detected and captured, hence being a proxy to the number of people present in a certain environment. This method was chosen in favor of other methods due to ubiquity of Wi-Fi devices, especially smartphones, it does not require additional sensors, and has been explored in different projects before [19], [44] although from a tracking perspective. If indeed feasible this approach could be economical.

As potential drawbacks it must be considered that a person might carry several Wi-Fi enabled devices, devices in adjacent rooms might be detected, and more specific to this project, having a ESP32 running on battery and using Wi-Fi for extended periods has a significative impact on its battery duration. Some concerns could be raised regarding privacy, but in the final system, the probes detected containing MAC address are not being stored or used in conjunction with other information to identify a concrete individual, therefore this problem is mitigated.

The Wireless Local Area Network (WLAN) technology is defined by the IEEE 802.11 family of specifications[45]. These specifications define the structure of the wireless frames used for communication. The MAC Sublayer Frame Structure (Figure 19) corresponds to the Physical Layer and Data Link Layer of the OSI model.3 different Frame Types exist: Management, Data, and Control. A probe request is a subtype of a Management Frame; thus, the objective is to capture these frames by setting the ESP32 in promiscuous mode. While in promiscuous mode the ESP32 accepts all the incoming Wi-Fi signals, although only probe requests are of interest for this case.

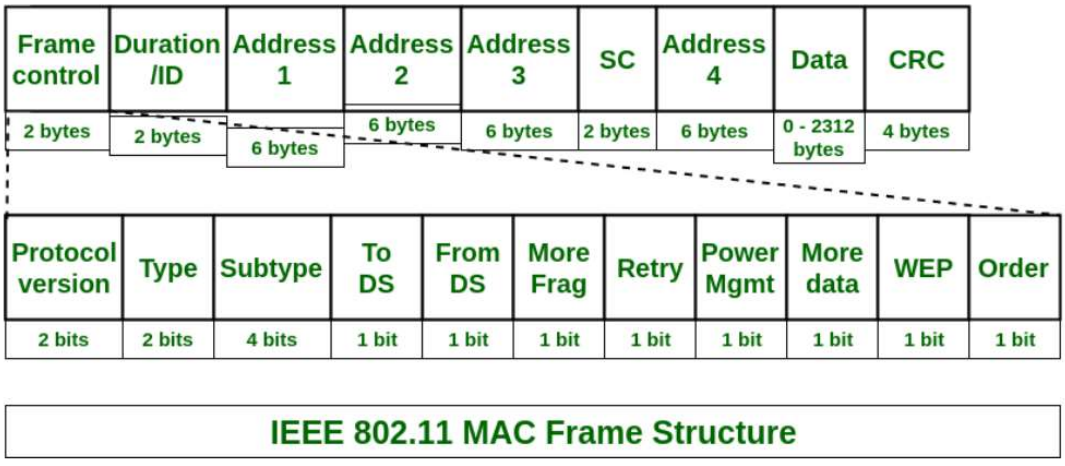


Figure 19 IEEE 802.11 MAC Frame Structure [46, p. 8]

The most important part of the MAC Frame Structure that is necessary to capture is the Address 2 seen in Figure 19. This corresponds to the MAC address of the sender and

thus a possible device. Smartphones employ MAC address randomization techniques in order to preserve the privacy of its users and difficult their tracking[45]. In this project the goal is to estimate the occupancy of a specific location, so being able to identify and isolate an individual is not necessary, the important matter is to detect a device, although a MAC randomization could influence this, because a device could emit two different probes with two different MAC address that correspond to the same device.

Prior to Android 10, only when searching for networks would a mobile device randomize its MAC address, from Android 10 onwards this is the default behavior, IOS 14 follows a similar approach[47].

3.7.1 CHANNELS

A channel can be defined as a range of frequencies with a defined center frequency[48]. The IEEE 802.11 protocols defines up to 14 channels (Figure 20) in the 2.4Ghz frequency roughly 20MHz apart from each other. To capture the request probes is necessary to listen on these channels. Depending on the geographic location the number of channels used differs. In the US, 11 channels are used (channels 12,13 and 14 are excluded) while in Europe it is 13 (channel 14 excluded).

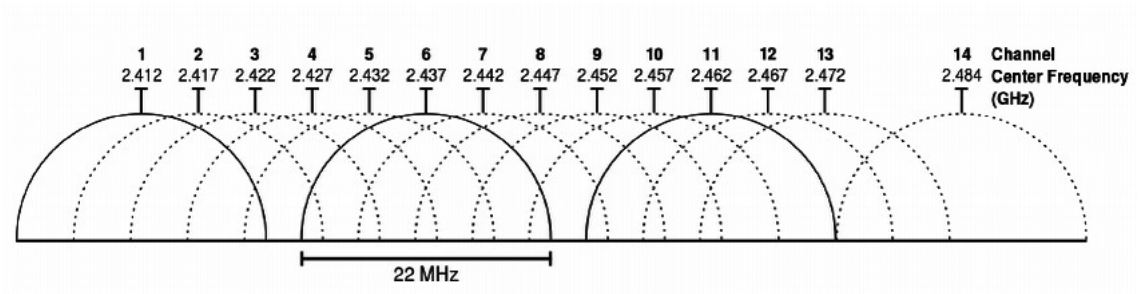


Figure 20 IEEE802.11 Available Wi-Fi Channels [49]

Miguel Ribeiro et al. [44] used several routers set in promiscuous mode, flashed with an open-source firmware *OpenWRT,* to harness the probes emitted in different location around Madeira island. The study ran from 2016 to 2020 and was observed that the number of randomized MAC addresses has been on the rise.

3.7.2 SUMMARY

Taking into account these findings from previous studies, the algorithm proposed for getting an estimation of the occupancy of a certain location, could feature filtering options when processing the packages. This filtering options could provide a more accurate estimation of the occupancy of a certain zone and could be tailored to different locations. For instance, if in a certain location it is known in advance that a family of equipment such laptops are constantly connected, a filter to ignore this family of MAC addresses (manufacturer identifier) could be implemented. Or the reverse could be true, in a location where everyone uses the same family of devices, filter the incoming probes to only process those from that specific manufacturer. This way the number of occupants should

be closer to the number of unique probes detected. These two examples are very specific to these use cases, for a more general approach, the filtering options below are proposed.

- **Minimum and Maximum Number of Same Probe Occurrences**: This value can be set according to the sensor and its location. When processing the probes, sender MAC addresses that do not appear a minimum period. In the articles cited above its possible to see that the number of MAC randomization implemented by companies is increasing, while some still use the default google implementation resulting in the address (DA: A1:19) being one of the most common, as more companies implement their own randomization techniques different addresses appear more frequently. The recommended minimum number of probes occurrences could be set to two or other value.
- **Minimum and Maximum RSSI:** RSSI corresponds to the signal strength that reaches a certain device. This in turn permits that some gross location discovery to be implement as seen in [50],[51] usually the stronger the signal received, the closer a certain device will be from its emitter, although due to interference, signal refraction, or attenuation this might not always be the case.
- **Target Specific Channel**: In this mode only the probes of one channel will be captured ignoring all other channels, this might prove useful in certain circumstances, such as when it is known in advance about the channel that is being used in.

3.8 COMMUNICATION PROTOCOLS

Regarding IoT applications, the most known and discussed protocols are HTPP, MQTT and CoAP. Although HTTP has become the standard regarding web communications, IoT devices present new challenges such as constrained resources (i.e., battery operation), limited bandwidth available, thus the need of having different protocols with different characteristics.

3.8.1 HTTP

At the core of the World Wide Web (WWW) is the HTTP protocol, being used for over 30 years since its first use in 1989 and first documented version HTTP v0.9 in 1991. The protocol creation is attributed to Tim Berners-Lee also the inventor of the World Wide web, and his team at CERN.

3.8.1.1 HTTP MAIN ATRIBUTES:

- **Stateless**: The Server does not keep any information about the client after the response is send.
- **TCP connection**: Before a client and a server can exchange HTTP request/response pair, they must establish a TCP connection. For each HTTP request/response made a new TCP connection is open. HTTP relies on TCP rather than UDP, because the protocol requires a reliable way to send messages, this is guaranteed by using a TCP connection, instead of UDP. Figure 21

exemplifies the working principle of HTTP, where a client requests resources (HTML) from a server that are then sent back to the client.

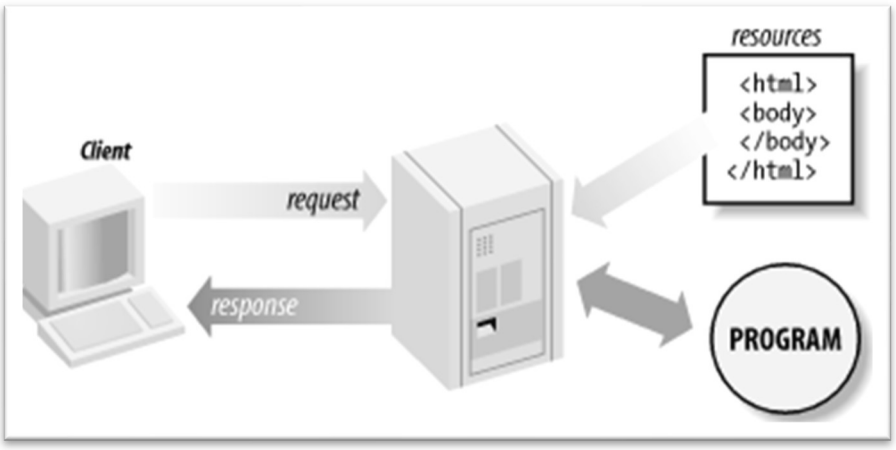


Figure 21 HTTP Functioning Mode [52]

3.8.2 COAP

The First Draft of the Constrained Application Protocol (COAP) was established in June 2014, by the IETF Constrained RESTFul Environments (Core). The main objective was to provide a communication protocol for constrained devices in a HTTP-Like request/response manner. CoAP has two basics layers the request-response layer as happens with HTTP, (Figure 22) and the transactional layer, that handles single messages exchanges between endpoints. CoAP also features a resource/observe model similar to MQTT publish/subscribe model, where an observer subscribes to a certain variable and is notified when changes occur. CoAP is connectionless, using UDP as a transport protocol instead of TCP. As UDP is not reliable, CoAP implements its own confirmation model.[53], [54]

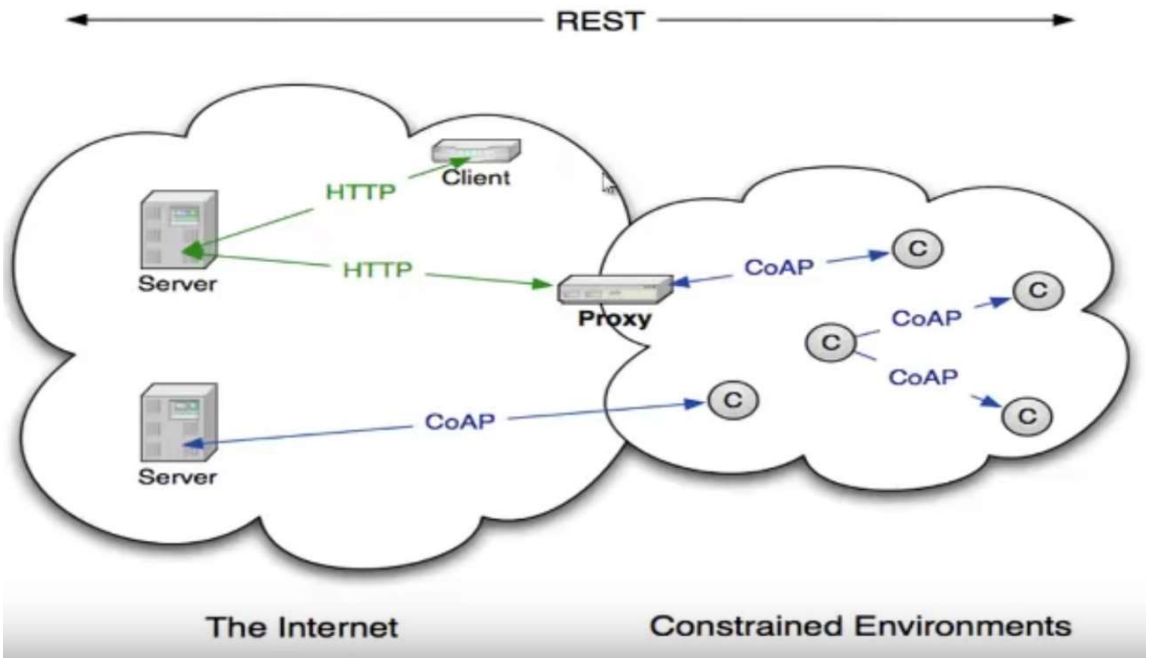


Figure 22 Coap [55]

3.8.3 MQTT

The MQTT protocol was invented in 1999 by Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, now Cirrus Link). They needed a protocol for minimal battery loss and minimal bandwidth to connect oil pipelines via satellite.[56]. MQTT follows a Publish/Subscribe (Figure 23) model, where a client subscribes to one or more topics. A publisher can also be a client. A Broker then handles and routes the messages accordingly.

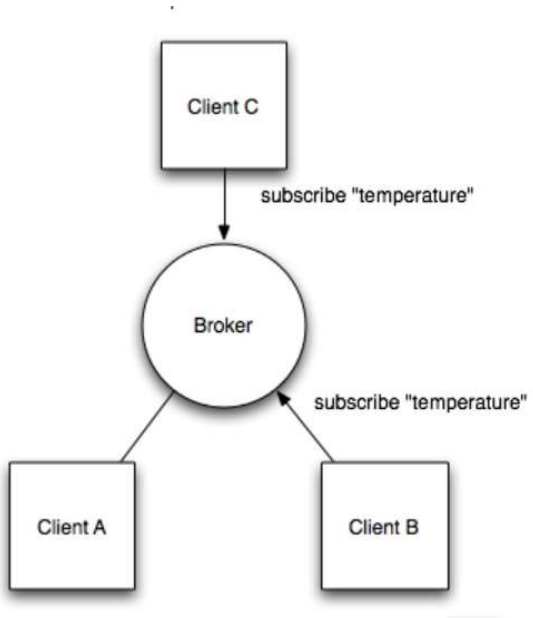


Figure 23 MQTT Conceptual Architecture [57]

Nitin Naik [58] conducted a survey comparing the different protocols. For the most important requirements for this project, power consumption, bandwidth use and security, CoAP has the best theoretical performance amongst the different protocols, with MQTT coming in second regarding these parameters. HTTP is not recommended by the study as a protocol for the IoT industry, what is excepted, given the high overhead of this protocol which was not designed with IoT in mind. Another key element was standardization, MQTT is the most used, and supported protocol in IoT, CoAP being third, being a much more recent protocol might be one of the reasons for this. A summary of this study can be seen on Table 3.

Another paper by Dinesh Thangavel et al. [54] proposes a common middleware that could select between MQTT and CoAP depending on the network conditions. It was observed that messages exchanged using MQTT experienced lower delays than CoAP when packet loss was low, and higher delays when package loss was higher. CoAP also generates less traffic than MQTT. These findings are in line with the findings in the study conducted by Nitin Naik survey.

Criteria	MQTT	CoAP	AMQP	HTTP
1. Year	1999	2010	2003	1997
2. Architecture	Client/Broker	Client/Server or Client/Broker	Client/Broker or Client/Server	Client/Server
3. Abstraction	Publish/Subscribe	Request/Response or Publish/Subscribe	Publish/Subscribe or Request/Response	Request/Response
4. Header Size	2 Byte	4 Byte	8 Byte	Undefined
5. Message Size	Small and Undefined (up to 256 MB maximum size)	Small and Undefined (normally small to fit in single IP datagram)	Negotiable and Undefined	Large and Undefined (depends on the web server or the programming technology)
6. Semantics/ Methods	Connect, Disconnect, Publish, Subscribe, Unsubscribe, Close	Get, Post, Put, Delete	Consume, Deliver, Publish, Get, Select, Ack, Delete, Nack, Recover, Reject, Open, Close	Get, Post, Head, Put, Patch, Options, Connect, Delete
7. Cache and Proxy Support	Partial	Yes	Yes	Yes
8. Quality of Service (QoS)/ Reliability	QoS 0 - At most once (Fire-and-Forget), QoS 1 - At least once, QoS 2 - Exactly once	Confirmable Message (similar to At most once) or Non-confirmable Message (similar to At least once)	Settle Format (similar to At most once) or Unsettle Format (similar to At least once)	Limited (via Transport Protocol - TCP)
9. Standards	OASIS, Eclipse Foundations	IETF, Eclipse Foundation	OASIS, ISO/IEC	IETF and W3C
10. Transport Protocol	TCP (MQTT-SN can use UDP)	UDP, SCTP	TCP, SCTP	TCP
11. Security	TLS/SSL	DTLS, IPSec	TLS/SSL, IPSec, SASL	TLS/SSL
12. Default Port	1883/ 8883 (TLS/SSL)	5683 (UDP Port)/ 5684 (DLTS)	5671 (TLS/SSL), 5672	80/ 443 (TLS/SSL)
13. Encoding Format	Binary	Binary	Binary	Text
14. Licensing Model	Open Source	Open Source	Open Source	Free
15. Organisational Support	IBM, Facebook, Eurotech, Cisco, Red Hat, Software AG, Tibco, ITSO, M2Mi, Amazon Web Services (AWS), InduSoft, Fiorano	Large Web Community Support, Cisco, Contiki, Erika, IoTivity	Microsoft , JP Morgan, Bank of America, Barclays, Goldman Sachs, Credit Suisse	Global Web Protocol Standard

Table 3 Comparrison of the Different IoT protocols[58]

While CoAP appears to be a very good candidate for IoT applications and for this project, there are still some drawbacks that push the decision to MQTT protocol. CoAP being a newer protocol, still lacks support and documentation when compared to MQTT. For the ESP32 several examples of an MQTT client implementation are available whether in the ESP-IDF SDK, or in the form of Arduino Libraries. As MQTT is a tried and tested solution and being the most used IoT communication protocol, it was chosen for this project.

3.9 WEB-TECHNOLOGIES AND PERSISTANCE

In the previous sections, hardware and lower-level details were reviewed as communication protocols. Moving now to a higher abstraction level, the relevant Web Technologies for implementing this part of the system are presented.

One of the main criteria for choosing some of the technologies presented in this section, were the author's, previous experience with them, so while other approaches could be possible, they would probably result in an extended development process.

3.9.1 BACK-END

Mosquitto MQTT does not possess a persistence mechanism, so it is necessary to develop a way to persist the data collected into a database. While some commercial applications such as HiveMq[59] do have mechanisms that ensure the persistence of the messages posted, such methods are limited. In order to process the incoming MQTT messages and store them, it is necessary to have an intermediate layer, this middleware or back-end layer, could be developed using several different programming languages and Frameworks such as Laravel (PHP), Spring (Java), Express (Node.js) .ASP.NET (c#) or even Django(Python) which is gaining ever-increasing momentum, fueled by Data Mining and AI research and development, that heavily use Python. Due to the author's familiarity with Java and JavaScript especially Spring and the Express framework respectively, only these two approaches were considered. The chosen language ended up being Javascript, with the run-time environment being Node.js with Express framework. Node.js was chosen because of its rapid development time when compared to other back-end languages such as Java. For use cases that rely heavily in I/O tasks, Node.js non-blocking single threaded nature is an advantage, both in request received and processed, as in development simplicity when opposed to language where is necessary to the programmer to deal directly with thread creation and management. Another factor is that JSON is the "de facto" standard, as a data interchange format, and supported natively in JavaScript. All these aspects make it the most suitable choice for this use case. Naturally, this project is not of commercial intent and is not expected to have millions of users where these considerations would have a practical effect, thus given identical practical outcomes, the familiarity of the software developer with the tools, should be one of the main criteria.

3.9.1.1 NODE.JS

Node.js is an asynchronous event-driven JavaScript runtime. It was originally developed by Ryan Dahl and runs on Google V8 engine (C++). When node.js first appeared, its main objective was to increase the number of concurrent access a typical HTTP server could handle. This was achieved using a non-blocking I/O approach where an event loop handles (Figure 24) the incoming requests using a single thread, in a non-synchronous way. Although node.js is referred as asynchronous and single threaded, in reality the V8 engine is multithreaded and synchronous but these details are abstracted from the developer using Node.js. Another big plus of Node.js is that it brings JavaScript which was typically a client-side language to the back end making it easier for front end developers to transition to back-end. Node.js uses npm, (node package manager) as its package manager, giving developers the possibility to add external libraries to the system.

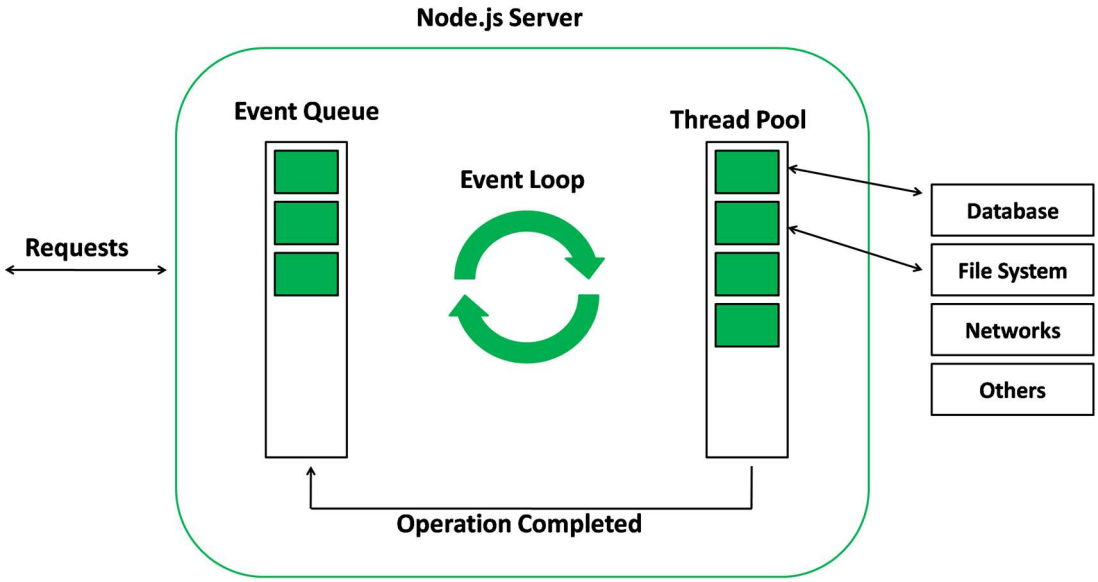


Figure 24 Node.js Event Loop [60]

3.9.2 PERSISTANCE

Traditionally, relational databases have been the go-to solution for storing data. Relational databases store the data in a structured way with different relations being established between tables. In recent years non-relational Databases, NoSQL have been on the rise, this is part due to Big Data, as relation databased are pushed to save more data, thus increasing the necessity for the host machine for more RAM or Disk Space which eventually reaches a limit. The alternative is to distribute the load across several machines. While this might at first glance, seem an unproblematic solution, relational databases were not designed to be easily distributed. Joins and transactions are at a premium in distributed environments with Terabytes or Petabytes of data and millions of users. While the reduced data duplication was an important feature in the dawn of modern development, as the cost per MB of storage decreases, this feature becomes of less importance, the focus becomes on scalability. NoSQL attends to this limitation by using a new way of storing data in a folder like way. For the necessities of this project, there is

no clear answer, again, this project is not supposed to scale to the point where the differences between the two storage types would matter, although, it is true, that IoT projects are referenced heavily in the literature as a common example of applicability of NoSQL solutions, but usually in the context of millions of devices. Another less known type of database, but sometimes suggested for IoT systems, is a time-based storage system, a Time Series database TSDB. As the name suggests, data is stored as a series of time-value pairs.

Database Type	Advantages	Disadvantages	Most known DBMS
Relational Database	Tried and tested technologies. Several DBMS available. ACID.	Scalability issues. Structured data emphasizes necessity to have well defined data models. More complex compared to NoSql.	PostgreSQL. MySQL. Oracle.
NoSQL	Better Scalability. JSON Like Structure.	For smaller database performance is inferior to relational databases. Possible Data Duplication.	MongoDb
Time Series	Excellent when dealing with values that always need to have an associated timestamp	Still relatively new in the market when compared to more traditional Db types.	InFluxDb. TimeScaleDB.

Table 4 Database Types Comparison

After pondering and comparing (Table 4) the different Db types, a SQL relational database was chosen. PostgreSQL is an open source RDMS, pgAdmin provides an UI to interact with the database schema more easily. PostgreSQL could also support Time Series database types by using an PostgreSQL extension called TimeScaleDb.[61], this was one of the reasons, for choosing this database, a potential switch to this type of format, for some part of the system could be feasible in the future, while still maintaining a structured format.

3.9.3 PRESENTATION

Currently the three main front-end frameworks or libraries are Angular, React and Vue. Angular is the oldest of the three and was developed by Google, has been on the market since 2010. React is a library developed by Facebook and was initially released in 2013. Vue is the most recent of the three and was initially released in 2014. It was developed by an Ex-google employee Evan You [62].

For the presentation layer of this system, Angular was chosen due to its very good documentation, extensive support, and the author's limited previous experience with the framework. Although being the most complete framework of the three (React is an Ui Library), thus being favored for enterprise development, the learning curve can be steeper when compared to the other two.

The first version of Angular named AngularJs was released in 2010, it used JavaScript as its programming language, and was intended as a framework to develop single page applications where the content of the page is dynamically changed. With the release of Angular2 a switch to Typescript was made from this version onwards Angular was simply referred as Angular. Typescript is a superset of JavaScript. In Angular the code written in Typescript is translated to JavaScript. Typescript possesses some functionalities that were lacking in JavaScript. An implementation of OOP closer to other languages such as Java or C#, instead of the object-based prototype version of JavaScript released with ES6. Another difference is the addition of types, hence the name Typescript. A variable might have a type such as a String, or Number, some additional features such as Class decorators that permit to modify a class's behavior are amongst the features provided by Angular.

4 IMPLEMENTATION

After an analysis of the relevant technologies necessary for this project in the previous chapter, the implementation begins. This chapter is divided into three main sections, one regarding the development of the SensingBox, one for the back-end system and persistence layer, and finally the data presentation. Key concepts of the ESP-IDF and FreeRTOS are also presented, to provide a better understanding of the conduced development. All the development code will be made available in a public repository on GitHub. These repositories can be found at:

1. SensingBox ESP32 Software: https://github.com/LeoCf/SensingBox_ESP32
2. SensingBox Back-End: https://github.com/LeoCf/SensingBox_Node_server
3. SensingBox Angular Front-End: https://github.com/LeoCf/SensingBox_Angular

The current website of the project can be found in the following link: https://umasense.live/pt/home

4.1 ANALYSIS AND REQUIREMENTS

In this section, an analysis of the project is made, regarding the different considerations and aspects of the development process. The functional and non-functional requirements are defined, and a high-level view of the system is presented. (Figure 25)

Requirements gathering and analysis are an integral part of any software or system development. They can be divided in two categories such as Functional Requirements, what the system should do, and Non-Functional Requirements that describe not what the system will do, but how the system will do.

Occasionally, functional requirements when part of a Software Requirements Specification document (SRS), have a must/want column that has the priority of such requirements, or/and have additional columns where comments can be made. In real life project where multiple developers and teams are present, this makes some sense since it helps to prioritize the different requirements. Some research also encourages this approach.[63], [64] In this case and since this project implementation is not made by a large team, the functional requirements here defined will have no implicit priority assigned to them. Table 5 list the functional requirements for the project, with Table 6 accounting for the non-functional requirements.

Nº	Functional Requirements
RF01	The system devices must be able to collect temperature, humidity, and sound level from different locations.
RF02	The system must estimate the occupancy of different locations using Wi-Fi probe request capture.
RF03	The system must save the collected data from RF01 and save the estimation from RF02.
RF04	The system must provide a way to visualize the data specified by RF01 and RF02.
RF05	The system must be able to manage sensor's locations (associate device to locations).
RF06	The system must permit it's connected devices to have their configurations changed remotely.
RF07	The system's connected devices must be able to send data using a Wi-Fi residential network, or Enterprise Network.
RF08	The system should have fail-safe mechanism in case of network instability, ensuring the captured data is not lost due to disconnections.

Table 5 Functional Requirements

Nº	
NFR01	The data gathering portion of the project should be decoupled and reusable in a different context.
NFR02	The system should not be tied to a specific infrastructure.
NFR03	The system should provide a clear separation of concerns between presentation, and business logic.
NFR04	The system should provide external API enabling the collected data to be used by other systems.
NFR05	The System Presentation Layer should be adaptable to different devices (Responsive).
NFR06	RF05 and RF06 functionalities should be limited to authenticated users.

Table 6 Non Functional Requirements

4.2 SYSTEM HIGH LEVEL VIEW

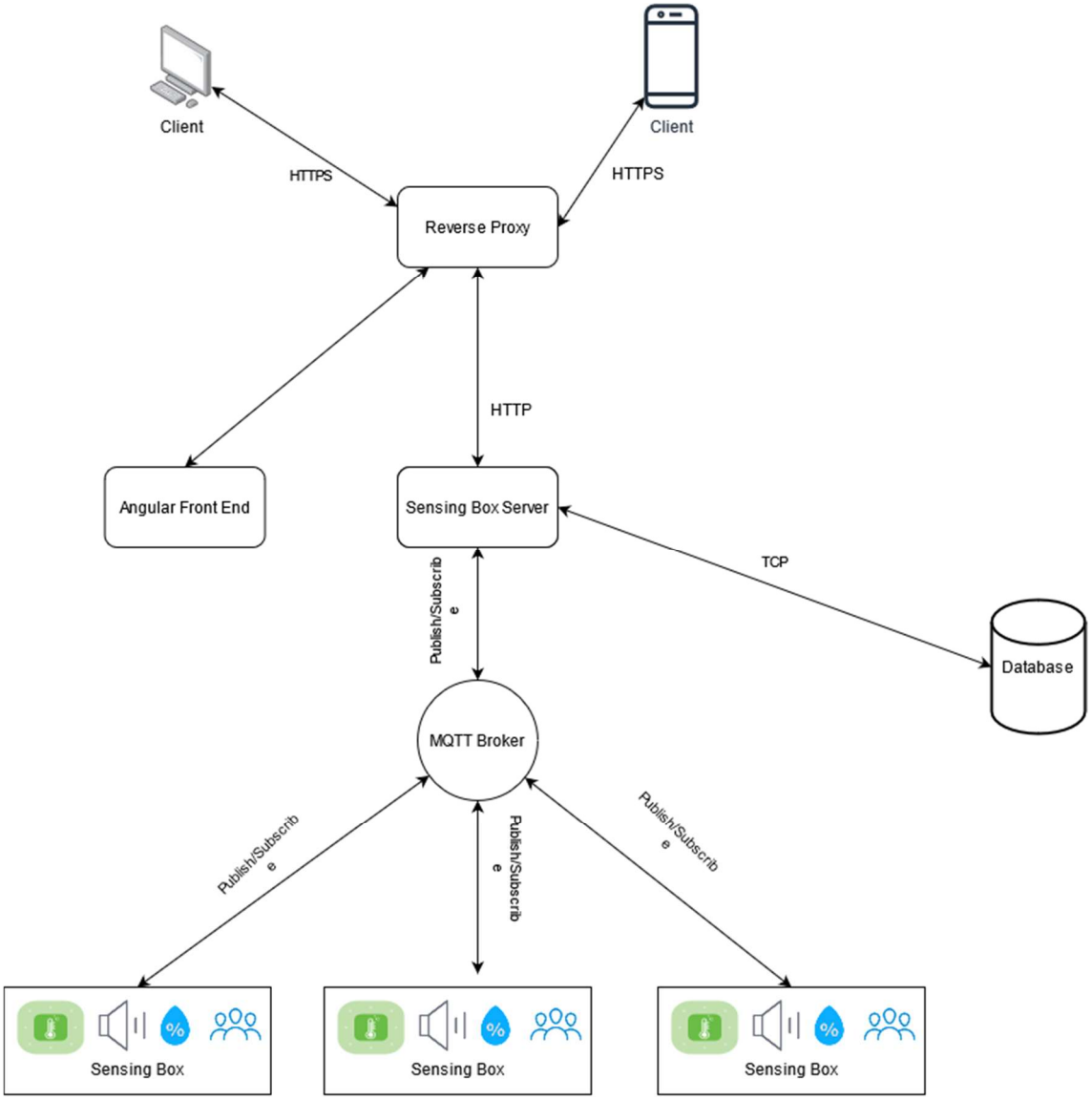


Figure 25 High Level View of The System

The system can be decomposed in two parts, one related to the sensor data gathering portion and other to the presentation and processing the data collected. The first part comprises the sensing boxes the MQTT broker, server, and database. The other part is concerned with the application itself and presentation of the data collected and is composed by a backend application and a frontend web server. A reverse proxy is used. This is a common practice in Web development, a reverse proxy provides a single external interface for its clients. While the incoming traffic has a single entrance point and looks the same from an external perspective, the proxy can route the traffic to the different servers depending on what type of requests are being made. Scaling is another issue that modern applications face. Typically a server might receive more request than it can handle, thus more servers are required, or more computational power is needed, the first approach is called horizontal scaling the second approach vertical scaling. To distribute the request amongst multiple servers, a reverse proxy can also work as a load balancer

distributing the request throughout the server's instances. Sometimes, one is more beneficial than other. For example, given the single threaded nature of Node.js vertical scaling is not as beneficial as horizontal scaling, although a combination of the two can be applied, in most cases, cost is the driving factor of such decision.

In this project, typically the expected incoming traffic load, is not large enough to warrant the use of load balancing capabilities, thus this functionality will not be present. Reverse proxy can also add additional features such as caching mechanism, and request authentication.

4.3 SENSINGBOX

The core or usual functionality of the sensing Box can be seen as a sequence of steps, and 3 major states: Data Gathering, Data Publishing, and System Sleep. (7) In these 3 states, its necessary to account for other actions that can happen occasionally, such as receiving configurations, configuring the SensingBox for its first use, dealing with Wi-Fi disconnections, or resetting the SensingBox to its original state.

In this section the SensingBox circuit and development is explored with emphasis in the functionalities developed and their implementation that support these 3 states and fulfill the system requirements. The circuit and its components are presented in section 4.3.1, and from sections 4.4.2 onwards until 4.5, the development of the different functionalities is described.

4.3.1 SENSINGBOX CIRCUIT

The SensingBox is protected by a cardboard box Figure 27 , using a cardboard the attenuation of incoming or outgoing signals is negligible. All the used sensors are placed in a 400-point breadboard. Figure 26 exhibits the inside view of the circuit, and the identified components are the following:

1. Liitokala 3500 mAh Lithium-Ion Battery
2. DHT11 sensor
3. MAX4466 Sound sensor
4. 10uf Capacitor
5. LolinD32 board with ESP32-WROOM
6. System Status LED
7. System Switch

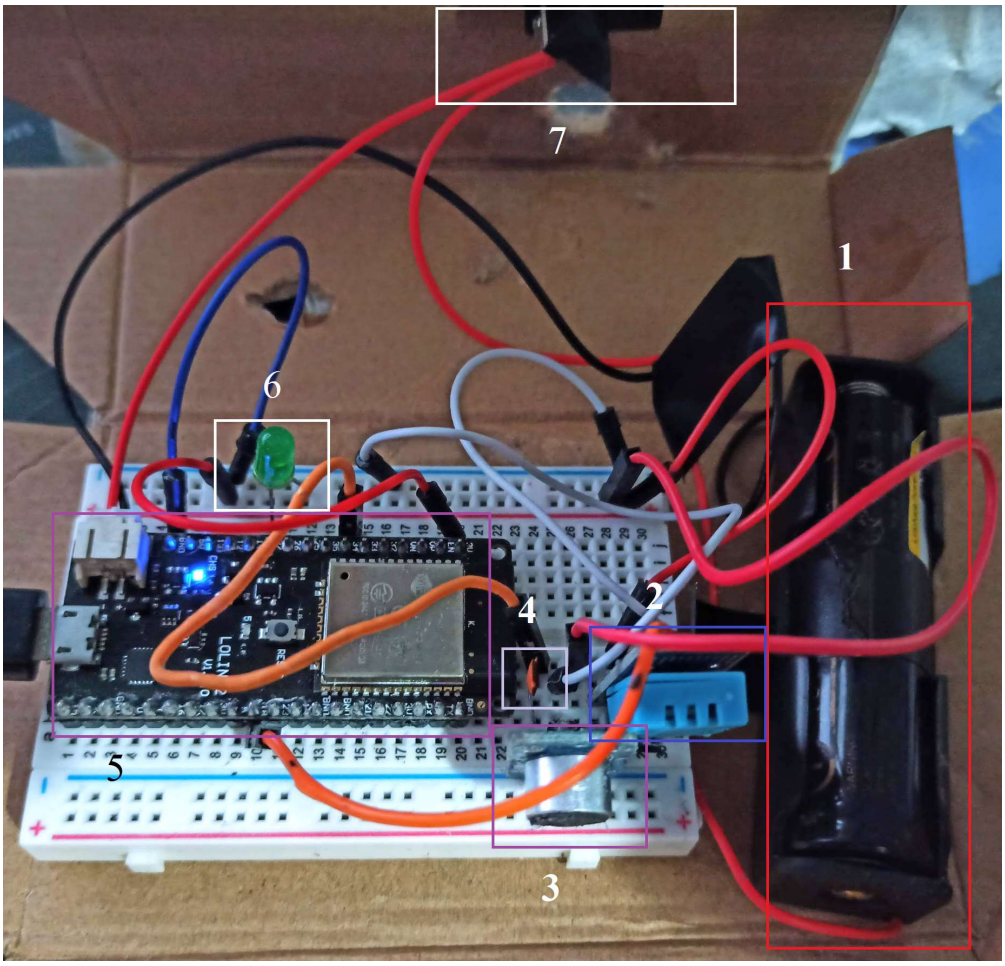


Figure 26 SensingBox circuit inside view

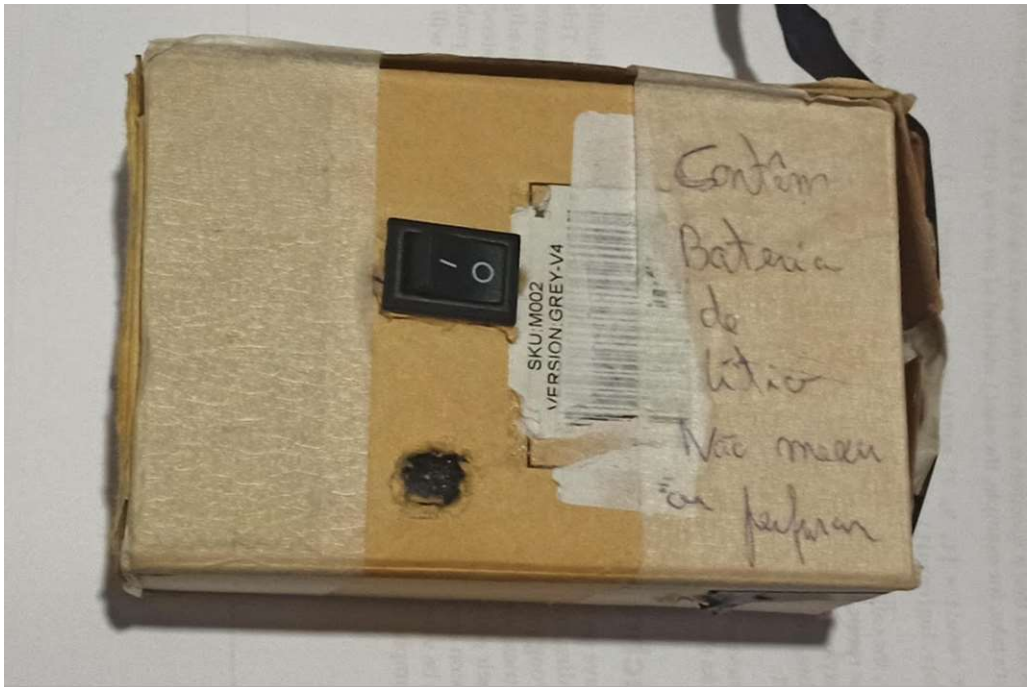


Figure 27 SensingBox external view

The cost for 8 SensingBox was 137,17€ this equates to about 17€ per box, coincidentally this is also the advertised price for the Arduino Nano 33 BLE[65]. Table 7 shows the price per component including shipping cost, most of the components were imported from China.

Item	Quantity	Price per Unit (€)	Shipping Cost (€)	Total Price (€)
LOLIN D32 board	8	5,56	4,24	48,72
DHT11	8	0,56	0	4,48
MAX4466 Sound Sensor	8	2,95	2,75	26,35
Dupont Wires	3 (40 pack)	4,76	0	14,28
Liitokala Lit-Ion Battery	8	3,23	4,10	29,94
Battery Holder 1 Cell	8	0,55	0	4,4
Switch	20 (1 Pack)	9	0	9
				137,17€

Table 7 Circuit cost per component

4.3.2 SENSINGBOX DEVELOPMENT

In order to begin development in the ESP32, it is necessary to install the ESP IDF-Tools following the guide provided by Espressif [66], the development environment chosen was Eclipse with the IDF plugin [67]. For debugging a JTAG board was connect to the ESP32 this provides a way to use the OpenOCD on chip debugger of the microcontroller.[68]

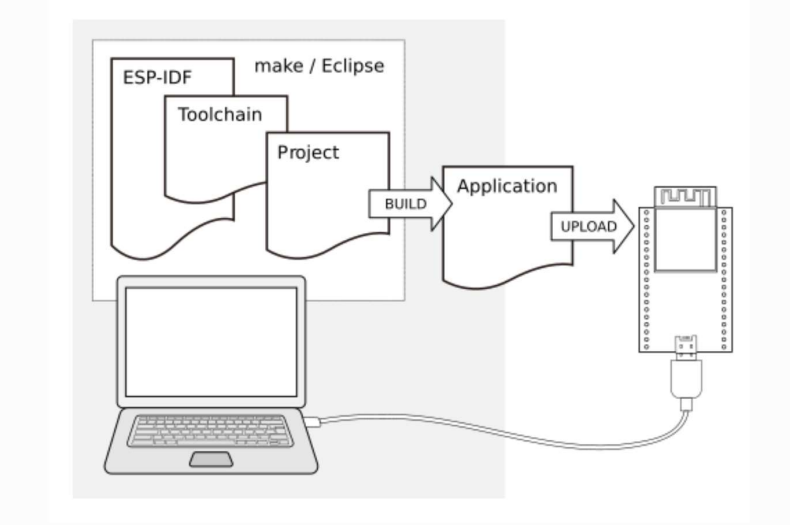


Figure 28 ESP-IDF Build Process

4.3.3 IDF COMPONENTS

ESP-IDF defines the concept of components (Figure 29), components can be seen as a sort of libraries with associated MakeFiles that establish the dependencies between components. A Kconfig file can also be present, this provides the possibility of configuring the components, and inputting such configuration when building the project. Another advantage of using component is increasing system modularity, a component declares in the CmakeList, what other components it needs to have as its dependencies.

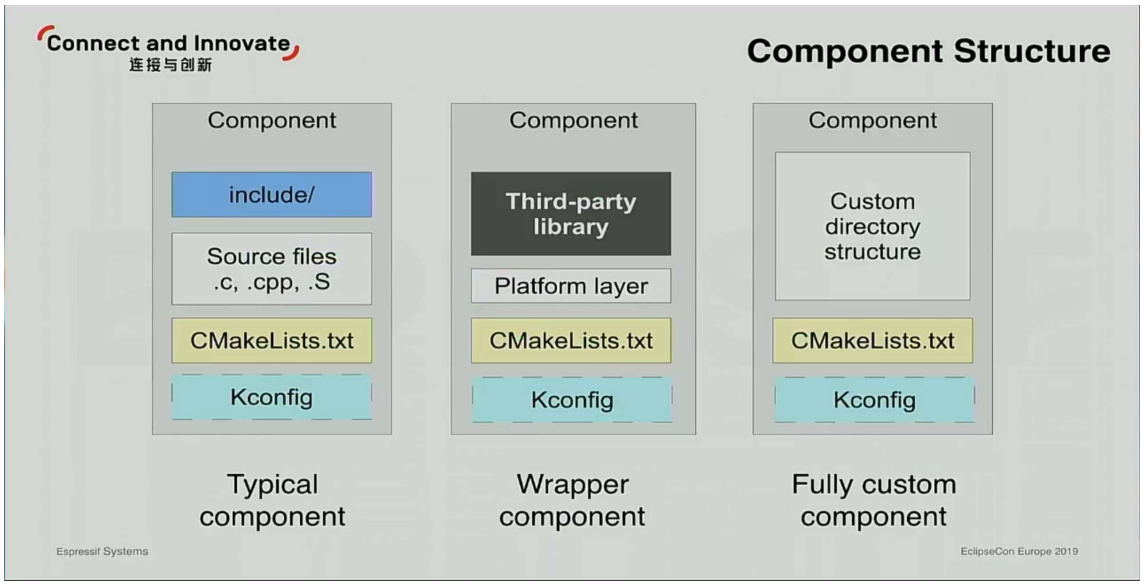


Figure 29 ESP-IDF typical configuration [69]

The Project is structured using components, each component can be configurable if necessary, using the *Kconfiglib*. For each necessary component a folder is created (Figure 30) with the CMakeList.txt designating the list of dependencies necessary for that component and a Kconfig file specifying the component configuration menu.

Figure 30 Project Structure

4.3.4 FREERTOS KEY CONCEPTS

Before going into more details about the implementation is important to refer some key aspect about FreeRTOS, the underlying kernel used by ESP-IDF, here the main concepts used throughout the project are explained. A more comprehensive explanation of these and other topics can be found in the book *Mastering the FreeRTOS Real Time Kernel a Hands on Tutorial Guide,* the FreeRTOS website[26], and the ESP-IDF Documentation, all of which were heavily consulted during the development stage of this project. The Espressif Forum[70] is also a very good source of information provided by the community.

4.3.4.1 TASKS

A FreeRTOS task is a sequence of instruction or "program" that can be executed independently from the rest of the system. Each FreeRTOS task has its own stack space. Tasks have priorities assigned to them. Based on the priority, the FreeRTOS scheduler chooses from the list of tasks available to run, a task to run. Figure 31 shows the 4 states that tasks can have.

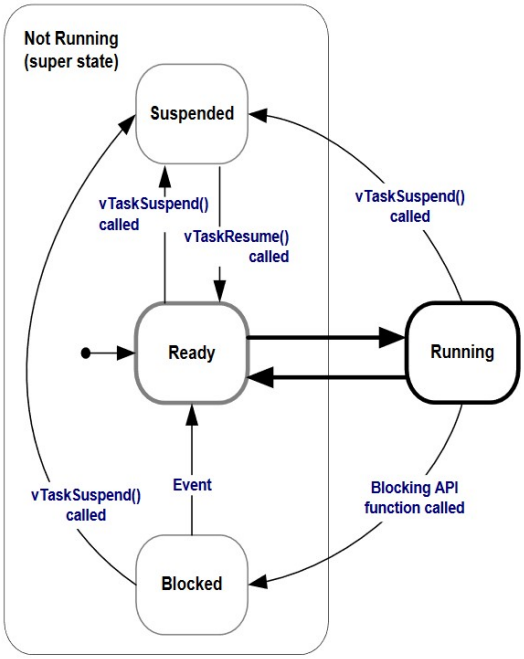


Figure 31 FreeRtos Task states

A task function should be implemented using an infinite loop and should never return. The most important commands used in a task implementation are the following:

- *xTaskCreate()* – Task Creation.
- *xTaskDelay()* – Time delay until a task is able to continue its execution.
- *xTaskDelete()* – Deletes the task.
- *xTaskSuspend()* – Sets the task in the Suspended State.

Figure 32 Task Creation Example

4.3.4.2 SYNCRONIZATION METHODS:

When developing a multithreaded system, task synchronization is a key aspect, it is necessary to assure that tasks do not access the same memory space (a global variable for example) at the same time (Critical Section), or that deadlock situations arise, from two or more concurrent threads blocking each other. These problems are solved by employing synchronization techniques, using mutexes or semaphores. While FreeRTOS supports both methods, it also builds upon them, offering more advanced synchronization features such as Event Groups and Task Notifications, which were used in this project. By using the FreeRTOS API to implement these functionalities, the development process is shortened and made clearer.

4.3.4.3 EVENT GROUPS:

- Event Groups provide the possibility for tasks to wait in the blocked state until an event or combination of events occurs.
- Event Groups unblock all the tasks that were waiting for the same event, or combination of events, when the event occurs.
- Event Groups can reduce RAM usage, by substituting several binary semaphores.

Figure 33 exhibits the working principle of event groups. A variable handle EventBits_t will hold the Event Register that contais the bits that can be set(1) or reset(0), when an event occurs in another set of tasks or interrupt. A bit or several from the Event Register is then set, or reset. The Task that was waiting for this event or combination of events can then leave the blocked state. It is possible to provide a timeout, a time for which the blocked task will wait for the combination of events, or no timeout, this means that if a certain event does not occur the task will be perpetually in the blocked state, or if a timeout is set, the task will be able to run after the timeout time even if no event has occurred.

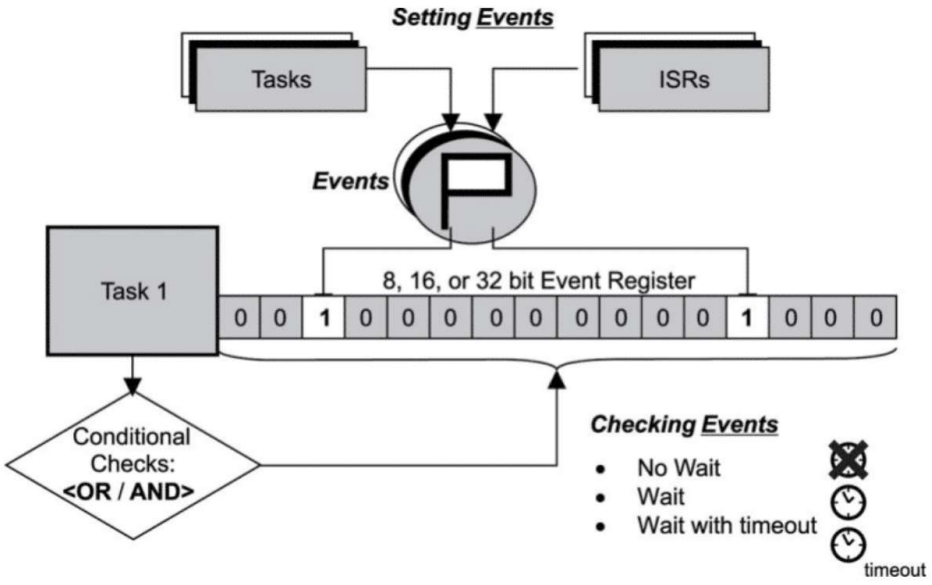


Figure 33 Event Group Functionality [71]

4.3.4.4 TASK NOTIFICATION:

As the name indicates, a task notification sends a notification to another task. The receiving task can be in a blocked state waiting for a notification, it is also possible to send other arguments along with the notification. The Task Notification API from FreeRTOS can use task notification as a substitute for several synchronization methods, such as: a binary semaphore, a counting semaphore, an event group and even in certain situations, a queue.

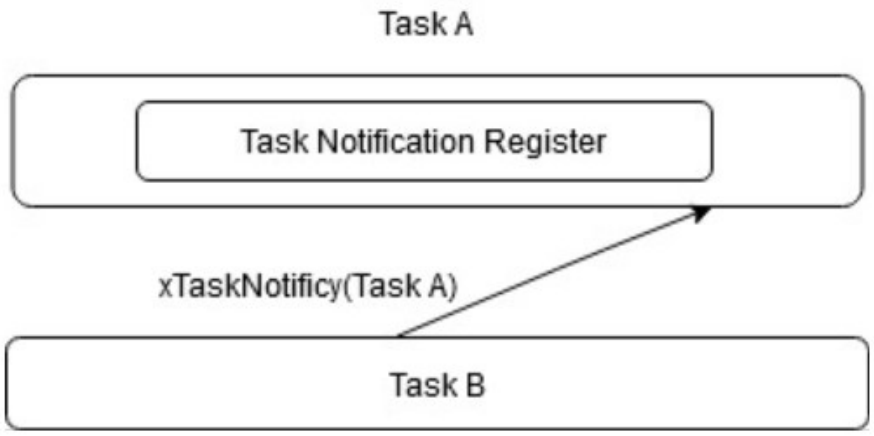


Figure 34 Task Notification Representation

4.3.5 SENSING BOX COMPONENT VIEW

The SensingBox software is divided into multiple components. Developing the system this way ensures loose coupling and increases component reutilization in other projects. Each component has its own set of responsibilities. The component developed are the following:

- **Wi-Fi Handler**: Responsible for Wi-Fi initialization and its associated working modes as Wi-Fi events such as disconnects.
- **MQTT Handler**: Handles publishing and receiving data from the MQTT broker.
- **Input Data Handler**: Handles incoming data, in this case from the MQTT handler, if other communication protocol was used, the workflow would be the same.
- **Configuration Handler**: Handles configuration related events, such as getting configuration from memory or saving new configuration.
- **Output Data Handler**: Handles response messages previous received from the Input Data Handler
- **Persistence Handler**: Responsible for sending data to the appropriate memory handler, NVS or SPIFFS.
- **NVS Handler**: Saving and recovering data from the NSV memory.
- **SPIFFS**: Writing or Reading from the SPIFFS File System

- **Data Collection**: General Component with each sensor data implementation.
- **Provision Handler**: Initially developed to support device provisioning using Bluetooth or Wi-Fi, currently not active, could potentially be a feature in the future.

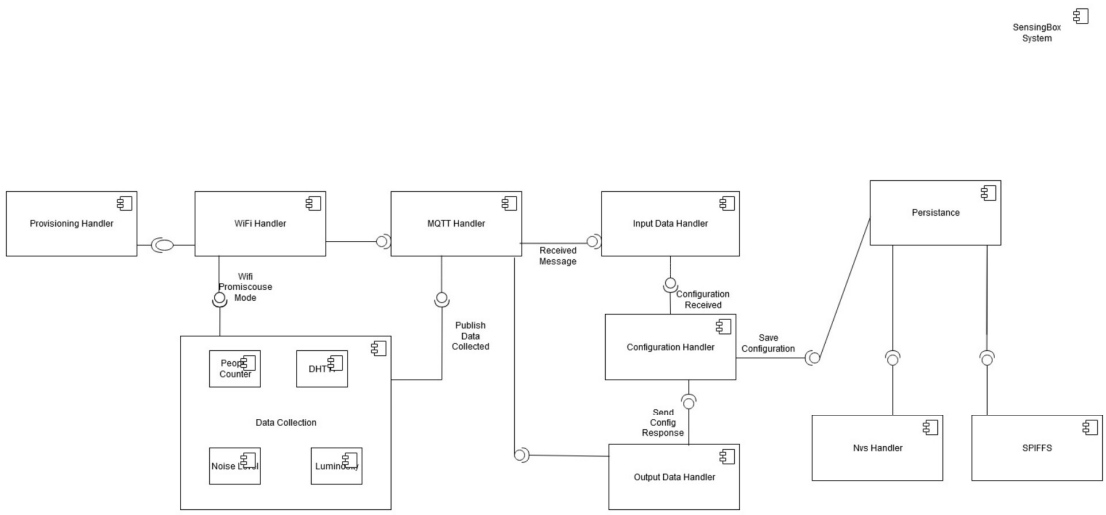


Figure 35 SensingBox Component View

4.3.6 TIME SYNCRONIZAITON

Keeping track of time is often a concern of a multitude of projects, especially in embedded development. SNTP is usually the default protocol chosen for this task, it permits the synchronization of the internal clock of the device with the time sent from an NTP server. Following a client/server structure, an UDP request is sent from the client to the server, the server then responds with the current time. (Figure 36)

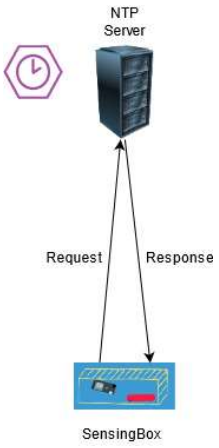


Figure 36 NTP Example

In this project time synchronization is not the most important requirement for the SensingBox, since the time of a data reading could be set, albeit with some delay on the

server, as soon as the server receives the data from the SensingBox. In projects where time synchronization is needed amongst different devices, or time is critical for the execution of the program, time synchronization gains and increased importance for these use cases. Espressif provides APIs to handle this case, that permit amongst other things, set the time for which the internal clock should synchronize with the server, i.e., what is the pool interval from the NTP server and how should the internal clock be updated, should it be set immediately set to the time received, or update time smoothly by gradually reducing time error. Here, the time is set immediately as it is received.

4.3.7 DATA GATHERING.

Each sensor has its own FreeRTOS task, in this implementation there are 3 tasks that belong to the data gathering function:

- Dht Sensor Task: Collecting Temperature and Humidity using DHT11 sensor.
- Sound Sensor Task: Estimating Noise Level using max4466 microphone.
- People Counter Task: Estimate Number of People using Wi-Fi probe capturing.

All the sensor use the same type of data structure for saving their data (Figure 37), this eases the process of sending the data to the server and possibly extending the system to support additional sensors. cJSON is used as the underlying data structure. All tasks must ensure that their cJSON structure contains the SensorName key with the name of the sensor, this is necessary to identify the topic of the message to be published. Figure 37 exhibits an example of the format implement on the data gathering tasks.

```
{"DeviceName":"SensingBox",
"SensorName":"DHT",
"Metric":"Temperature",
"Unit": "Celsius",
 "TimeStamp": "",
 "Data":[]}
```

Figure 37 Example of data gathering Format

The system uses a timer to dictate the general flow of the data gathering tasks. This timer is started immediately before the data gathering threads are created. These data gathering tasks check the timer when entering their loop. If the set time has been reached, the task will set its event group bit and enter the suspended state. While an interrupt-based approach could be more appropriate if time was a critical constraint for the project, this implementation trades precision for simplicity. For example, if the DHT11 reading task runs for a second more than it should, it is not critical.

4.3.7.1 TEMPERATURE AND HUMIDITY

The reading of temperature and humidity is straightforward. The only thing to have in consideration is the 1Hz sampling rate of the DHT11, for this reason is advisable to set the vTaskDelay() to at least a second.

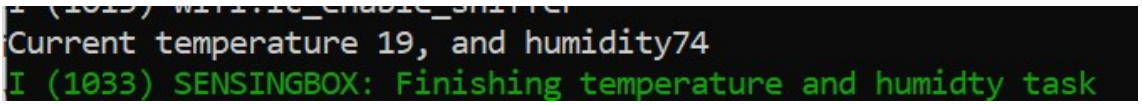


Figure 38 Temperature and Humidity Testing

To create a thread, *xTaskCreate* must be called and passed the task function (Figure 39), the name associated to the task, the stack size to be assigned to that task, additional parameters to pass to the task, the task priority, and an optional pointer to hold the task reference. Figure 39 depicts the creation of the sensor task. The sensor name *dht_sensor* is passed to the task, as the name of the function, then the minimum stack size defined in FreeRTOS multiplied by 3 to allocate enough stack size. For this task no additional argument is passed to the task, so is set to null, the priority is set to two which is the same for other sensor task, and finally since there is no need to hold a reference to this task so a null value is set.

```
xTaskCreate(dht_sensor, "dht_sensor", configMINIMAL_STACK_SIZE * 3, NULL, 2,
NULL);
```

Figure 39 Dht sensor task creation

4.3.7.2 NOISE LEVEL

To this implementation is necessary to use another communication protocol, the I2S.It is necessary to gather the samples from the ADC(24). To collect the data samples provided by the ADC, the CPU would have to be constantly pooling the ADC to get new samples, this could interfere with other tasks in the system or samples could be lost.

The IS2 protocol solves this, having DMA access to the SRAM of the ESP32. This way it is possible to save all the values sampled by the ADC in a buffer without using the CPU. The system follows and approach of storing 5 seconds of sound before processing. This value remains constant unless a smaller value is provided for the sound gathering time. For values that exceed this one, for example, if a 60 second interval is chosen, the system will process the sound collected every 2 seconds and average the result of all measurements, for 60 seconds this would mean the average of 30 sound intervals. Figure 40 shows the I2S configuration. The most important thing to notice is that the *I2S_MODE_ADC_BUILT_IN* must be active, the sample rate chosen was 2500, with 16 bits per sampling. Although it is not stated in the IDF-ESP documentation, the *dma_bug_len* is not measured in bytes, but in samples so the internal DMA buffer will hold 1024 samples. So for two seconds of sound collection almost 5 buffers of 1024 sample size would be necessary (1024 is the maximum allowed size). The *dma_buf_count*

was defined two 8 this way, when the task is reading from the DMA buffer, the ADC can continue to write to the other buffers that are not being read at the current moment.

```
1       i2s_config_t i2s_config =
2       { .mode = I2S_MODE_MASTER | I2S_MODE_RX | I2S_MODE_ADC_BUILT_IN,
3                       .sample_rate = 2500,
4                       .bits_per_sample = I2S_BITS_PER_SAMPLE_16BIT,
5                       .communication_format =
6                       I2S_COMM_FORMAT_STAND_I2S,
7                       .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
8                       .intr_alloc_flags = 0,
9                       .dma_buf_count = 8,
10                      .dma_buf_len = 1024,
11                      .use_apll =0,
12      };
```

Figure 40 I2S Configuration Setting

To calculate the amount of memory necessary to allocate and save the data, the formula is the following:

$$size_{bytes} = \left(buffer_{size} * bits_{sample} *\ \ number\ of\ buffers\right)/8$$

```
1       uint16_t *i2sReadBuffer = (uint16_t*) calloc((DMA_BUFF_SIZE_SAMPLE *
2       sizeof(uint16_t)*DMA_NUM_BUFF), sizeof(uint16_t));
3       ESP_ERROR_CHECK(i2s_adc_disable(I2S_NUM_0));
4       ESP_ERROR_CHECK(i2s_read(I2S_NUM_0, (void*)i2sReadBuffer,
5       (DMA_BUFF_SIZE_SAMPLE *     sizeof(uint16_t))*DMA_NUM_BUFF,
6       &i2s_bytes_write,portMAX_DELAY));
```

Figure 41 Memory Allocation and Reading from I2S DMA

Figure 41 display the allocation of memory according to the previous formula. At line 1 the memory buffer is allocated with a size of 16 bits for each DMA sample, and number of buffers. The DB level obtained by the SensingBox was compared to a decibel meter App installed on a smartphone and to a decibel meter acquired to the effect. While none of these two devices is certified for calibration purposed, the sound level detected was similar on all three.

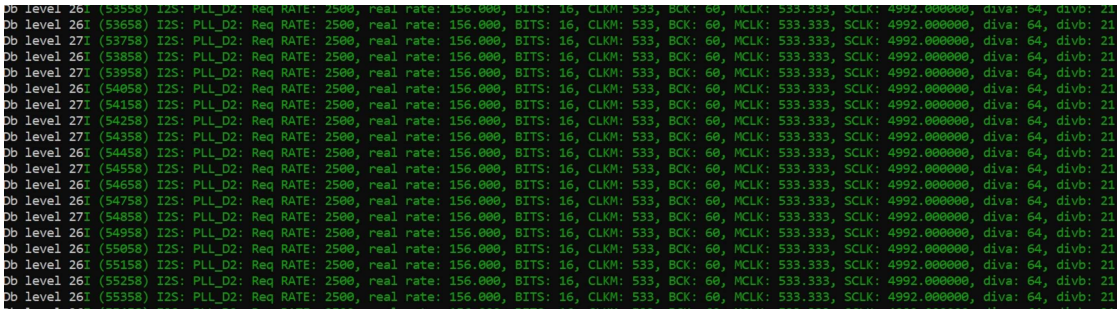


Figure 42 DB level testing

4.3.7.3 CAPTURING PROBE REQUESTS

To capture the necessary probe requests identified on the previous chapter, is necessary to set the ESP32 in Promiscuous mode. The data structures to receive the incoming packages are already defined but is necessary to provide additional functionalities on how to handle the packages and process them.

The first step is to define a callback function (Figure 43) that will be triggered when a probe is received, the objective here is to have a function that simply captures the packages and sends them to another task, using a Queue, so they can be further processed. The reason for this approach is because, having the callback spend too much time on processing will limit the number of probes that can be captured by the system, in addition to this, an IRAM _ATTR annotation is added in front of the functions, this will guarantee that this part of the code executable is stored in RAM, instead of flash, thus sacrificing RAM but guaranteeing a constant and faster execution time. In the ESP32 the RAM access speed is significantly faster than Flash, as with most systems.

```
void IRAM_ATTR promiscuous_packet_handler(void *promiscuous_buffer,
wifi_promiscuous_pkt_type_t type)
{

	static int debugCounterSent = 0;
	BaseType_t promiscuous_queue = 0;

	const wifi_promiscuous_pkt_t *promiscuous_package =
(wifi_promiscuous_pkt_t*) promiscuous_buffer;

	const wifi_ieee80211_mac_hdr_t *ipkt = (wifi_ieee80211_mac_hdr_t*)
promiscuous_package->payload;
	int sub_type = ntohs(ipkt->frame_ctrl);
	if ((sub_type & 0xFF00) == 0x4000)
	{
		promiscuous_queue =
xQueueSendToBack(packageCapturedQueue_wifi_handle, (void*
)&promiscuous_buffer, 0);
		debugCounterSent++;
		if (promiscuous_queue != 0)
		{
			ESP_LOGI(WIFI_TAG, "Package captured and sent to queue");
		}
		else
		{
			ESP_LOGI(WIFI_TAG, "Queue is Full!");
		}

	}

}
```

Figure 43 Probe Capture Function

One problem found with the implementation of the probe capturing feature was that the promiscuous callback function is faster that the probe processing task. To compensate for

this, it is necessary that even after the time set for the promiscuous, has been reached, to check if the queue has still packages waiting to be processed (Figure 44).

```
if (endTime &&uxQueueMessagesWaiting(packageCapturedQueue) == 0)
```

Figure 44 Check If there are packages still waiting to be processed

In Figure 45 the package captured test can be seen, where the first 3 captured MAC address correspond to randomized MAC-addresses, this happen because the smartphone still has not been connected to a network, after being connect the true MAC-address is revealed.

```
channel: 1, db:-92, mac_sender:     :e7:75:79:99,mac_rec: ff:ff:ff:ff:ff:ffI (3821) WIFI HANDLER: Package captured and sent to queue
channel: 1, db:-43, mac_sender:     :e7:75:79:99,mac_rec: ff:ff:ff:ff:ff:ffI (3871) WIFI HANDLER: Package captured and sent to queue
channel: 1, db:-44, mac_sender:     :e7:75:79:99,mac_rec: ff:ff:ff:ff:ff:ffI (3901) WIFI HANDLER: Package captured and sent to queue
channel: 1, db:-67, mac_sender:     :53:a3:cd:fc,mac_rec: ff:ff:ff:ff:ff:ffI (3921) WIFI HANDLER: Package captured and sent to queue
channel: 1, db:-67, mac_sender:     :53:a3:cd:fc,mac_rec: ff:ff:ff:ff:ff:ffI (23881) WIFI HANDLER: Package captured and sent to queue
```

Figure 45 IDF-console log with captured MAC address with First 2 Octets obscured

After being captured, the probes are sent in groups of 50 to the server (Figure 46). A Boolean flag is sent along the data packages indicating if still more packages groups are pending.

```
{message_id: 'SensingBoxGenesis0', sensor_name: 'Occupation', packages: Array(50), last_packages: false}
  last_packages: false
  message_id: 'SensingBoxGenesis0'
> packages: (50) ['11,-78,[redacted]45:1e:eb:9d,ff:ff:ff:ff:ff:ff', '11,-[redacted]78:b2:7b:0e:47,ff:ff:ff:ff:ff:ff'
  sensor_name: 'Occupation'
```

Figure 46 Probes Request sent to the server

4.3.8 MQTT-EVENTS

For the MQTT client implementation, the ESP-IDF MQTT client was used. This client follows an event-based approach where publishing or receiving events are sent to an event handler. The ESP-IDF provides the event loop library, this event library facilitates the development of event driven systems in a decoupled manner. Components can declare events for which other components register handlers, when a certain event occurs in one component a handler in another component can be triggered and process that event. This is what occurs with the ESP'S implementation of the MQTT client. External component can emit MQTT events that are handled by the MQTT event handler and are then published.

4.3.9 PUBLISHING QUEUE

The publishing of the sensor data collected follows a producer/consumer approach using a FreeRTOS queue. In this approach a set of producers, in this case a set of tasks sends

the data to a queue, this queue can contain a determined number of items. Figure 47 exemplifies the implemented behavior.

A producer reads the data from the Queue, in this case the MQTT Publishing Task and publishes the collected data to the MQTT broker. This queue mechanism was implemented using FreeRTOS queue API where the underling synchronization amongst producers and consumers is already implemented, so there is no need to employ mutexes when a task gets a hold of the queue. What is left for the programmer to define is the time the consumer/producer task should wait for data to arrive or space be made available if the queue is empty, or full respectively.

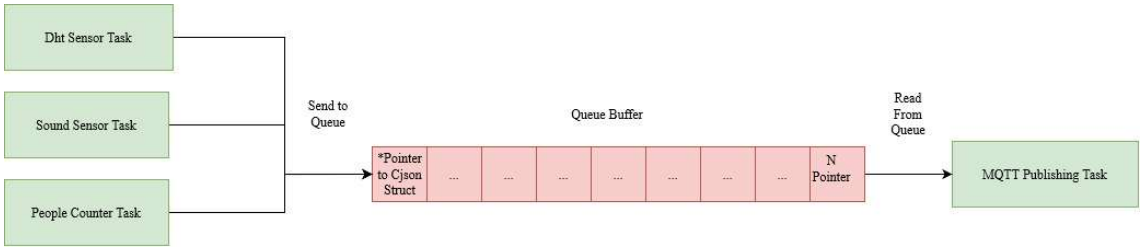


Figure 47 Publishing Queue FreeRtos

Figure 48 shows the MQTT publishing implementation. This task will only run after all the data gathering tasks have finished their data collection time, and after a successful connection to the MQTT Broker has been established, this corresponds to line 397 to Line 400 of the code.

At line 410 *xQueueReceive* tries to get data from the queue. If data is present, success flag will be returned. After this step on line 412 the sensor name is retrieved from the cJSON structure. The publishing topic is built based on the SensingBox name and the sensor name, on line 415 the data is published by the MQTT IDF client implementation. At line 419, after all the data has been published SYNC_GROUP_SLEEP_BIT is enabled, this signals the task handling the sleep cycles of the SensingBox to initiate the shutdown procedures preparing the SensingBox to enter sleep mode, thus a gathering cycle has been completed.

```
Publish Task,
Fetches sensor data from the Queue and publish to apropriate topic
*/
void mqttPublish(void *pvParamters) {

    EventBits_t xEventGroupValue; //Event Group Handle
    const EventBits_t xBitsToWaitFor = ( PEOPLE_COUNTER_EVENT_GROUP_BIT
            | SOUND_SENSOR_TASK_EVENT_BIT | DHT_SENSOR_TASK_EVENT_BIT
            | MQTT_START_FLAG);
    BaseType_t sensorQueueStatus; //Sensor Queue Handle
    cJSON *sensorDataToPublish;
    char *dataToPublish;
    char topic[128];
    char *sensorName;

    for (;;) {
        xEventGroupValue = xEventGroupWaitBits(sensorSyncEventGroup,xBitsToWaitFor,pdFALSE, pdTRUE, portMAX_DELAY);
        mqttStartStruct *myStruct = (mqttStartStruct*) pvParamters;
        sensorQueueStatus = xQueueReceive(sensorDataQueue,&(sensorDataToPublish), publishQueueFetchTime);
        if (sensorQueueStatus == pdPASS) {
            sensorName = cJSON_GetStringValue(cJSON_GetObjectItem(sensorDataToPublish, SENSOR_NAME_KEY));
            sprintf(topic, "%s%s\/%s", BOXNAME, sensorName,"reading");
            dataToPublish = cJSON_Print(sensorDataToPublish);//debug
            esp_mqtt_client_publish(mqtt_client, topic, dataToPublish,strlen(dataToPublish), 1, 0);
            cJSON_Delete(sensorDataToPublish);
        } else {
            ESP_LOGI(QUEUETAG,"QUEUE IS EMPTY ALL DATA SENT CURRENT WAIT TIME: %d",publishQueueFetchTime);
            xEventGroupSetBits(sensorSyncEventGroup,
            SYNC_GROUP_SLEEP_BIT);
        }
    }
    vTaskDelete(NULL);
}
```

Figure 48 FreeRtos Publishing Queue

4.3.10 SYSTEM SLEEP TASK.

To save energy, the ESP32 provides two main sleeping modes. Light Sleep and Deep Sleep. In light sleep most of the RAM, digital peripherals and CPU are clock gated. Clock Gating is a technique to dynamically reduce the power consumption of a device by disabling part of the circuitry so that the Flip Flops in them do not switch state. Another sleep mode is Modem Mode, similar to Light Sleep. In this mode the system can maintain a Wi-Fi connection at the expense of an increased power consumption. In Deep Sleep most of the system is turned off with the exception off ULP processor and the RTC and RTC peripherals, this makes it the most power saving mode, and the mode used in this implementation. The sleep cycle task will run after all the data has been published to the MQTT broker, it is a recommended practice to turn off the Wi-Fi functionalities before sleeping, also some event handlers are disabled.

4.3.11 WIRELESS FAULT RECOVERY MECHANISM.

Wireless connections are inherently less stable and more prone do disconnects that a physical connection, several factors such as interference from other signals, or the presence of walls or other obstacles account for this. Due to this it is necessary to have a mechanism that can compensate for a faulty connection, or loss of signal. The used mechanism for connection to Wi-Fi and sending data, uses a timer which can be defined by the user, if this timer is reached and internet connection is not establish the system will retry to reconnect to the network, if the reconnection fails, then it is necessary to save the data collected inside the ESP32 flash memory. An SD card could possibly be used to save the data, but when dealing with multiple devices, the cost of an SD card module and a SD card, increase not only the cost of the system, but also the power consumption of the overall system. Since the data to be stored typically is not large enough to warrant a bigger

capacity that an SD card could provide, the internal memory of the ESP32 is sufficient for this use case. ESP32-IDF enables the creation of a virtual file system called SPIFFS, in a partition of the device's flash memory. To enable this is necessary to create such partition in the Flash Memory. The LOLIN D32 has 4MB of memory built in, while by today's modern computational standards this is not much, it is sufficient to implement the desired functionalities, without resorting to external storage mediums. It is necessary to have in consideration that although the flash memory is 4MB, the RAM is limited to 512Kbits. For this reason, it is necessary to send the backup information in multiple small portions, allocating a small section of memory to save the data read from the SPIFFS, otherwise memory allocation errors will arise if the data read from the backup file is too large. (Figure 49). In certain versions of the ESP32 such as the ESP32-WROVER series it is possible to increase the amount of RAM available using integrated SPIRAM, albeit at this slower access speed. The LOLIND32 uses the ESP32-WROOM that only supports Flash Memory[72].

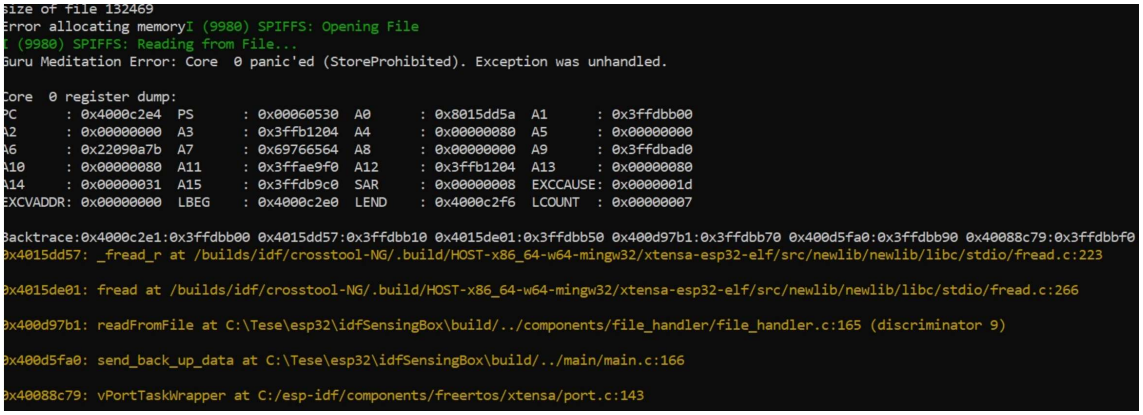


Figure 49 Memory Allocation Error reading from SPIFFS

A maximum file size for backup storing is defined. If this limit is reached, it will not be possible to save further data. Figure 50 exemplifies this case, a purposely small file size is defined in order to trigger this behavior, the four times the system tries to back up the data correspond to the collected data for Temperature, Humidity, Sound and Occupancy.

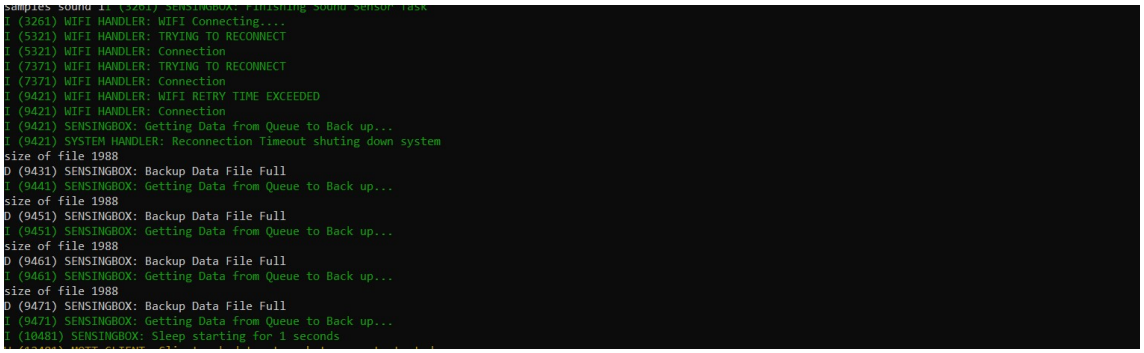


Figure 50 Small File size Defined

4.3.12 LOGGING

Another important aspect of any system is having a logging ability, the ESP-IDF has a logging library that permits the use of logging functionalities with different levels. The following verbosity levels exist:

- Error,
- Warning,
- Info,
- Debug,
- Verbose

This library was used extensively when developing this system. Each component has its own logging messages in figure Figure 51 several logging messages related to persistence (NVS and SPIFFS) can be seen, that can be display on the serial monitor.

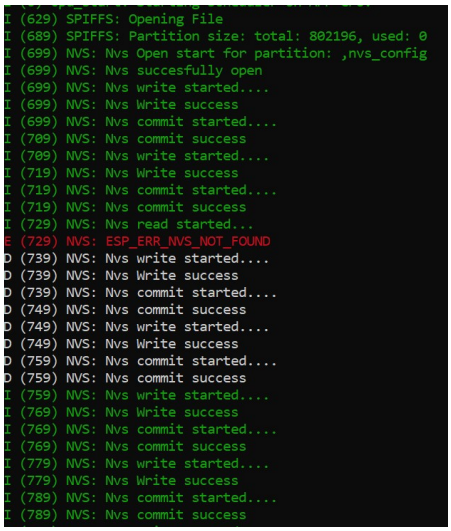


Figure 51 Different Types of Log Messages

As the SensingBox operates on battery and not connected to another device where these logs could be displayed or stored is necessary to write some logs to a file using the SPIFFS. Using the same approach as with the data backup, the error logs are written into a log file stored in the ESP32 flash and can be sent to the server.

```
# Name,   Type, SubType,  Offset,   Size,  Flags
nvs,      data, nvs,      0x9000,  0x4000
otadata,  data, ota,      0xd000,  0x2000
phy_init, data, phy,      0xf000,  0x1000
factory,  app,  factory,  0x10000, 2M
ota_0,    app,  ota_0,    ,         500000
nvs_config,  data, nvs, ,        0x3000
storage,  data, spiffs,  ,        0xF0000,
```

Figure 52 ESP32 Flash Partitions

To configure the ESP32 Memory is necessary to divide the Flash Memory into partitions, as shown in Figure 52. A default configuration is provided, but for this project it was a better option to have a defined set of partitions for each storage functionality needed but maintaining some of the partitions defined in the default configuration.

- **nvs:** default NVS storage that for example is used by Wi-Fi to store credentials other native IDF functionalities that require NVS.
- **otadata**: metadata about an OTA update,
- **phy_init**: is the bootloader when the system first starts,
- **factory**: is the space for the default code flashed into the device,
- **ota_0**: is the code for the Ota update,
- **nvs config**: is the defined nvs storage for the components developed for this project
- **storage:** SPIFFS file system that holds the back-up sensor measurements file and the logging file.

4.3.13 BATTERY READING

To read the battery voltage level one of the internal ADC pins can be used. The current LOLIND32 boards already have internal resistors so implementing a voltage divider by using for example two 100k resistors is not necessary, when using other board this might be a necessary step, since most development boards only accept 3.3v on the input pin and depending on the battery chosen to power the board this number could be exceeded. The battery data is collected on device startup and is then stored in memory. When a successful connection is established, the data is recovered from memory and sent to the server.

4.3.14 CONFIGURATION HANDLER.

Changing system configuration is a necessary aspect of this project, it might be necessary to change the network the devices is going to use, the number of samples gathered from the different sensors, or the sleep time between samples. The objective is to provide a way that a user might change the parameters remotely and having a system that is configurable and applicable to different environments.

A default configuration is provided, this configuration is saved in memory when uploading the software to the SensingBox, it includes parameters related to each sensor such as pooling time or other sensor specificities. Connectivity parameters such as MQTT broker address, Wi-Fi connection, are also stored in this memory, but cannot be changed remotely.

All SensingBox subscribe a MQTT topic with the following format **"sensingBoxName/config".** This way the broker will deliver the message to the appropriate SensingBox.

After a message is received by the MQTT handler it is sent to the input data handler which checks what type of message was received. If it is a configuration type message, then the message will be forward to the **Configuration Handler**. For a configuration to be valid, the configuration key sent, has to match the configuration key on the SensingBox memory. If the configuration is valid, the data to be saved is sent to the persistence handler. After the data has been saved, the **Output Data Handler** will send the response message to the **MQTT Handler** that will then publish the response message to the broker. Each configuration has its own message identifier.

The MQTT broker can remember the devices that previously established a session with the broker. After a configuration message has been published by the server to the broker, this message will be sent to the client, in this case the SensingBox as soon as the device connects to the broker after a sleep cycle.

4.4 WEB-SYSTEM

The "Web" portion of the system follows a layered architecture (Figure 53) more specifically a three-tier architecture with a distinct separation of concerns between business logic, presentation, and persistence. While Service-Oriented Architectures (SOA) have been popularized in recent years, especially through Micro Services, this type of architecture is better suited to large enterprise systems than small applications. A layered architecture provides enough separation of concerns in a system of this nature and makes the development easier to accomplish and easier to test.

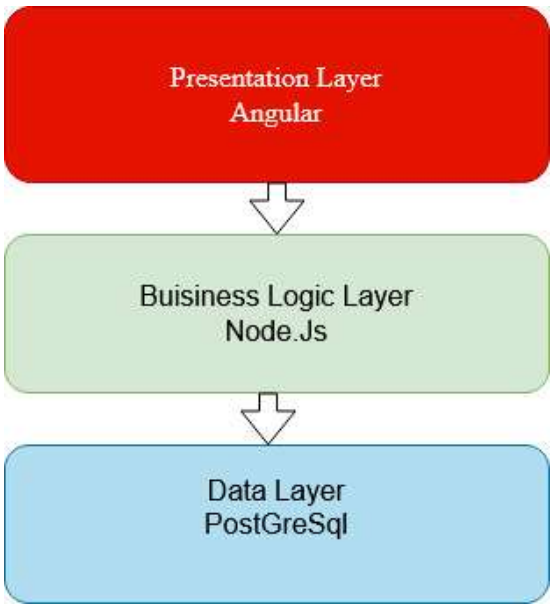


Figure 53 3 Tier Layered Architecture

4.4.1.1.1 EXPRESS SERVER

Express is a Node.js web application framework that provides a set of features useful for setting up a web server, this way simplifying the development process.

The server structure can be seen on Figure 54, it follows the following structure:

- **Routes**. The API Routes Endpoints, forward the received request to the appropriate controller.
- **Controllers.** Handle the received data from the routes, or from the MQTT broker.
- **Services** Service Layer responsible for business Logic functions and interaction with the models.
- **Models**. The functions that should interact with the database.

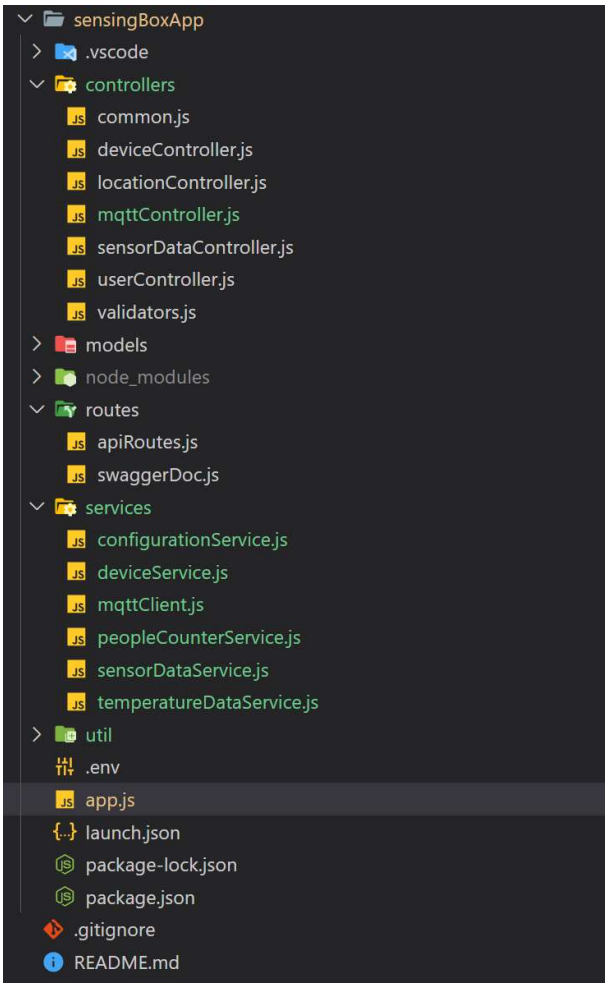


Figure 54 Back-End Server Structure

All the REST APIs are documented using Swagger, to perform API testing Postman was used.

4.4.1.1.2 PROCESSING MAC ADDRESS

Upon receiving the MAC addresses captured by the SensingBox, they are filtered according to their RSSI, those that are within range, are processed and classified according to their frequency. Figure 55 exemplifies this, the number of occurrences, refers to the number of distinct MAC addresses detected, each MAC address has its own number of occurrences. The total number of messages received corresponds to the global number of probe request captured by the SensingBox in one data gathering cycle.

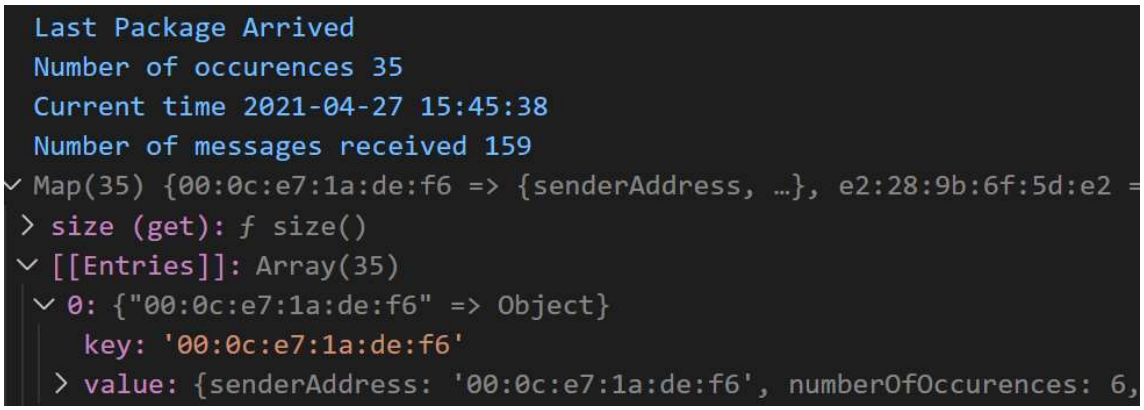


Figure 55 Probes Captured

While the objective is not to save the captured MAC addresses some additional information such as the channel with the greatest number of probes emitted, or the most common MAC addresses detected could be used to further improve the detection algorithm.

After this step, the server will save the number of unique MAC addresses occurrences as the “estimated” occupancy of a location.

4.4.2 RELATION DATABASE DATA MODEL

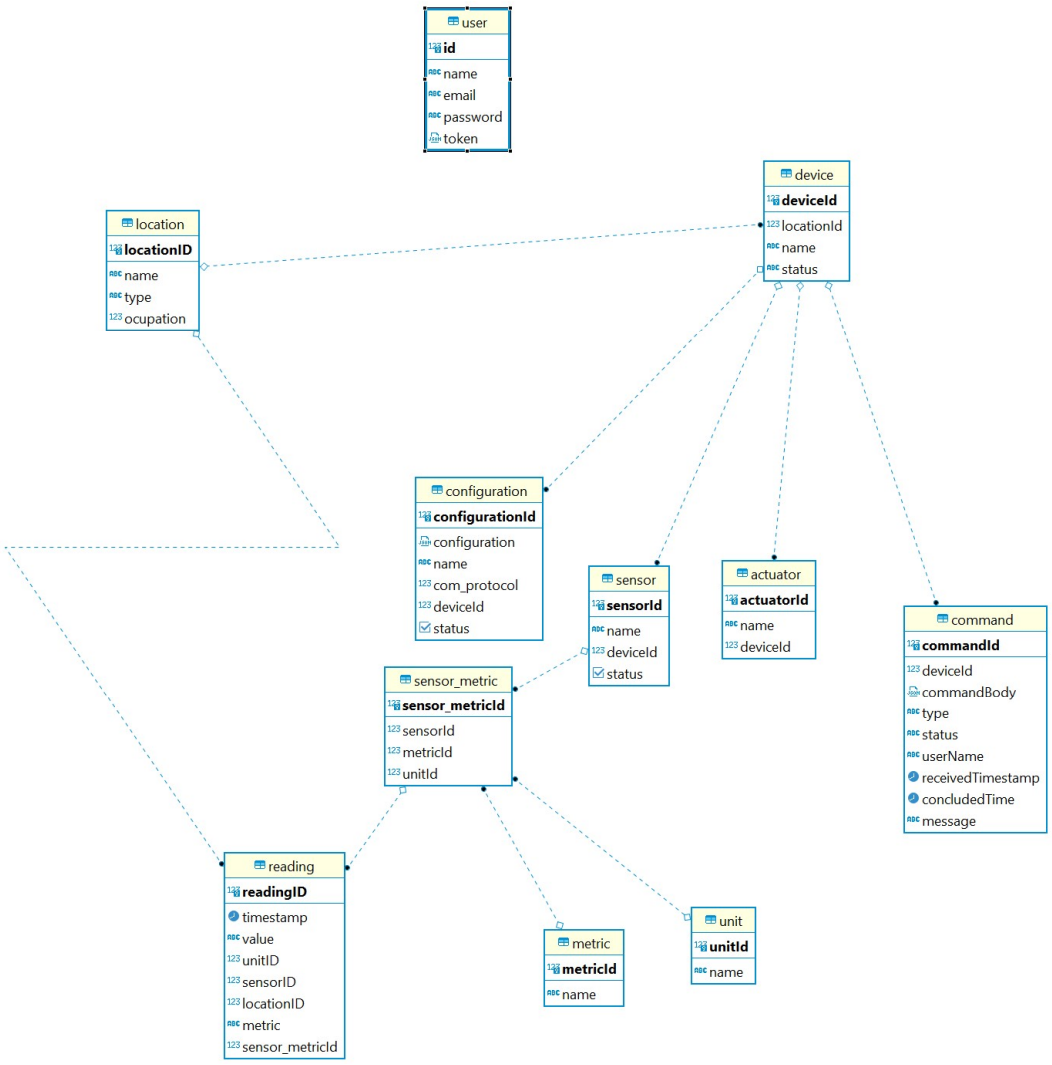


Figure 56 Relational Database Model

The Data Model (**Erro! A origem da referência não foi encontrada.**) model depicts the interaction of the different entities that composed the project. Although currently only SensingBox type of devices are present, other devices could be used. The data model was thought in a more general view of what an IoT system constituted by different IoT devices would need rather than be tied to the SensingBox characteristics, although each system and each device might have its own peculiarities that could require some changes to the data model in the future.

- The user entity represents a user than manages a site with different locations where different IoT devices are placed.
- Location: The location where a device is placed. A location can have multiple devices, but a device can only be associated to a location at a time.

- Device: An IoT devices, composed by sensors and actuators. A device might also have different configurations available with one being active. To interact with the device different commands can be emitted.
- Sensor: A sensor belongs to a device, but a device can have one or multiple sensors, a sensor has a metric such as Temperature, Humidity… a metric can have different units. For example, the temperature can be collected in Celsius or Fahrenheit. The combination of sensors and unit can belong to multiple sensors, but a sensor can only have an active combination at a time.
- Reading: The measurement made by a sensor, in a specific time and location.

4.4.3 ANGULAR DEVELOPMENT

The first step in the conception of the presentation layer, is doing a basic mockup (Figure 57) of how the page should look. The main idea is to have a page that displays information about the project, and then the last data collected by the sensors. Since one of the most interesting aspects, it is estimating the occupancy of the classroom, a small, animated video should be provided illustrating how this action works. Typically, after a low fidelity prototype, the next step is designing high-fidelity prototypes. Since the information here displayed is not very extensive and without a significant number of elements, the low fidelity prototype of the landing page was the only one made. With the other page having a similar layout and structure.

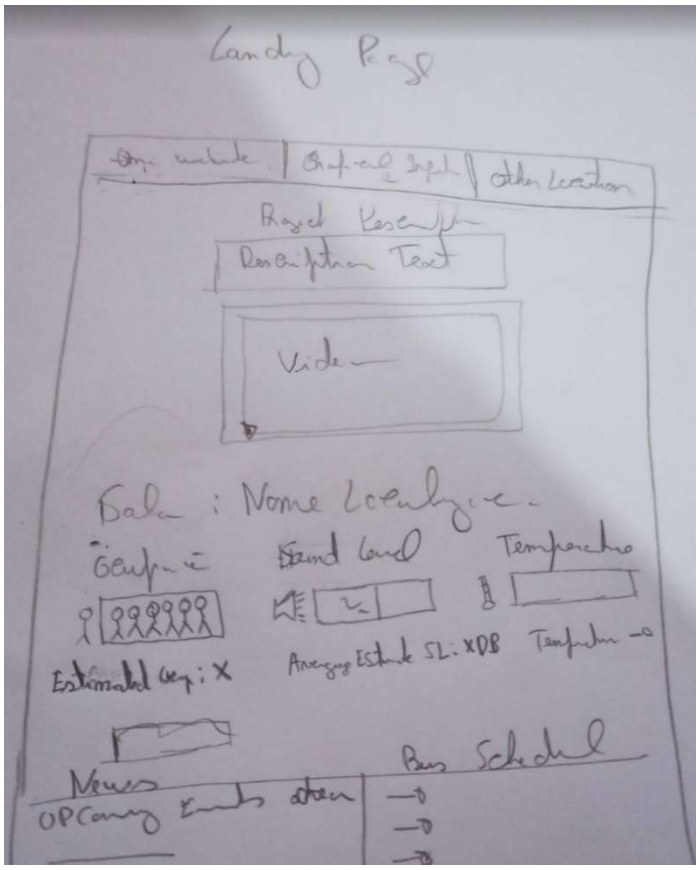


Figure 57 Low Fidelity Prototype

4.4.4 ANGULAR PROJECT STRUCTURE

Angular organizes its structures around components. Components define views which are parts of screen elements that can be changed. Angular is built on a block-like approach where each view element can have its own tag, thus on a single page different parts of the view can be injected and removed. In Figure 58, the tags seen represent parts of the view to be injected into the page. The router-outlet is a special type of tag used to load components dynamically based on the activated component.

```
<div class="flexbox-container">
<app-header></app-header>
<app-nav-bar></app-nav-bar>
<router-outlet></router-outlet>
<app-footer></app-footer>
</div>
```

Figure 58 Angular Main Component HTML

The main constituents of the projects are the components that support each view. Each component has its own folder. It is not mandatory to separate the CSS styles and HTML from the component Typescript file, both the CSS and HTML can be used inline in the component Typescript file such as shown in Figure 59. If the HTML code and CSS code is not very extensive (less than 3 lines is the recommended) this type of component definition can be used, for more extensive styles and templates a separate file approach should be chosen, and this is the method used for this project.

```
@Component({
  selector: 'app-root',
  template: `
    <h1>Tour of Heroes</h1>
    <app-hero-main [hero]="hero"></app-hero-main>
  `,
  styles: ['h1 { font-weight: normal; }']
})
export class HeroAppComponent {
/* . . . */
}
```

Figure 59 Component with inline template and Styles [73]

The Angular project structure can be seen in Figure 60. In the grey box the different folders that contain each component. The models folder contains the file of the different Interfaces that define the data models used in the project. Services contain the different types of services used by the system, such as HTTP request, authentication. Utils contain utility functions, such as sorting functions. Assets contain the images used throughout the web application. Environments correspond to the different application environments such as testing, production, or local. Locale holds the translation of the HTML content this way supporting internationalization.

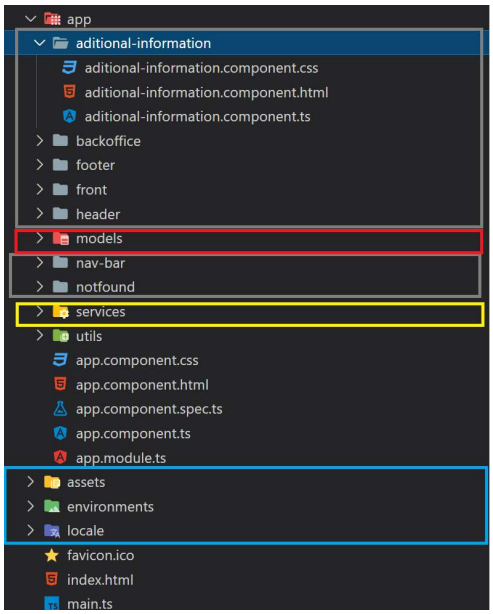


Figure 60 Angular Project Structure

4.4.5 VIEWS

To display the information gathered by the sensors, as the project information several views are implemented. The application is adaptable to multiple screen sizes, (responsiveness 92) The Views currently implemented are the following:

- Home Page.
- Location Selection View
- Location Page
- Additional Technical Information
- Login Page
- System Management
- Device Management
- 404 Page

4.4.5.1 HOME PAGE

In this page, a brief description of the project is accessible, as a short, animated video about this master's thesis. The animated video was conceived using PowToon[74], using this animation viewers with less technical knowledge can quickly learn about the project's main goals.

4.4.5.2 LOCATION SELECTION

In this view, (Figure 61) the overall information about each location is provided, this way a quick comparison between locations can be established. It is possible to display the locations ordered according to the different parameters that are collected by the SensingBoxes, by selecting the left-hand selector on the page. By doing this action the

locations will be ordered according to the parameter selected, in ascending order. Each location is clickable, this will transition the view to the specific location page.

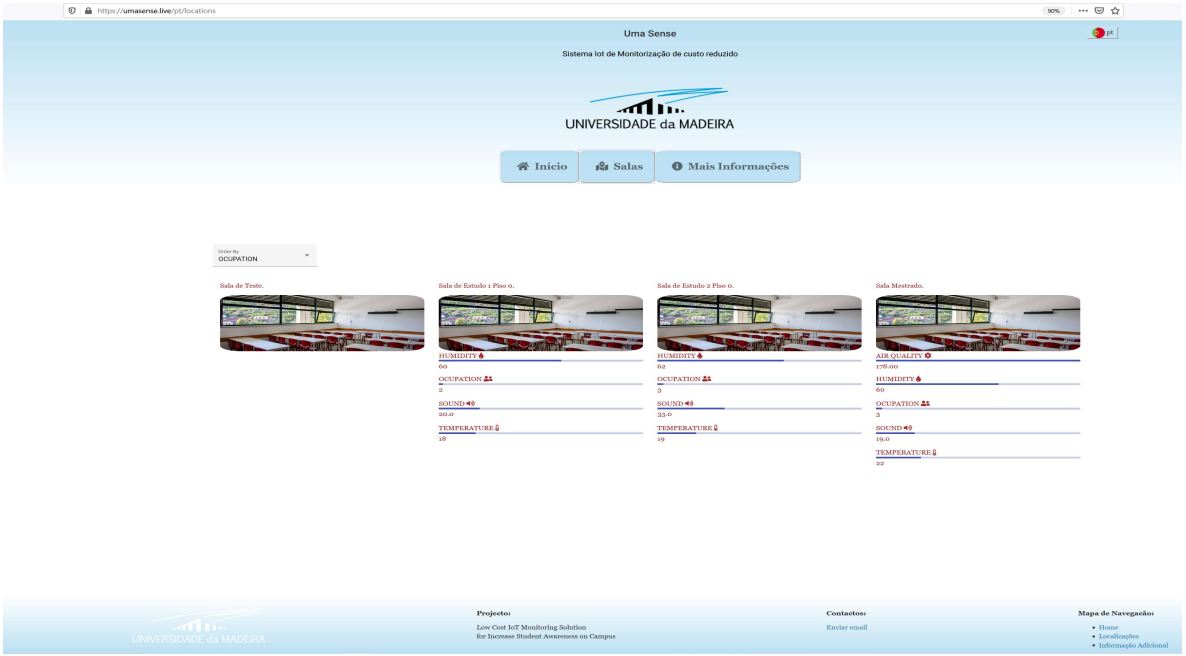


Figure 61 Location Selection

4.4.5.3 LOCATION PAGE

To consult the information collected by the SensingBox each location has an associated ID, by scanning a QR code in the SensingBox a student could follow a hyperlink that will provide the information of that specific room (Figure 62). In this page the last measurement collected from the sensors in that location are displayed, as a brief description of the project and the active sensors on the current location.

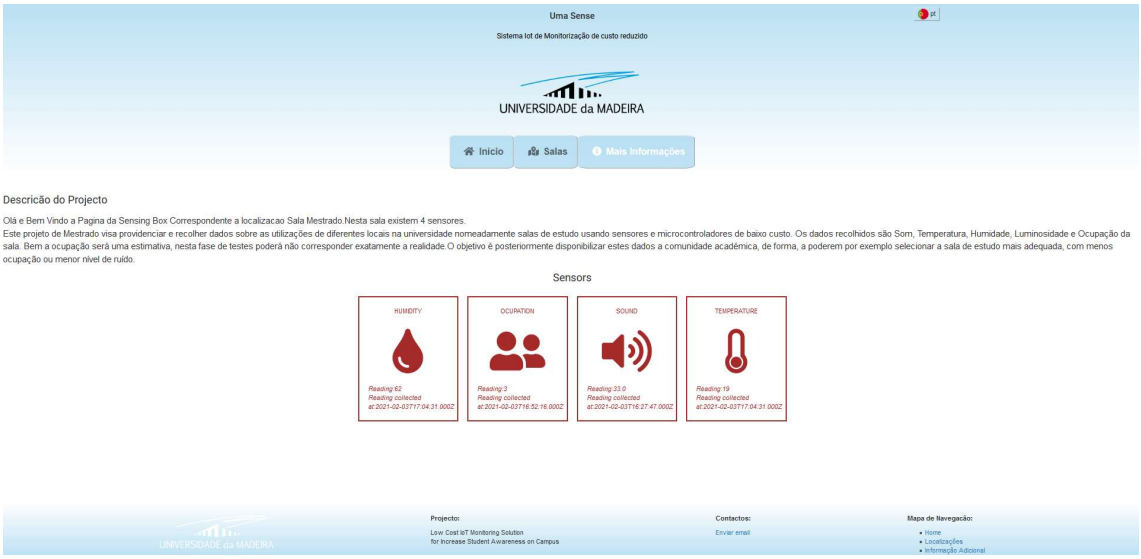


Figure 62 Location Page View

4.4.5.4 PROJECT INFORMATION

The project information page (Figure 63) is also provided. This page acts as a summary of this master thesis focusing on the overall view of the system and the technologies used. The objective is to provide a more technical overview of what was developed.

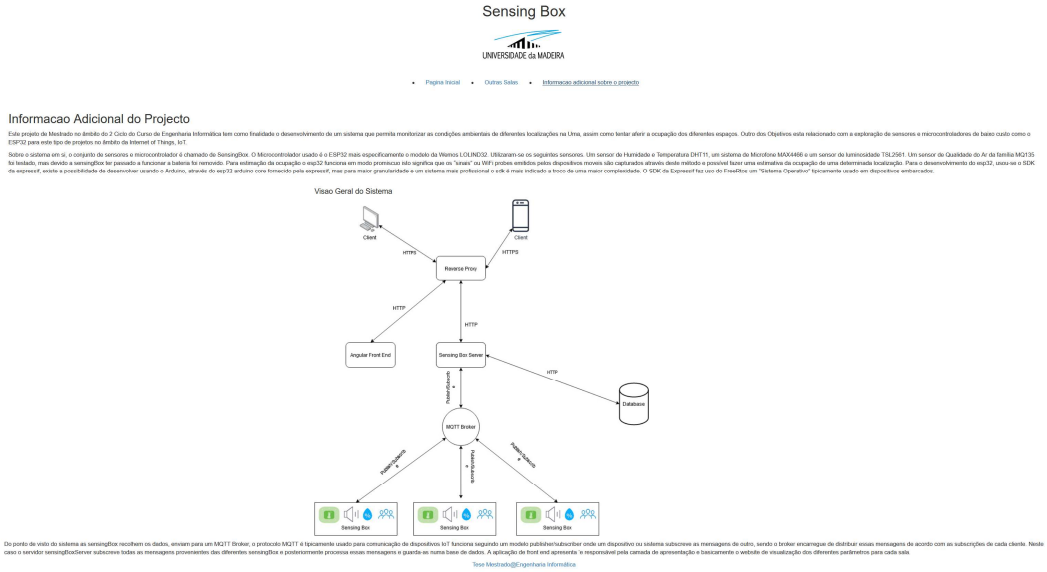


Figure 63 Project Information Page

4.4.5.5 BACKOFFICE VIEWS

To manage the different sensors, a Backoffice functionality was developed. This way it is possible to remotely change parameters of the SensingBox, add locations to the system, and assign SensingBoxes to locations. To use these features the user must be authenticated. To filter the access of management resources to only authenticated users, a token-based approach was implemented to limit access to certain resources, especially those in the management area of the web application.

4.4.5.6 AUTHENTICATION

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties.[75] JWT can be signed using a secret or a public/private key pair. Figure 64 depicts a resumed sequence diagram for authentication with the server. The user inserts its credentials; the server verifies if the credentials provided match with the credentials stored on a database, if the credentials are valid, a JWT token is generated and signed using the server's private key and sent back to the client. The JWT is stored in the browser's local storage. From this point on until the token expiration date is reached all management requests are sent with the JWT header.

To intercept the request made from the web client and sent to the server is necessary to add the token to outgoing requests, this is made using a service called HTTP Interceptor

that adds the token to all management requests, this ensures that these resources can only be accessed if the user has a valid token.

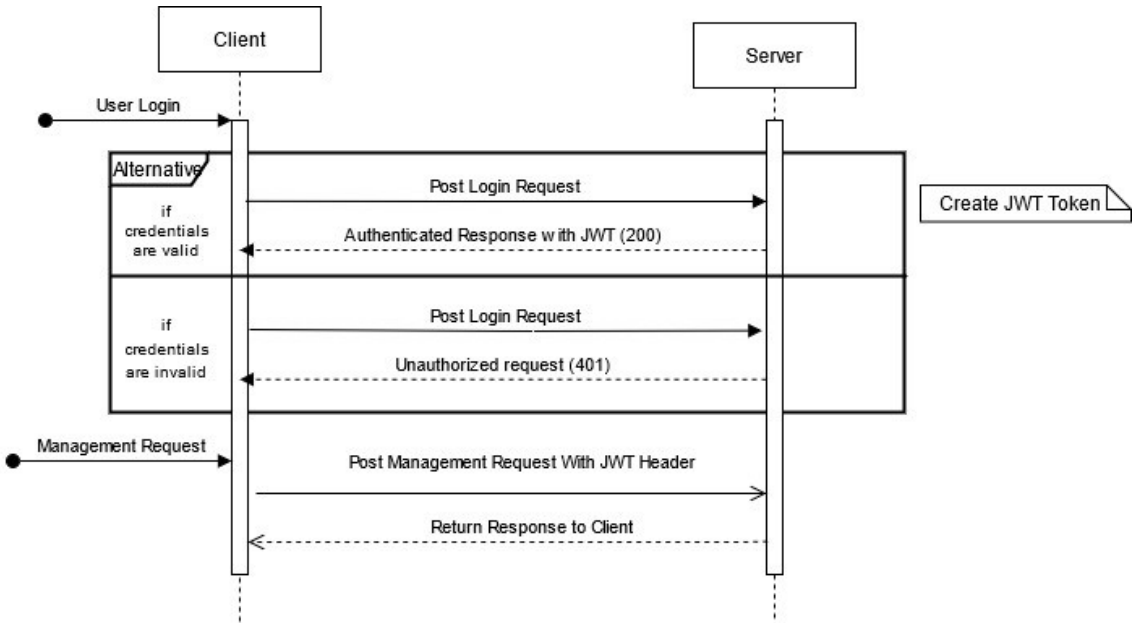


Figure 64 JWT Sequence Diagram

4.4.5.6.1 REAL TIME VIEW

To implement a real time view of the data sent by the SensingBox and processed by the server, a conventional Restful approach is not appropriate. REST requests are stateless and do not provide a permanent TCP connection. To solve this problem is necessary to implement a WebSocket. A WebSocket is a permanent TCP bi-directional connection between a client and a server. The backend server will be responsible to forward the messages received from the MQTT Broker to the WebSocket, this way messages in real time will be displayed in the management window (Figure 65). This is essentially useful for testing purposed to verify if the server is receiving the data from the SensingBox. The socket library used was Socket.Io [76].

Figure 65 Real Time View using WebSocket

4.4.6 INTERNATIONALIZATION

In Web development internationalization is the process of preparing and designing a web application to be usable in multiple locations which differ in languages. Localization is the process of building different version of an app for different locales, it consists in extracting text to translation and formatting dates, numbers for particular locations. Angular provides the localize packages to aid in this process. By marking the relevant html elements with the "i18n" tag and running the command *ng extract-i18n* a XLF file is generated, this file can then be sent to the appropriate translator or the translation can be done manually by adding the "<target>" markdown in each portion selected for translation. For this project text was manually translated since the objective was to just have an English version of the application, although this could be extended to support additional languages.

After the translation is done, the host web-server in this case Nginx will have to serve both version of the website, one which appends the "en" identifier to the site address so www.umasense.live/en and the other with the pt locale for the Portuguese version so www.umasense.live/pt. A selector was implemented to change the page location at the top right off the page.

Figure 66 Language Selection

4.5 DEPLOYMENT

A crucial step of any development is the distribution or deployment of the system developed. Modern technologies that gained popularity in recent years, such as containerized applications, CI/CD are discusses here.

4.5.1 HOSTING.

To host the Node.js server, an Amazon AWS EC2 instance was chosen. If necessary to change the hosting solution as the project is containerized, the migration from one hosting provider to other should be simple. Amazon AWS EC2 instances provide several functionalities regarding security, traffic monitoring and since this application is not expected to serve a substantial number of users, the free EC2 instance provided by amazon should suffice.

In commercial web application the deployment of an application can be done in several ways. On recent years, a Continuous Integration and Continuous Deployment (CI/CD) approach has been gaining popularity. Continuous Integration is a software development practice where members of a team integrate their work frequently, and usually each person integrates at least daily. Each integration is verified by an automated build. This automated build process usually includes tests, this approach has the advantage of detecting errors quickly and reduce integration problems[77].

Continuous Delivery is the practice of ensuring an application is always at a production ready state after successfully passing automated tests and quality checks.[58] The combination of these two approaches consists of a pipeline (Figure 67) where the previous step output is the input of the next step. This CI/CD has become industry standard when it comes to the deployment of applications, thus the increasing demand in recent years for DevOps engineers that ensure the maintenance and implementation of these practices by using tools such as Jenkins[78].

While the use of Docker does not constitute itself a form of CI or CD, it greatly simplifies the building and deployment process by eliminating restrictions of using tools, framework and testing suites, due to their lightweight and portable nature, amongst other advantages[59].

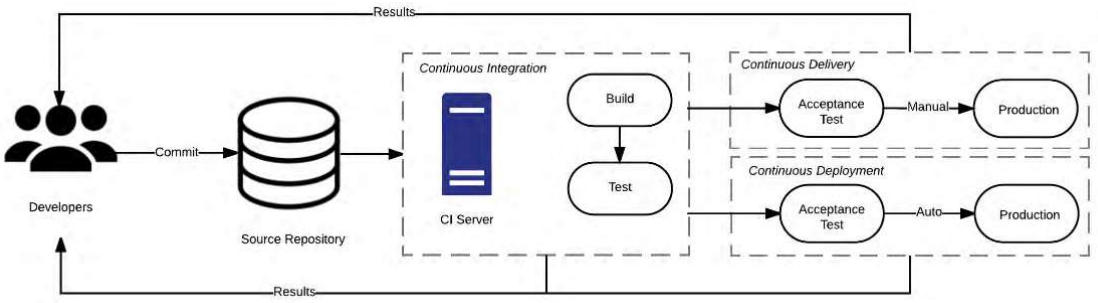


Figure 67 A typical CI/CI pipeline and the relationships established among Continuous Integration and Delivery process [79]

4.5.2 DOCKER.

Docker is essentially a light-weight virtual machine in the form of a container.[80] The main goal of Docker was to reduce the overhead necessary to run an application on top of a virtual machine. In modern system typically a server will host several virtual machines, these virtual machines can serve multiple applications, thus using less computational resources from host machines leads to an increase in the number of applications that such machine can support, this in turn results in cost reduction. Another advantage of using Docker is portability. An application that runs on Docker, runs on any operating system that supports Docker containers, thus the underlying operational system is irrelevant if the host machine runs Docker. These characteristics increase the portability of the system, it becomes much easier to migrate the system from one host platform to another if the system is containerized.

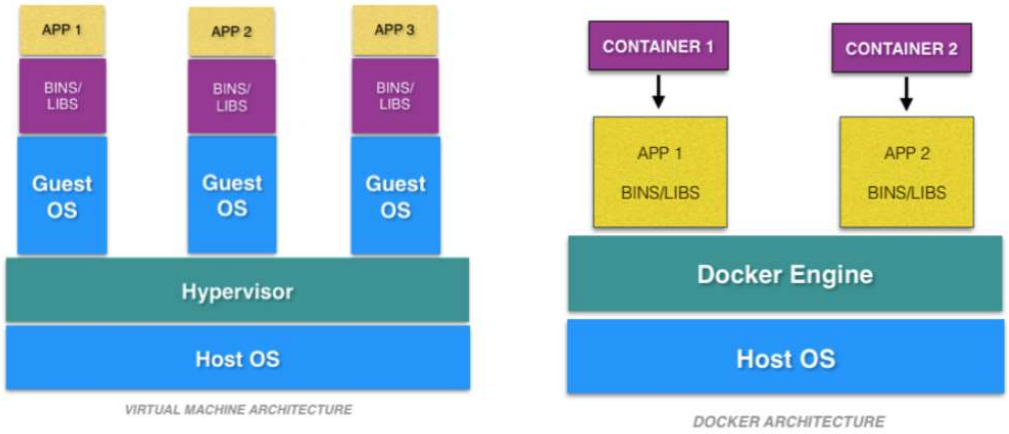


Figure 68 Docker Vs Virtual Machine[81]

4.5.2.1 SETTING UP THE CONTAINERS

To containerize our application, it is necessary to create our docker images. A docker image, in gross terms, consist in the metadata of an image and how to run it.[82] A docker image is the basic constituent of a docker container, images do not feature complete

operating system, they can use modules from the Docker host. There are several Docker images in Docker-hub [83] that can be used as the basis of a container.

To set the containers a file containing the instructions on how to build the image, a DockerFile is used. To further manage containers and associations between containers as facilitating the startup process of our system, docker-compose CLI is used. Docker-compose reads a *yaml* file that can have the relationships amongst containers, environmental variables to pass to the container, amongst other parameters. It also facilitates the startup of the system transforming several commands and configuration, in a one line startup command. An example of the Node.js image can be found in

4.5.2.2 NECESSARY IMAGES.

To build the containers the following images are necessary:

- Nginx: Nginx will be used as our reverse proxy.
- Node.js: Node.js application to the back-end server
- Mosquito: MQTT broker that receives the data from the Sensing Boxes and sends them to the Node.js server.

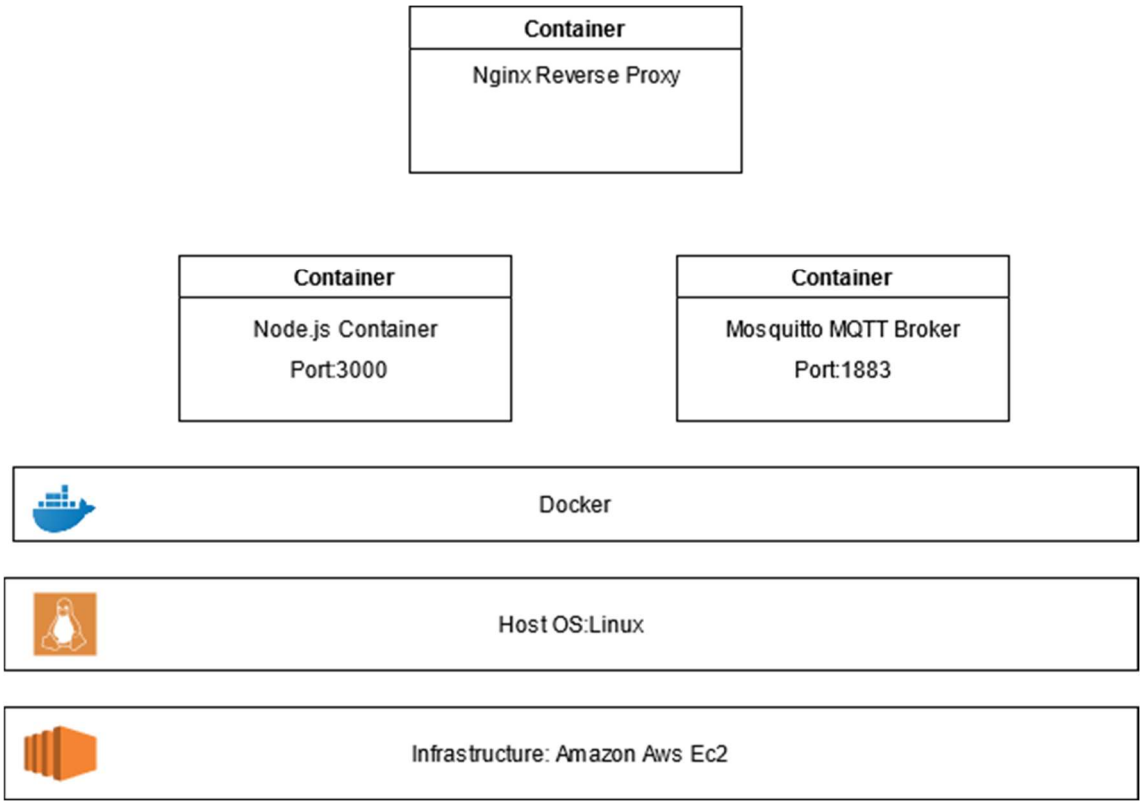


Figure 69 Docker infrastructure

The traffic received by the NGINX proxy is diverted to the correct host so a request to *www.umasense.live/api* will redirect the request to the Node.js server. If a request is made in the front-end application and this request in turn needs to access the back-end server, the communication between the front-end application and the back-end application is done inside the Docker network. Figure 69 shows the general container infrastructure.

4.5.3 SECURITY.

Besides the reverse proxy, which provides the possibility of hiding our internal system topology by having only one access point to the external network, SSL certification is also implemented. A SSL certificate was acquired without cost from *www.name.com* through GitHub for student packages discount, the domain *umasense.live* was also registered without cost. AWS also provide Security Groups features that can serve as a sort of virtual firewall allowing the control of Inbound and Outbound traffic, in this project there are two ports that can be accessed from external networks. One is the entry point for the Nginx Reverse Proxy, the other is the MQTT broker connection in port 1883. It is possible to restrict the inbound access of these ports to only certain IPs (Figure 70) such as the IP address of University of Madeira for instance, limiting the direct access to the MQTT broker.

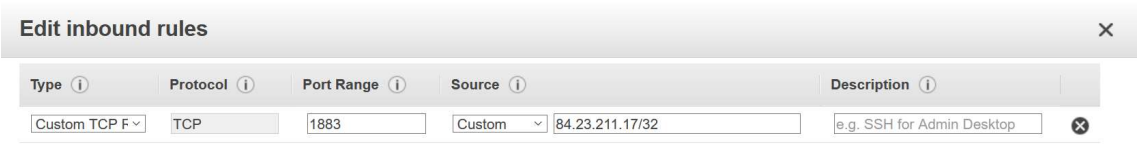


Figure 70 Inbout Rules for Connecting to the MQTT Broker

5 TESTING AND RESULTS

In this first set of preliminary tests the main objective is to observe how the system behaves regarding probe request sniffing and battery duration. How different RSSI filtering options applied to the collected probes might yield different results. Static single Wi-Fi channel targeting and Wi-Fi channel hopping strategies are also tested.

Most of the tests conducted used a data collection portion of two minutes. This is in line with a previous study [84], where within a two minute timeframes 11 out of 12 devices emitted a probe request at least once. In another experience, Hande et al. [85] state that typically 88% of devices will emit probes within 5 minutes.

5.1 TEST CASE 1: UNIVERSITY

This test case is very similar to what a "production" environment would be for this project since its goal is to be used at the University. Here several differences should be remarked when comparing to a previous test case conducted in a residential property to test the system. The Wi-Fi network is an enterprise network, most of the time the SensingBoxes are not being directly monitored. The SensingBoxes can be moved unintentionally or intentionally by members of the academic community.

The SensingBoxes were positioned in three locations at the university campus. Two were placed in distinct study rooms on the ground floor, for one week. The other SensingBox was put in a study room exclusive for Informatics Engineering graduate students on the second floor.(Figure 71) This test case was used to estimate how long would the Sensing Box would be able to run on battery and test the overall system functionality.

Sleep time was set to 10 minutes and data gathering time to 2 minutes. Due to the Corona virus restrictions, the movement of students and staff was almost nonexistent.

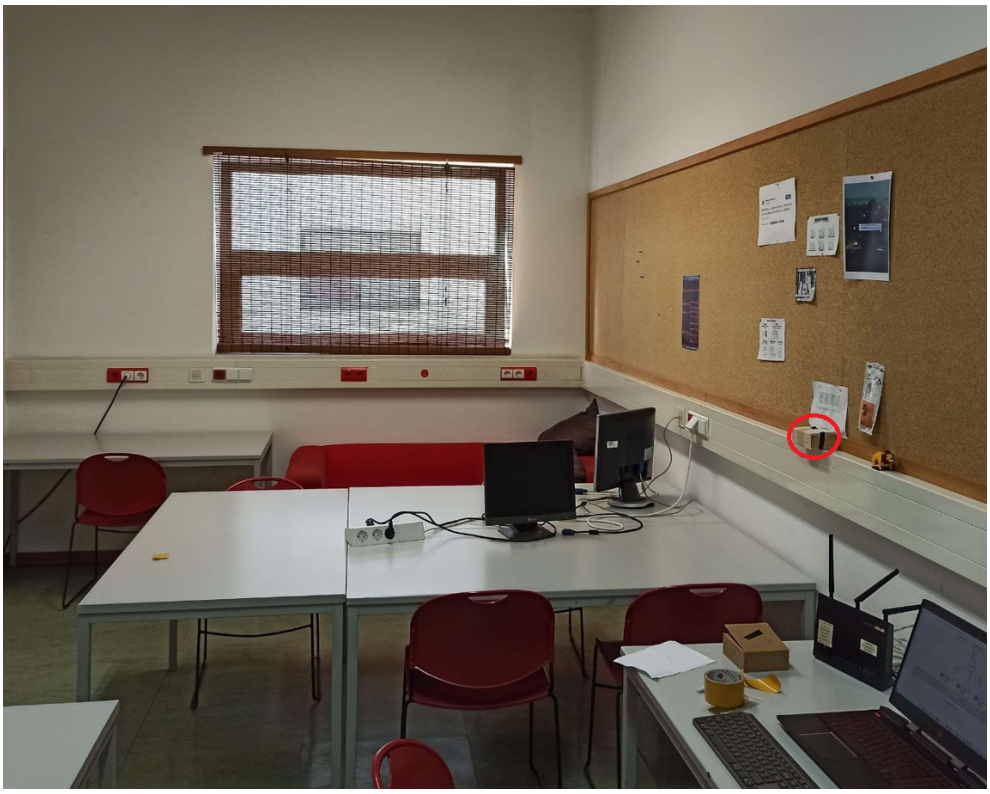


Figure 71 Test case 2 location: Study Room with a sensingBox in the right of the frame.

The SensingBoxes were able to run for seven days on battery power only, with a sleep time of 10 minutes and a data gathering time of 120 seconds. It was observed that as the supplied current from the Lithium-Ion battery drops below 3.3V, the readings of the ADC become erratic, so the sound level measurement becomes skewed, displaying values inferior to what a normal reading would be.

5.2 TESTE CASE 2: OPERATING SYSTEM EXAM

For the operating system course an exam was conducted in an open corridor due to COVID-19 restrictions, thus in an open space subjected to interference from multiple sources such as signals from upper floors or other nearby classrooms. At the start of the exams there were 27 students, plus two professors, and the test observer.

The SensingBox had a data gathering time of 2 minutes, and a sleep time of 10 minutes. The collected probes were subjected to an RSSI filter which disregards any probes with a RSSI inferior to -80db.

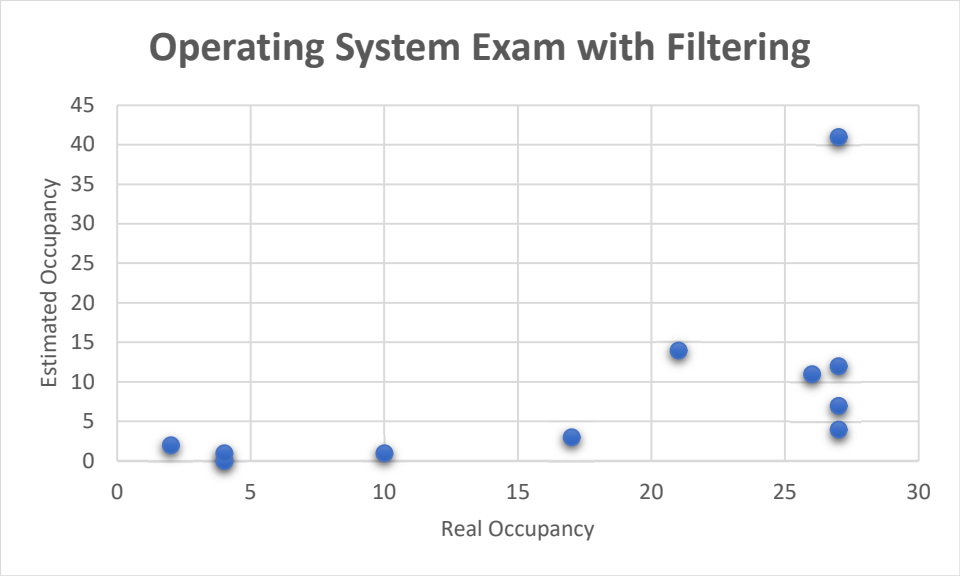


Figure 72 Operating System Exam with Filtering Applied

Figure 72, exhibits the graph comparing the real occupancy comparing to the estimated occupancy. Applying the RSSI filter yielded a Pearson's correlation coefficient of r = 0.56.

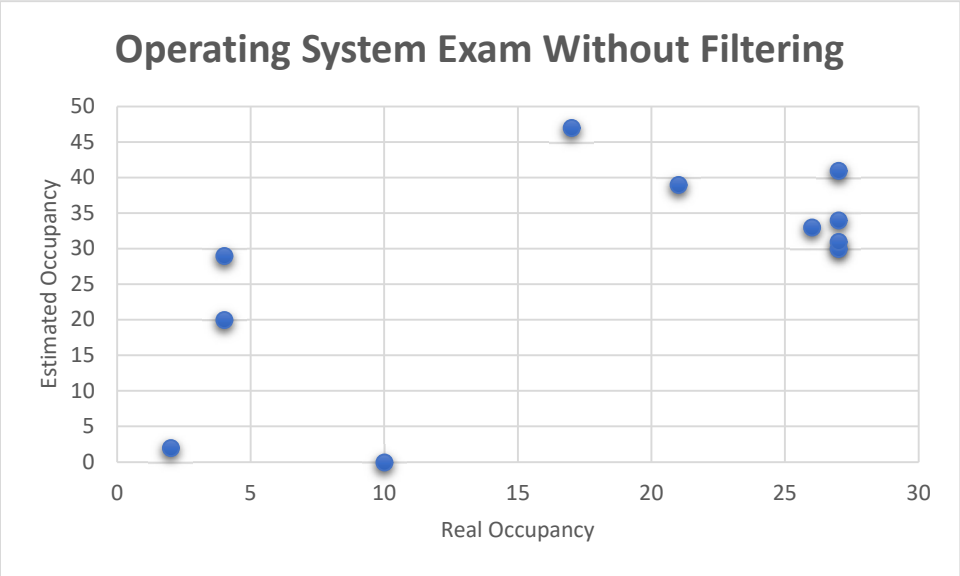


Figure 73 Dispersion graph real occupancy vs estimated

Without RSSI Filtering the correlation coefficient found is higher r = 0.68 but with an increased error in each measurement in relation to ground truth, this can be seen in (Figure 73), as for example, a real occupation of 10, was estimated to be 0.

5.3 TEST CASE 3: DISTRIBUTED SYSTEM EXAM

This experiment was conducted during the Distributed System exam, in a closed classroom with an initial occupancy of 9 people, 7 students, the professor, and the experiment observer.

Two SensingBox were used, both with a sleep time of 10 seconds and a data gathering portion of 2 minutes. The first SensingBox was positioned in the middle of the classroom. The RSSI filter used was -70db as this was a more enclosed environment when comparing with the previous Operating System exam. In one of the SensingBox the standard multi-channel switching approach was used, in the other a single channel was targeted.

5.3.1 SENSINGBOX MULTI-CHANNEL SWITCHING

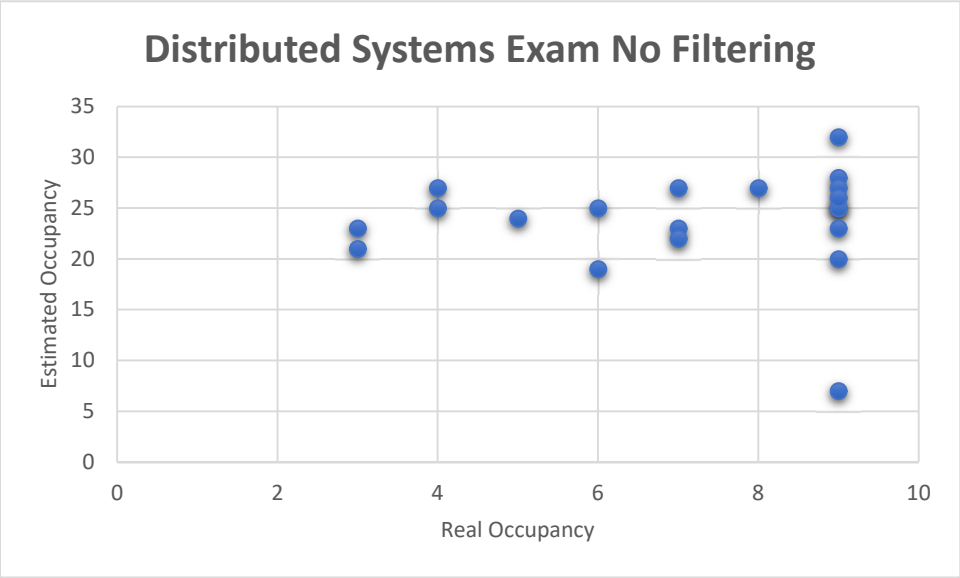


Figure 74 Distributed System Exam scatter plot without RSSI Filtering

Without filtering options no significant correlation coefficient r = 0,0249 was found. This lack of correlation becomes apparent by observing Figure 74 where, for the same ground truth, the estimated occupancy can vary substantially. This may indicate that the outside environment interferes too much with the signals captured inside the classroom and probes from adjacent rooms were being captured.

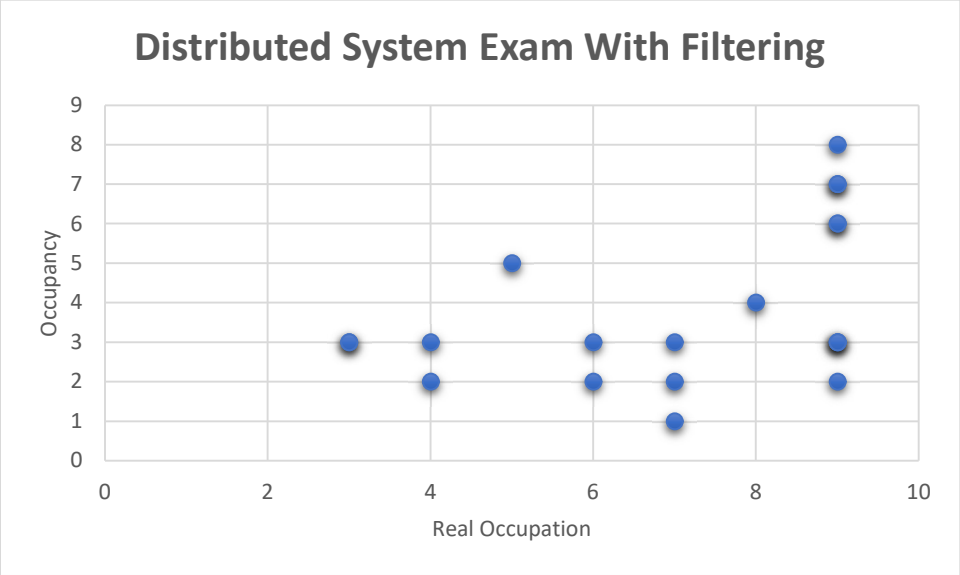


Figure 75 Distributed System exam scatter plot with filtering.

The correlation coefficient observed with filtering between the real occupancy and estimated occupancy was r = 0.43, this implies a medium correlation. Figure 74 exhibits the data points collected for this test case.

5.3.2 SENSINGBOX SINGLE CHANNEL

The other SensingBox in this test was placed in a central position but closer to the back of the classroom. Only channel 1 was targeted, as this is the channel used by the nearest access point. The filtering options were the same as with the multi-channel Sensing Box. Both correlations obtained were inferior to the multi-channel test case. Figure 76 and Figure 77 display the collected data points. No significant correlations could be established using both approaches although the filtering option still displayed a better value with a coefficient of r =0.2349, without filtering the coefficient is negative r= -0,09274.

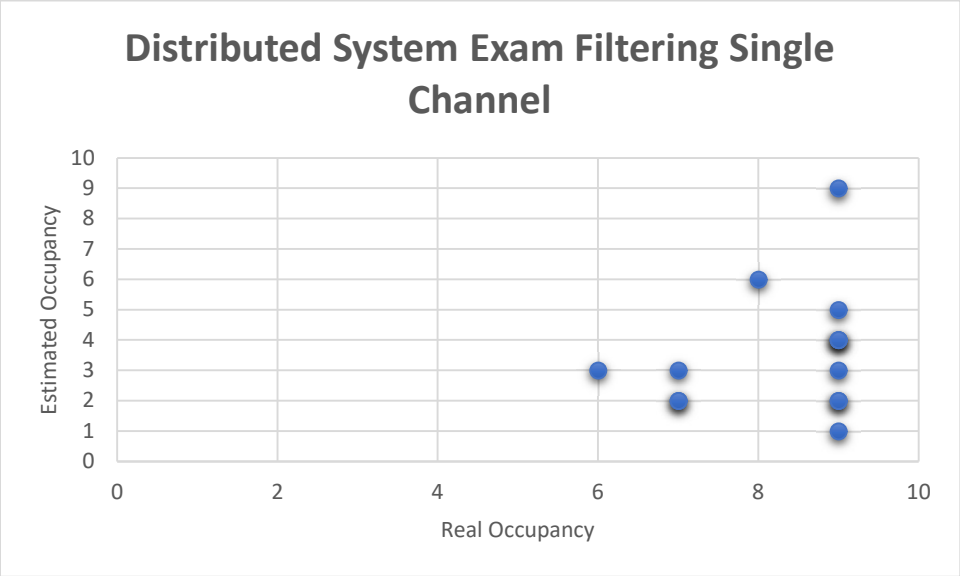


Figure 76 Distributed System exam single channel scatter plot.

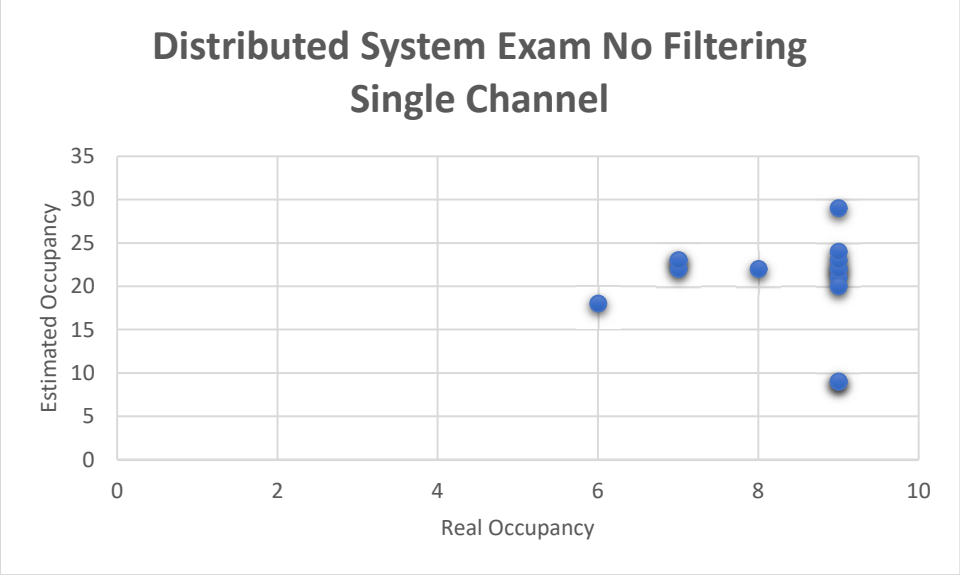


Figure 77 Distributed System exam scatter plot, no filter, single channel.

5.4 FIRST SET OF TEST ANALYSIS

While previous experiments [19], [86] consider the physical location of where the Wi-Fi tracking occurs (indoor or outdoor), another aspect that might be important to notice is the type of activities conducted at the locations. In Test Case 2 and Test Case 3 both were conducted in an environment in which its occupants cannot make use of their mobile devices, and some probably have the device with Wi-Fi capabilities turned off. As stated in previous studies, and observed in Test Case 2, if a device is not being actively used it might take longer to emit a probe request, this in turn could influence the occupancy estimation depending on the type of algorithm or approach used. Perhaps classifying the data not only according to the type of environment (indoor or outdoor) but also what the location is used for, might yield more accurate results.

Regarding Test Case 3, where multi-channel hopping was compared to single channel it was observed that the number of probes captured using the single channel was 20.1% inferior to the multi-channel approach.

The average probe request captured per data gathering cycle was 89 probes, with a total of 1780 probes captured for the multi-channel test, and 70 per cycle and 1400 total probes for the single channel test. Julien Freudiger [86] realized a similar test comparing different probe request capturing strategies. The study showed that when using only one single Wi-Fi network interface listening on a single channel (1) the number of probes captured was the highest. Dynamic channel switching between non overlapping channels (1,6,11) was second, the author suggests that this result might be attributed to the overlapping of channel 1 with channels 2 to 5, therefore a device emitting probes on one of these channels could have its probes also captured even if the Wi-Fi Interface is set to channel 1. As inductively assumed, the greater the number of Wi-Fi network interfaces used, more probes are captured and less probes are lost. The results obtained in the test case 3 show different results compared to this study. Dynamic channel switching captured the most probe requests, although the switching was made from 1 to 12 and not between non overlapping channels as in the study. The positioning of the single channel SensingBox was not equal to the dynamic channel box. The second SensingBox was being USB powered, so it was about 4 meters away from the multi-channel SensingBox. This fact might have influenced the global amount of captured request probes.

It becomes apparent by looking at the graphical information that the estimated occupancy in several measurement have a substantial amount of deviation from ground truth. For the next series of tests perhaps is necessary to increase the probe request capturing time, and experiment with different RSSI filters, the single channel vs. multi-channel strategy also should be further explored.

5.5 SECOND SET OF TESTS

This second set of test tries a new approach in terms of channel hopping, so the SensingBoxes will only jump from non-overlapping channels. So, the hopping will be from channel 1, to channel 6 and finally to channel 11, repeating this process every 10 seconds.

The location chosen was a snack bar, with an open terrace located near a popular Shopping Center. Two SensingBoxes, named SensingBox Genesis and SensingBox Prometheus were used. The difference amongst the two is that SensingBox Genesis uses a capture window interval of 120 seconds and the other 240 seconds. The "x" on Figure 78 signals the approximate spot where the SensingBoxes were positioned. The experiment was conducted for approximate 45 minutes.

Figure 78 Snack Bar Location and SensingBoxes

Measurement	Real Occupancy	Estimated Occupancy Genesis	Total Captured Probes Genesis	Average Sound Level Db
1	8	35	159	54
2	8	43	161	53
3	8	38	155	54
4	4	27	104	55
5	7	36	141	53
6	8	14	66	55
7	6	32	133	52
8	8	22	94	53
9	9	15	63	53
10	9	12	60	55
11	6	40	146	51
12	6	23	72	51
13	5	16	54	51
14	3	15	49	51
15	5	9	52	52
16	4	12	84	53
17	4	16	121	55
18	4	7	35	54
19	5	12	104	52
20	5	17	159	51
21	5	6	72	51
22	8	13	71	54

Table 8 SensingBox Genesis Collected Data

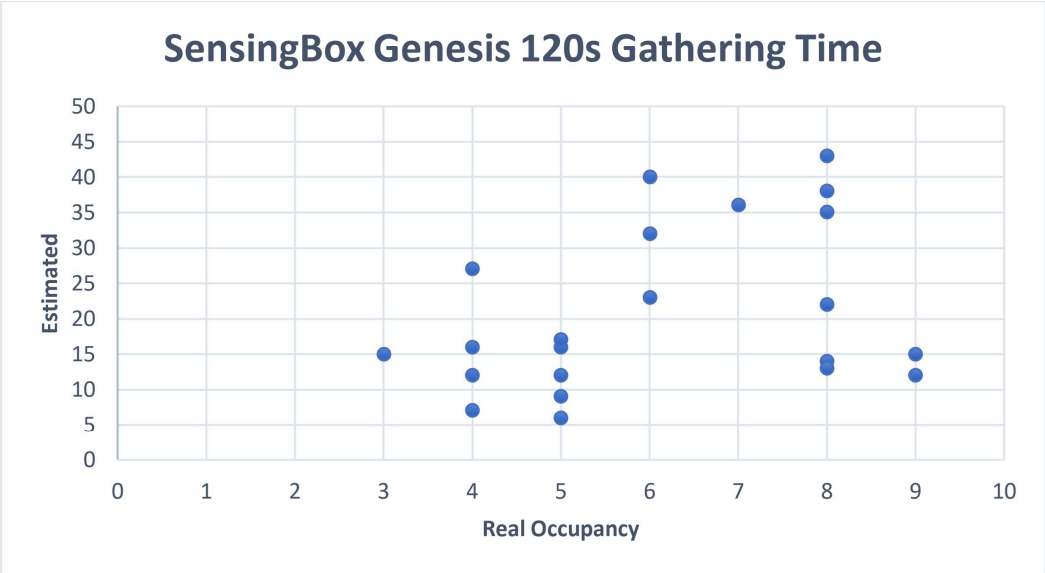


Figure 79 SensingBox Genesis Scatter Plot

The correlation found had a coefficient of r =0.345905, this translates a weak co-relation between the estimated value and the real occupancy. Table 8 shows the

collected values for SensingBox Genesis, with Figure 79 and Figure 80 showing the scatter plot for the collected data points.

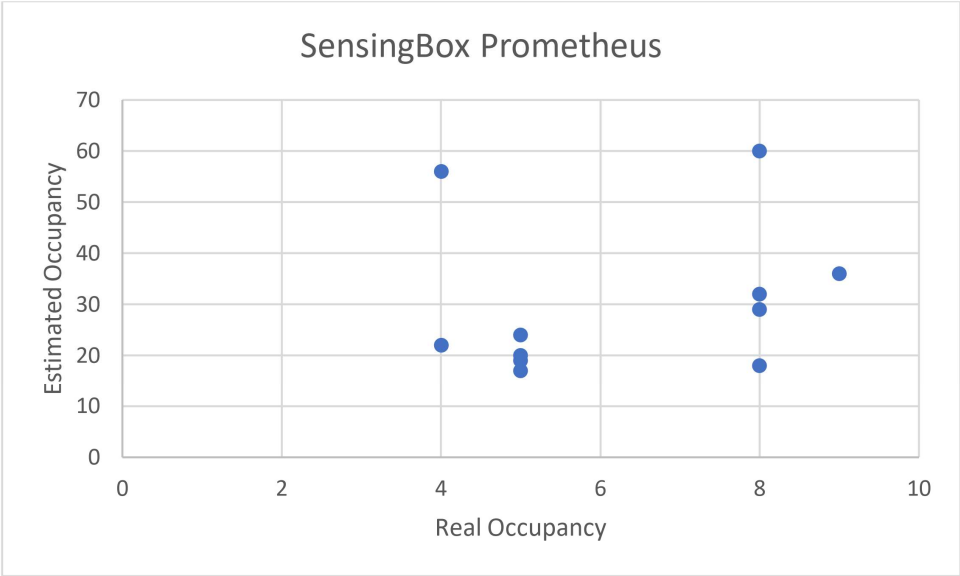


Figure 80 SensingBox Prometheus Scatter Plot

The SensingBox Prometheus values were further from the real occupancy, no correlation can be established. One thing to notice is that even though the gathering time of the SensingBox Prometheus was higher than Genesis, the overall number of probes captured was less. Genesis captured a total of 2155 probes, while Prometheus captured 1779 probes, even though the filter and channel were the same.

6 CONCLUSION

With this project the development of an IoT monitoring system was achieved. The system has proven to be modular and applicable to different context, as seen from the different test cases in different locations. The system was designed to accommodate not only the SensingBox but also other types of devices as long as the API and general message format is respected and MQTT is used as the message exchange protocol.

Most of the correlations found were either non-existence or weak to moderate, these results seem to indicate that for open spaces the Wi-Fi signals are subject to too much interference for multiple sources to have an algorithm based only on probe capturing and RSSI filtering to be accurate enough. In smaller better isolated from interference indoor locations further testing should be conducted, even though the tests seem to suggest that an algorithm based exclusively on counting unique MAC Addresses as a proxy for occupancy seems unlikely. This could also be attributed to the ESP32, it is necessary to test how many probes requests can a ESP32 capture, in a controlled environment using different channel hopping techniques and filters.

As seen in Table 7 the overall cost of the system is low. Although this cost does not account for the hosting infrastructure.

The embedded system development of the project occupied a very significant portion of the overall development time. This was necessary to explore the capabilities of these low-cost IoT development boards from Espressif, albeit the steep learning curve. This also lead that the project ended up having an exploratory course where different approaches were tested before reaching a definitive answer. This exploration nature manifested itself in the selection of sensors, development boards and even methodologies, perhaps one criticism that can be made, is that the project should had been more focused in just a single aspect, for instance the people counting techniques using Wi-Fi sniffing, instead of trying to develop and deploy a complete system.

A good technological review was provided especially in terms of IoT development boards and their potentialities, other Master Thesis projects developed by some of my colleagues related to IoT, that used this type of devices, did not delve as much into the electronic portion of IoT, so perhaps and added value can be found on this work by those who want to develop IoT projects as their thesis project.

6.1 LIMITATIONS AND DIFFICULTIES.

During the development of this project, several difficulties and obstacles were present. The COVID-19 pandemic was an external event that affected us all. In this project for example, some sensors, ESP32s and other electronic components necessary to build the SensingBoxes took an additional time to arrive, also with the university closed or heavily restricted it was not possible to start testing sooner the key elements such as the use of an Enterprise Wi-Fi Connection (Eduroam), or the people counting algorithms/approaches.

In terms of technical difficulties, this project is a very horizontal project, it encompasses several technologies with different abstraction levels, a significative amount of electronics, which were outside of the author's expertise. The number of features to develop also led that sometimes it was easy to get lost or lose focus in the task at hand. Changing through different abstraction levels also might lead to somewhat naïve error such as error of (Figure 49), while nowadays memory is not a key aspect when developing at a higher abstraction level, with most of the high-level programming language abstracting memory control from the developer, this is not true when developing for embedded devices.

With this work the authors also tried to provide a "guideline" of important consideration in terms of development methodologies for the ESP32 platform and general use of sensors. In a broader view, the technologies chosen and practices, have a strong ground in industry practices that were implemented in this project. At a lower level, the Espressif IDF can be used for a more professional and thorough development while the Arduino ecosystem is mostly used for prototyping.

6.2 FUTURE WORK

Several improvements could be made to the project, as additional testing.

Regarding the circuit, better sensors could be used, although this would defeat the purpose of having a low-cost system but depending on the budget a significant improvement could be achieved.

The initial idea for this project was too ambitious to be the basis of a Master Thesis. The first idea proposed a system that would integrate concepts such as gamification. For instance, using the SensingBox to detect who was in a certain room and trigger a quiz for its occupants. Another idea was, using the SensingBox to count and identify the users in a classroom so that the professor would not have to count the attendance. This type of functionalities could prove to be unfeasible or extremally hard to implement properly, although still interesting topics to explore, especially the fusion of gamification with IoT.

In more realistic terms, and looking to this project from an incremental standpoint, the best course of action would probably be, as previously stated, acquiring better quality sensors, modify the SensingBox software so that it could abstract the physical sensors connected to it, giving the possibility to easily change sensor types. The IDF has some features that might be useful for this purpose [87]. Bluetooth Based detection was also not explored although the ESP32 has Bluetooth capabilities (that consume significant memory resources), this could be an interesting topic to explore and compare the two people counting approaches or perhaps combine the two. In a study done by Edoardo Longo et al.,[88] Wi-Fi and Bluetooth were combined and a machine learning algorithm was used to estimate the occupancy of different locations, the results obtained were promising having an overall accuracy of 85.8% for 5 different locations.

Another issue that could also be addressed, is developing the ability to combine the information collected by the SensingBox, especially regarding the people counting algorithm. For example, one SensingBox could be set to stay on Wi-Fi channel 1 scanning for probes, while the others could stay on different channels, and then the collected information could be combined. This would require time synchronization between the different boxes.

The project presentation layer could be further improved in terms of its design and data visualization capabilities, providing different ways to visualize data. A notification system could also be implemented alerting the system administrator in case of low battery or if the SensingBox did not send data when it was supposed to. It is possible that the system infrastructure is changed to an on-premises hosting solution on the near future, as the different components are containerized this process should be relatively straightforward.

References.

[1] «The Growth in Connected IoT Devices Is Expected to Generate 79.4ZB of Data in 2025, According to a New IDC Forecast», *IDC: The premier global market intelligence company*. https://www.idc.com/getdoc.jsp?containerId=prUS45213219 (acedido Jul. 24, 2020).

[2] * All products require an annual contract Prices do not include sales tax, «Global internet user penetration 2021 | Statistic», *Statista*. https://www.statista.com/statistics/325706/global-internet-user-penetration/ (acedido Mai. 10, 2018).

[3] A. Abuarqoub *et al.*, «A Survey on Internet of Things Enabled Smart Campus Applications», em *Proceedings of the International Conference on Future Networks and Distributed Systems*, New York, NY, USA, 2017, p. 50:1-50:7. doi: 10.1145/3102304.3109810.

[4] W. Muhamad, N. B. Kurniawan, Suhardi, e S. Yazid, «Smart campus features, technologies, and applications: A systematic literature review», em *2017 International Conference on Information Technology Systems and Innovation (ICITSI)*, Bandung, Out. 2017, pp. 384–391. doi: 10.1109/ICITSI.2017.8267975.

[5] Cisco Australia and New Zealand, *Deakin University smart campus*. Acedido: Jun. 21, 2019. [Em linha Video]. Disponível em: https://www.youtube.com/watch?v=O1eKiQRMQ9o

[6] «IBM Watson | IBM». https://www.ibm.com/watson (acedido Jun. 21, 2019).

[7] Microsoft Developer, *Microsoft's smart campus IoT and AI project "Garcon"*. Acedido: Jun. 21, 2019. [Em linha Video]. Disponível em: https://www.youtube.com/watch?v=Ad_EHDcomR8

[8] J. Domaszewicz e S. Lalis, «Soft Actuation for Home and Office», Jul. 2013, pp. 188–195. doi: 10.1109/IE.2013.32.

[9] J. Schneider, D. Börner, P. Van Rosmalen, e M. Specht, «Augmenting the Senses: A Review on Sensor-Based Learning Support», *Sensors*, vol. 15, n. 2, Art. n. 2, Fev. 2015, doi: 10.3390/s150204097.

[10] «Flipped learning | Higher Education Academy». https://www.heacademy.ac.uk/knowledge-hub/flipped-learning-0 (acedido Jun. 28, 2019).

[11] «Case-Based Learning | Centre for Teaching and Learning». https://www.queensu.ca/ctl/teaching-support/instructional-strategies/case-based-learning (acedido Jun. 28, 2019).

[12] «Smart Building Energy Management», *Intel*. https://www.intel.com/content/www/us/en/energy/solutions/smart-building-energy-management.html (acedido Jun. 21, 2019).

[13] «The average cost of IoT sensors is falling», *Atlas*, Abr. 27, 2016. http://www.theatlas.com/charts/BJsmCFAl (acedido Abr. 27, 2018).

[14] C. Raghavachari, V. Aparna, S. Chithira, e V. Balasubramanian, «A Comparative Study of Vision Based Human Detection Techniques in People Counting Applications», *Procedia Computer Science*, vol. 58, pp. 461–469, 2015, doi: 10.1016/j.procs.2015.08.064.

[15] «Density | A People Counter & API», *Density*. https://www.density.io (acedido Abr. 27, 2018).

[16] «Fig. 1. Working principle of RFID.», *ResearchGate*. https://www.researchgate.net/figure/Working-principle-of-RFID_fig1_260711663 (acedido Dez. 15, 2020).

[17] J. W. Choi, X. Quan, e S. H. Cho, «Bi-Directional Passing People Counting System Based on IR-UWB Radar Sensors», *IEEE Internet of Things Journal*, vol. 5, n. 2, pp. 512–522, Abr. 2018, doi: 10.1109/JIOT.2017.2714181.

[18] N. Nunes, M. Ribeiro, C. Prandi, e V. Nisi, «Beanstalk: a community based passive wi-fi tracking system for analysing tourism dynamics», em *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '17*, Lisbon, Portugal, 2017, pp. 93–98. doi: 10.1145/3102113.3102142.

[19] G. Pipelidis, N. Tsiamitros, M. Kessner, e C. Prehofer, «HuMAn: Human Movement Analytics via WiFi Probes», em *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Kyoto, Japan, Mar. 2019, pp. 370–372. doi: 10.1109/PERCOMW.2019.8730703.

[20] «What is a System on a Chip (SoC)? - Definition from Techopedia», *Techopedia.com*. https://www.techopedia.com/definition/702/system-on-a-chip-soc (acedido Mai. 20, 2020).

[21] «Arduino - Introduction». https://www.arduino.cc/en/Guide/Introduction (acedido Jul. 05, 2018).

[22] «Arduino Technology Architecture and its Applications», *Edgefx Kits Official Blog*, Abr. 21, 2015. http://www.edgefxkits.com/blog/arduino-technology-architecture-and-applications/ (acedido Jul. 05, 2018).

[23] «Most Popular - Arduino». https://store.arduino.cc/arduino-genuino/most-popular?dir=asc&order=position (acedido Jul. 21, 2020).

[24] «1. The Arduino Family - Arduino: A Technical Reference [Book]». https://www.oreilly.com/library/view/arduino-a-technical/9781491934319/ch01.html (acedido Jun. 15, 2020).

[25] By, «New Chip Alert: The ESP8266 WiFi Module (It's $5)», *Hackaday*, Ago. 26, 2014. https://hackaday.com/2014/08/26/new-chip-alert-the-esp8266-wifi-module-its-5/ (acedido Jun. 15, 2020).

[26] «FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions», *FreeRTOS*. /index.html (acedido Mai. 21, 2020).

[27] «ESP-IDF FreeRTOS SMP Changes - ESP32 - — ESP-IDF Programming Guide latest documentation». https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/freertos-smp.html (acedido Mai. 21, 2020).

[28] «Understanding How a Voltage Regulator Works | Analog Devices». https://www.analog.com/en/technical-articles/how-voltage-regulator-works.html# (acedido Jul. 08, 2020).

[29] «#193 Comparison of 10 ESP32 Battery powered Boards without display (incl. deep-sleep) - YouTube». https://www.youtube.com/watch?v=-769_YIeGmI&t=77s (acedido Jul. 08, 2020).

[30] H. Fleischmann, *grillbaer/esp32-power-consumption-test*. 2020. Acedido: Jul. 08, 2020. [Em linha]. Disponível em: https://github.com/grillbaer/esp32-power-consumption-test

[31] Electrical4U, «Battery Working Principle: How does a Battery Work? | Electrical4U», *https://www.electrical4u.com/*. https://www.electrical4u.com/battery-working-principle-of-batteries/ (acedido Jul. 20, 2020).

[32] mischa, «How a battery works», *Curious*, Fev. 25, 2016. https://www.science.org.au/curious/technology-future/batteries (acedido Jul. 20, 2020).

[33] «Lithium-Ion Battery», *Clean Energy Institute*. https://www.cei.washington.edu/science-of-solar/battery-technology/ (acedido Jul. 19, 2020).

[34] D. Lisbona e T. Snee, «A review of hazards associated with primary lithium and lithium-ion batteries», *Process Safety and Environmental Protection*, vol. 89, n. 6, pp. 434–442, Nov. 2011, doi: 10.1016/j.psep.2011.06.022.

[35] K. Murashko, «Thermal modelling of commercial lithium-ion batteries», 2016. doi: 10.13140/RG.2.1.3930.0723.

[36] «Harding Energy | Lithium Ion batteries | Lithium Polymer | Lithium Iron Phosphate», *Harding Energy*. http://www.hardingenergy.com/lithium-2/ (acedido Jul. 20, 2020).

[37] «Nickel-cadmium Batteries: Basic theory and maintenance procedures», *Aviation Pros*, Mai. 2002. https://www.aviationpros.com/engines-components/article/10387569/nickelcadmium-batteries-basic-theory-and-maintenance-procedures (acedido Jul. 20, 2020).

[38] «Nickel-based Batteries Information – Battery University». https://batteryuniversity.com/learn/article/nickel_based_batteries (acedido Jul. 20, 2020).

[39] «Battery Power Online | The World of Alkaline Batteries». https://www.batterypoweronline.com/markets/batteries/the-world-of-alkaline-batteries/ (acedido Jul. 20, 2020).

[40] «Designing applications with Li-ion batteries - Battery Management | Richtek Technology». https://www.richtek.com/battery-management/en/designing-liion.html (acedido Jul. 24, 2020).

[41] «Sound and Noise - What is sound and What is noise?» https://www.epd.gov.hk/epd/noise_education/web/ENG_EPD_HTML/m1/intro_1.html (acedido Jun. 27, 2020).

[42] «Binary Representation of Sound», *teachwithict.com*. https://www.teachwithict.com/binary-representation-of-sound1.html (acedido Abr. 19, 2021).

[43] H. Q. T. Ngo, N. thanh phuong, e H. Nguyen, «Research on a Low-Cost, Open-Source, and Remote Monitoring Data Collector to Predict Livestock's Habits Based on Location and Auditory Information: A Case Study from Vietnam», *Agriculture*, vol. 10, p. 180, Mai. 2020, doi: 10.3390/agriculture10050180.

[44] M. Ribeiro, N. Nunes, V. Nisi, e J. Schöning, «Passive Wi-Fi monitoring in the wild: a long-term study across multiple location typologies», *Pers Ubiquit Comput*, Set. 2020, doi: 10.1007/s00779-020-01441-z.

[45] *Google Livros*. Acedido: Jul. 13, 2020. [Em linha]. Disponível em: https://books.google.com/books/about/_.html?hl=pt-PT&id=D_GrQa2ZcLwC

[46] «IEEE 802.11 Mac Frame», *GeeksforGeeks*, Dez. 28, 2018. https://www.geeksforgeeks.org/ieee-802-11-mac-frame/ (acedido Mai. 12, 2021).

[47] L. Hattery, «How MAC Address Randomization Can Affect the Wi-Fi Experience». https://blog.elevensoftware.com/how-mac-address-randomization-can-affect-the-wifi-experience (acedido Abr. 28, 2021).

[48] «802.11 WLAN or Wi-Fi Channels and Frequency Bands - A Primer Webinar - YouTube». https://www.youtube.com/watch?v=HHU2p_NfSx4 (acedido Jul. 17, 2020).

[49] «Legacy Open Mesh: Which Wifi channels should I use?» https://help.datto.com/s/article/KB360023805172 (acedido Abr. 28, 2021).

[50] P. Barsocchi, S. Lenzi, S. Chessa, e G. Giunta, «A Novel Approach to Indoor RSSI Localization by Automatic Calibration of the Wireless Propagation Model», em *VTC Spring 2009 - IEEE 69th Vehicular Technology Conference*, Abr. 2009, pp. 1–5. doi: 10.1109/VETECS.2009.5073315.

[51] M. Vari e D. Cassioli, «mmWaves RSSI indoor network localization», em *2014 IEEE International Conference on Communications Workshops (ICC)*, Jun. 2014, pp. 127–132. doi: 10.1109/ICCW.2014.6881184.

[52] «2.1. The HTTP Request/Response Model - JavaServer Pages, 3rd Edition [Book]». https://www.oreilly.com/library/view/javaserver-pages-3rd/0596005636/ch02s01.html (acedido Jul. 22, 2020).

[53] Z. Shelby, K. Hartke, e C. Bormann, «The Constrained Application Protocol (CoAP)». https://tools.ietf.org/html/rfc7252#page-5 (acedido Jul. 23, 2020).

[54] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, e C. K.-Y. Tan, «Performance evaluation of MQTT and CoAP via a common middleware», em *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Singapore, Abr. 2014, pp. 1–6. doi: 10.1109/ISSNIP.2014.6827678.

[55] «COAP vs MQTT | Difference between COAP and MQTT protocols». https://www.rfwireless-world.com/Terminology/COAP-vs-MQTT.html (acedido Abr. 28, 2021).

[56] «Introducing the MQTT Protocol - MQTT Essentials: Part 1». https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/ (acedido Ago. 21, 2019).

[57] R. Joncas, «MQTT and CoAP, IoT Protocols | The Eclipse Foundation». https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php (acedido Jul. 23, 2020).

[58] N. Naik, «Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP», em *2017 IEEE International Systems Engineering Symposium (ISSE)*, Out. 2017, pp. 1–7. doi: 10.1109/SysEng.2017.8088251.

[59] «HiveMQ - Enterprise ready MQTT to move your IoT data». https://www.hivemq.com/ (acedido Abr. 26, 2021).

[60] «Node.js | Event Loop - GeeksforGeeks». https://www.geeksforgeeks.org/node-js-event-loop/ (acedido Abr. 28, 2021).

[61] *timescale/timescaledb*. Timescale, 2021. Acedido: Abr. 02, 2021. [Em linha]. Disponível em: https://github.com/timescale/timescaledb

[62] <img class="avatar" src="https://iotvnaw69daj i optimole com/QGuBBBY hVRt~66341/w:auto/h:auto/q:90/https://s34628 pcdn co/wp-content/uploads/2018/06/shaumik-60x60 jpg" alt="Shaumik Daityari" width="60" height="60"> Shaumik Daityari, «Angular vs React vs Vue: Which Framework to Choose in 2020», *CodeinWP*, Jan. 10, 2019. https://www.codeinwp.com/blog/angular-vs-vue-vs-react/ (acedido Nov. 24, 2020).

[63] K. E. Wiegers, «First Things First: Prioritizing Requirements», p. 6, 1999.

[64] D. Firesmith, «Prioritizing Requirements.», *JOT*, vol. 3, n. 8, p. 35, 2004, doi: 10.5381/jot.2004.3.8.c4.

[65] «Arduino Nano 33 BLE». https://store.arduino.cc/arduino-nano-33-ble (acedido Abr. 27, 2021).

[66] «Get Started - ESP32 - — ESP-IDF Programming Guide latest documentation». https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html (acedido Jul. 08, 2020).

[67] *Developing ESP32 applications using Espressif IDF (IoT Development Framework) Eclipse Plugins*. Acedido: Jun. 02, 2020. [Em linha Video]. Disponível em: https://www.youtube.com/watch?v=DqnZy4d03_s&t=1561s

[68] «JTAG Debugging - ESP32 - — ESP-IDF Programming Guide latest documentation». https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/jtag-debugging/index.html (acedido Jul. 09, 2020).

[69] «Developing ESP32 applications using Espressif IDF (IoT Development Framework) Eclipse Plugins - YouTube». https://www.youtube.com/watch?v=DqnZy4d03_s&t=1857s (acedido Jul. 08, 2020).

[70] «ESP32 Forum - Index page». https://www.esp32.com/ (acedido Abr. 26, 2021).

[71] «RTOS: Grupo de eventos para sincronização de tarefas - Embarcados», *Embarcados - Sua fonte de informações sobre Sistemas Embarcados*, Set. 14, 2018. https://www.embarcados.com.br/rtos-grupo-de-eventos-para-sincronizacao/ (acedido Dez. 13, 2020).

[72] «ESP32 Wi-Fi & Bluetooth Modules I Espressif». https://www.espressif.com/en/products/modules/esp32 (acedido Mai. 10, 2021).

[73] «Angular - Component styles». https://angular.io/guide/component-styles (acedido Mai. 13, 2021).

[74] «Powtoon | The World's #1 Visual Communication Platform». https://www.powtoon.com/ (acedido Mai. 13, 2021).

[75] J. Bradley, N. Sakimura, e M. B. Jones, «JSON Web Token (JWT)». https://tools.ietf.org/html/rfc7519#section-3.1 (acedido Fev. 28, 2021).

[76] D. Arrachequesne, «Server API», *Socket.IO*, Mar. 01, 2021. https://socket.io/docs/v3/server-api/index.html (acedido Mar. 01, 2021).

[77] M. Fowler, «Continuous Integration», p. 14.

[78] «DevOps engineer: IT's most in-demand title for the future». https://enterprisersproject.com/article/2020/8/devops-engineer-in-demand-it-title (acedido Dez. 10, 2020).

[79] M. Shahin, M. A. Babar, e L. Zhu, «Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices», *IEEE Access*, vol. 5, pp. 3909–3943, 2017, doi: 10.1109/ACCESS.2017.2685629.

[80] C. Anderson, «Docker [Software engineering]», *IEEE Software*, vol. 32, n. 3, pp. 102-c3, Mai. 2015, doi: 10.1109/MS.2015.62.

[81] «Docker vs. Virtual Machines: Differences You Should Know», *Cloud Academy*, Out. 29, 2019. https://cloudacademy.com/blog/docker-vs-virtual-machines-differences-you-should-know/ (acedido Dez. 15, 2020).

[82] «Docker Mastery: The Complete Toolset From a Docker Captain», *Udemy*. https://www.udemy.com/course/docker-mastery/ (acedido Dez. 10, 2020).

[83] «Docker Hub». https://hub.docker.com/ (acedido Dez. 10, 2020).

[84] L. M. Candanedo e V. Feldheim, «Accurate occupancy detection of an office room from light, temperature, humidity and CO 2 measurements using statistical learning models», *Energy and Buildings*, vol. 112, pp. 28–39, Jan. 2016, doi: 10.1016/j.enbuild.2015.11.071.

[85] H. Hong, G. D. De Silva, e M. C. Chan, «CrowdProbe: Non-invasive Crowd Monitoring with Wi-Fi Probe», *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 2, n. 3, pp. 1–23, Set. 2018, doi: 10.1145/3264925.

[86] J. Freudiger, «Short: How Talkative is your Mobile Device? An Experimental Study of Wi-Fi Probe Requests», p. 6.

[87] «Hardware Abstraction - ESP32 - — ESP-IDF Programming Guide latest documentation». https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/hardware-abstraction.html (acedido Dez. 15, 2020).

[88] «Accurate occupancy estimation with WiFi and bluetooth/BLE packet capture | Elsevier Enhanced Reader». https://reader.elsevier.com/reader/sd/pii/S1389128618313045?token=574233429AEC8611FD2EE23707A4C6B01E2AE0557842F6B1129D5D19614AA744BE208AE79E0752B940FB6A71B6A4CECA&originRegion=eu-west-1&originCreation=20210407115143 (acedido Abr. 07, 2021).

7 APPENDICES

Appendix A – Docker Compose File

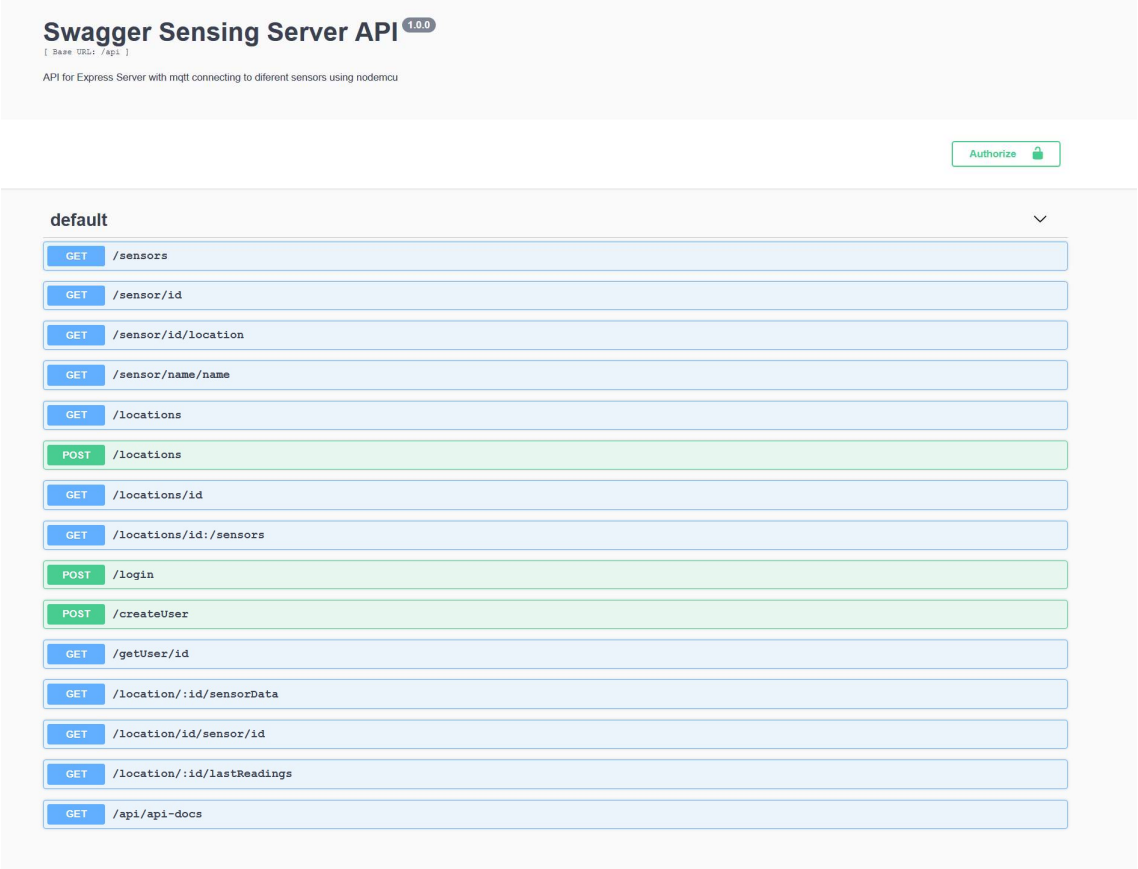
docker-compose.yml

```yaml
version: '3'

services:
  nodejsback:
    build:
      context: .
      dockerfile: nodejs.Dockerfile
    image: nodejsbackend
    container_name: sensingBoxBackEnd
    restart: unless-stopped
    env_file: .env
    environment:
      - POSTGRE_USERNAME=$POSTGRE_USERNAME
      - POSTGRE_PASSWORD=$POSTGRE_PASSWORD
      - POSTGRE_DATABASE=$POSTGRE_DATABASE
      - POSTGRE_HOST=$POSTGRE_HOST
      - POSTGRE_PORT=$POSTGRE_PORT
      - PORT_LISTEN=$PORT_LISTEN
      - HOST_IP=$HOST_IP
    ports:
    - "3000:3000"
  nginxreverseproxy:
    build:
      context: .
      dockerfile: nginxreverseproxy.Dockerfile
    ports:
    - "8080:80"
networks:
  default:
    external:
      name: sensingBox-network


```

Appendix – B Node.js Server Api



Appendix-C

I18n HTML annotation

```
<nav class="nav-bar">
    <div class="navigation" *ngIf="!logginFlag">
        <button i18n class="navigation-item" [routerLink]="['/home']">
          <fa-icon class="nav-icon" [icon]="icon_home"></fa-icon>Início
        </button>
        <button i18n class="navigation-item" [routerLink]="['/locations']">
          <fa-icon class="nav-icon" [icon]="icon_location"></fa-icon>Salas
        </button>
        <button i18n class="navigation-item" [routerLink]="['/additionalInformation']">
          <fa-icon class="nav-icon" [icon]="icon_infoCircle"></fa-icon>Mais
          Informações
        </button>
    </div>
    <div *ngIf="logginFlag">
      <button  class="navigation-item">
        <fa-icon class="nav-icon" [icon]="icon_infoCircle"></fa-icon>Salas
      </button>
    </div>
</nav>
```

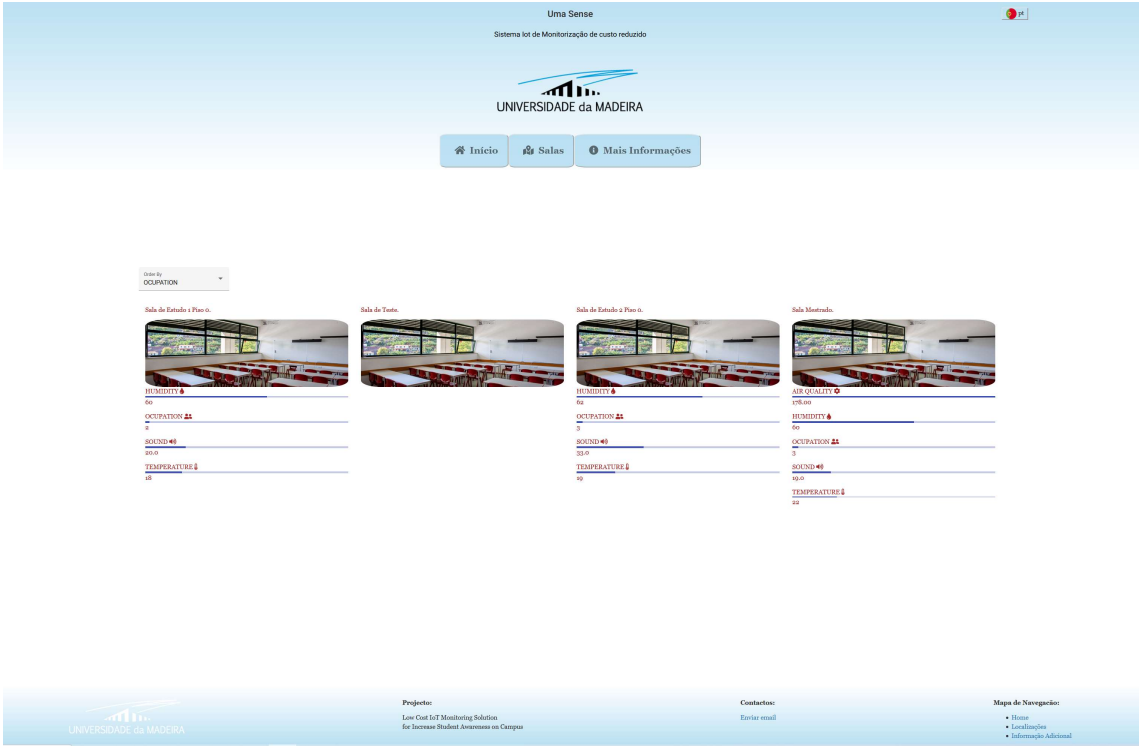
Generated Translation FIle

```
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
  <file source-language="en-US" datatype="plaintext" original="ng2.template">
    <body>
      <trans-unit id="84a7873a4a527831a147d2b0c1b78258fb354f54" datatype="html">
        <source><x id="START_TAG_FA_ICON" ctype="x-fa_icon" equiv-text="   &lt;fa-icon class=&quot;nav-icon&quot; [icon]=&quot;icon_hom"/><x id="CLOSE_TAG_FA_
        equiv-text="e&quot;&gt;&lt;/fa-ic"/>Início </source>
        <target>Home</target>
        <context-group purpose="location">
          <context context-type="sourcefile">src/app/nav-bar/nav-bar.component.html</context>
          <context context-type="linenumber">4,5</context>
        </context-group>
      </trans-unit>
      <trans-unit id="ef137831a4c3c684fd883bca9617f1c431ef91f5" datatype="html">
        <source><x id="START_TAG_FA_ICON" ctype="x-fa_icon" equiv-text="      &lt;fa-icon class=&quot;nav-icon&quot; [icon]=&quot;icon_loca"/><x id="CLOSE_TAG
        <target>Locations</target>
        <context-group purpose="location">
          <context context-type="sourcefile">src/app/nav-bar/nav-bar.component.html</context>
          <context context-type="linenumber">7,8</context>
        </context-group>
      </trans-unit>
      <trans-unit id="24f1519d5834c8a73aa75d8a746c234ebc228130" datatype="html">
        <source><x id="START_TAG_FA_ICON" ctype="x-fa_icon" equiv-text="         &lt;fa-icon class=&quot;nav-icon&quot; [icon]=&quot;icon_inf"/><x id="CLOSE_T
        <target>Additional Technical Information</target>
        <context-group purpose="location">
          <context context-type="sourcefile">src/app/nav-bar/nav-bar.component.html</context>
          <context context-type="linenumber">10,12</context>
        </context-group>
      </trans-unit>
      <trans-unit id="8bebe1e0df96caacc55da3c0e2bcabd3368b09bd" datatype="html">
        <source> Contactos: </source>
        <target> Contact </target>
        <context-group purpose="location">
          <context context-type="sourcefile">src/app/footer/footer.component.html</context>
          <context context-type="linenumber">12,13</context>
        </context-group>
      </trans-unit>
      <trans-unit id="65cb19c663ad26c9ca006b3823f5743a8c0f6e96" datatype="html">
        <source>Enviar email</source>
        <target>Send Email </target>
```

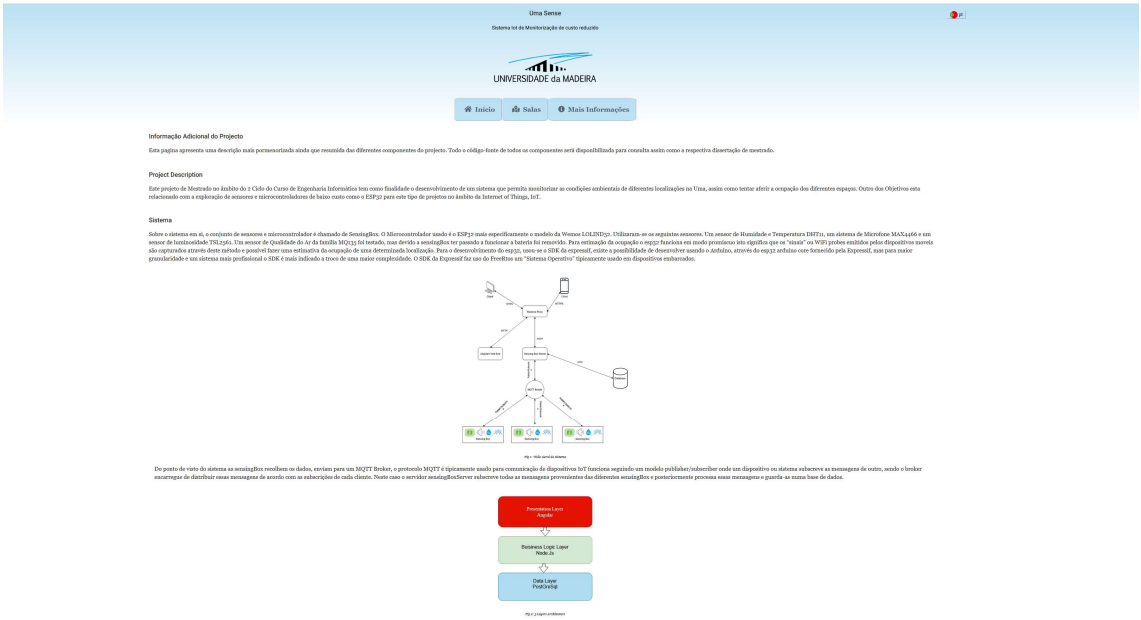
Appendix-D

Uma Sense Views - Portuguese Version

Landing Page



Additional Information View



Applicação Front end

A aplicação de front-end foi desenvolvida usando a Framework angular. Esta aplicação permite a visualização dos dados recolhidos de acordo com as diferentes localizações e possuí ainda uma descrição explicativa do projeto assim como um vídeo animado sobre as diferentes componentes do projeto. Foi ainda desenvolvido uma componente de backoffice que permite a gestão das diferentes sensingBox e modificação dos diferentes parâmetros tais como o tempo de captura de dados por sensor, o sleep time do dispositivo, entre outras configurações, o que conferem ao dispositivo e ao sistema uma maior modularidade e polivalência.

Fig 3. Localização View

Sensing Box

Na figura abaixo é possível ver uma das sensingBox e respetiva eletrónica dentro de uma caixa protetiva de cartão.

Fig 4. Interior da Sensingbox

Relativamente ao firmware do esp32 vários componentes foram desenvolvidos a Figura 5 representa o diagrama de componentes do sistema, revelando o sistema e as suas interações.

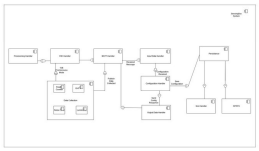


Fig 5. Diagrama de Componentes da Sensingbox

Mobile View

Appendix-D SensingBox Standard Workflow