



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
Department of Engenharia de Electrónica e Telecomunicações e de
Computadores

IoT System for pH Monitoring in Industrial Facilities

47185 : Miguel Agostinho da Silva Rocha (a47185@alunos.isel.pt)
47128 : Pedro Miguel Martins da Silva (a47128@alunos.isel.pt)

Report for the Curricular Unit of Project and Seminar
of Bachelor's Degree in Computer Science and Software Engineering

Supervisor: Doctor Rui Duarte

Abstract

In Germany there are several restrictions on the discharge of waste resulting from manufacturing activities into nature. Therefore, our group has teamed up with the German company *Mommertz GmbH* which manufactures gas boiler filters to regulate the acidity of the boiler discharge liquid before it is released into the public sewage system. The issue at hand is that, over time, these filters experience deterioration over time due to usage, leading to a decline in their original filtering capacity, necessitating regular servicing. Currently, the company's strategy for dealing with this problem is to advise customers to periodically test the pH of the filter unit, and if the pH is outside an established safety interval, specified by the company, at the point where the filtered material exits the filter, maintenance will be required for that unit. In this project, we implement an automation system with very low power consumption for making a proactive maintenance for the neutralization mechanisms. This solution is based on an embedded system and server-side computation. Specifically, it includes sensors to measure environmental temperature and humidity, as well as a pH sensor to measure the pH level of the liquid passing through the filter and a flood detection mechanism. The sensor data will be shared with a unit which will process the data and send a signal to the manager of the device, via email, indicating that a value surpassed a threshold. The prototype was tested and analysed, resulting in a system that meets all the required and additional criteria. Our device achieves ultra-low power consumption and captures environmental data.

Table of Contents

1	Introduction	8
1.1	Motivation	8
1.2	Objectives	9
1.3	Document Organization	10
2	Background	11
2.1	IoT Systems	11
2.1.1	Sensors	11
2.1.2	Embedded Systems	12
2.1.3	Backend	16
2.1.4	Communication Protocols	18
2.1.5	MQTT Broker	18
2.1.6	MCU Configuration	18
2.1.7	Frontend	19
2.2	Existing Solutions	19
2.3	Summary	20
3	Proposed IoT System Architecture	21
3.1	System Overview	21
3.2	IoT Node	22
3.2.1	MCU	22
3.2.2	Sensor Modules	24
3.2.3	Sensor readings - Parallelized vs Serialized	25
3.2.4	Sensor Calibration	25
3.2.5	Sensor For PH	29
3.2.6	Sensor For Ambient Humidity And Temperature	31
3.2.7	Sensor for water Leaks	32
3.2.8	Assembly	33

3.2.9	Wi-Fi	35
3.2.10	Time	35
3.3	Backend	36
3.3.1	MQTT: Topics And Representations	36
3.3.2	Application Server	37
3.3.3	Application Domain	37
3.3.4	Sensor Analysis	38
3.3.5	Authentication and Authorization	39
3.3.6	Database	40
3.4	Website	41
3.4.1	Data Visualization and Device Logs	42
3.4.2	Website Architecture	42
3.4.3	Hypermedia	43
3.4.4	Protected Resources	43
3.4.5	Error Handling	44
3.5	System Tests	45
3.5.1	Backend	45
3.5.2	MCU	45
3.5.3	Power Consumption	45
3.6	Security Mesures	48
3.6.1	Password Storage	49
3.6.2	Website	49
3.7	Deploy	51
3.8	Summary	52
4	Conclusion	53
4.1	Future Work	53
4.1.1	Hardware Adaptation	53
4.1.2	Dynamic Configuration	54
4.1.3	pH Transformation with Temperature Compensation	54
4.1.4	Hermetic Enclosure	54
4.1.5	Scalability	54
4.1.6	Water Flow sensor	54
4.1.7	System future support	55

List of Figures

2.1	DHT11 Sensor Circuit	12
2.2	Espressif ESP32-S2-SAOLA-1M (WROOM)	16
3.1	Main system architecture	21
3.2	IoT Node Module Architecture	22
3.3	Sliding window algorithm ¹	26
3.4	Standard Deviation over Time	27
3.5	Standard Deviation over Time	28
3.6	Voltage divider circuit used to achieve lower output voltage	30
3.7	Graph illustrating the measured voltage with the MCU against the real voltage measured with an oscilloscope, of a liquid solution, pH equal to 8.	30
3.8	Graph illustrating effect of differing reference voltages on the ADC voltage curve.	31
3.9	Graph illustrating the measured ADC and voltage of a liquid solution, pH equal to 4.	32
3.10	Graph illustrating the measured humidity and temperature, from with the DHT11 sensor, at room temperature.	32
3.11	Graph illustrating the measured ADC and voltage when the sensor is immersed in water and taken out.	33
3.12	Current IoT Node assembly in the breadboard	34
3.13	Spring server overview	37
3.14	Relational DB Entity Model	41
3.15	Time Series DB Sensor Record representation	41
3.16	Single Page Application components diagram	42
3.17	Authentication control in the SPA	44
3.18	Graph illustrating the ESP32-S2 energy consumption measured when the MCU wakes up to read from all sensors	47

3.19 Image showing the 4 stages in a reading cycle	48
3.20 Visual representation of the function of the salt ²	50
3.21 Containers for each system service layer	51

List of Tables

2.1	Comparison Between different MCUs from espressif family	13
2.2	MSP430FR413x analysis	14
3.1	Window Size vs. Time to Stabilize	28
3.2	Results for the different MCU consumption stages	46

1

Introduction

This document describes the implementation of the IoT System for pH Monitoring in Industrial Facilities project, developed as part of the final project in the Bachelor in Computer science and engineering degree.

1.1 Motivation

In Germany, the worksheet ATV-DWK-A 251 Edition 11/2011 regulates the obligation to neutralize acidic condensates¹. This is due to the protection of residual waters, and the sewage pipes, that conduct them. Any entity that acquired gas boilers, is obligated to neutralize the pH of water before discharging it into the public sewage system.

In the course of our final project, we collaborated with a German company called Mommertz GmbH. The aim of Mommertz's business model is to produce specialized filters designed to neutralize the pH of water resulting from the action of gas boilers, thus ensuring compliance with legal obligations.

The pH scale is a measure of the acidity or alkalinity of a solution. It is a logarithmic scale that indicates the concentration of hydrogen ions (H^+) in a solution. The pH scale ranges from 0 to 14, where a pH of 7 is considered neutral. Values below 7 indicate acidity, with lower numbers indicating higher acidity, while values above 7 indicate alkalinity or basicity, with higher numbers indicating higher alkalinity[1].

The filter comprises granules that serve to neutralize the pH of the water, including the water flowing from the boiler and passing through the filter. This system, like any industrial product, may suffer from several problems during its lifetime. As the water flows and undergoes neutralization, the granules gradually decompose, resulting in a reduction in the effectiveness of neutralization. Consequently, the pH of the water will not be fully neutralized. Conversely, if an excessive amount of granules is used, the pH of the water may become excessively high.

Furthermore, over time, water leaks may occur, and there is also a possibility of unexpected interruptions in water flow, necessitating maintenance. Currently, all the inspection for possible problems in the mechanism is done manually, which poses

¹Vorschriften, Mommertz GmbH, Available online: <https://mommertz.de/vorschriften.html>, Accessed: 01.07.2023.

significant drawbacks as these types of systems are often situated in hard to reach locations.

Additionally, manual inspections result in needless maintenance and are prone to human error. This heavy human dependency in the inspection for a possible malfunction in the behaviour of the mechanism is what our system aims to fix.

The very nature of this problem seems a very good candidate for the use of IoT (Internet of Things) technology in order to automate the device inspection process. Using sensors to monitor physical aspects of the neutralization system, we can, not only, determine if maintenance is needed but also extract key information about the efficiency of the neutralization device.

1.2 Objectives

The project aims to leverage Internet of Things (IoT) technology to automate the monitoring and maintenance of the filtration system. By integrating a microcontroller unit (MCU) with sensors, the system will gather data concerning the neutralization filter and transmit it to an application server.

Another objective of our project is to develop an application server that serves as the backbone of the system. This server will play a crucial role in storing, processing, and managing all the collected data from the multiple neutralization systems registered in this infrastructure. The backend has multiple functionalities:

- Data storage and analysis: The backend will include data storage mechanisms to efficiently store and manage the collected data;
- Broker functionality: The backend will incorporate a broker component that acts as an intermediary between the application server and the microcontroller unit (MCU). The broker facilitates communication and data exchange between the MCU and the application server, ensuring reliable and efficient transmission of information;
- REST Web API for system integration: To enable integration with other systems and applications, the backend will expose a Web API. This API will allow different systems to consume the collected data and access specific functionalities provided by the server. It provides a standardized interface for easy interaction with the system.

Finally, enabling remote monitoring and data visualization: The development of a user-friendly web application serves as an objective in this project. It allows users to remotely access and visualize the collected data from the neutralization system. This interface provides easy interaction and comprehensive insights into the system's performance, enabling efficient decision-making and proactive maintenance.

To ensure the project's value, competitiveness, and appeal, several requirements need to be fulfilled. Here are the guidelines we have agreed with the company to steer the project in the right direction:

- Cost-effective integration of the microcontroller unit into IoT sensor node;
- Prolonged battery life for minimal maintenance requirements;
- Comprehensive documentation for user-friendly operation and understanding;
- pH data reading from the filter;
- Notification to the filter owner or manager if any value exceeds a set threshold, i.e. proactive maintenance;
- Collect two pH readings per day.

In addition to the company's requirements, we have identified several additional requirements that aim to enhance the overall quality of the project:

- Intuitive and user-friendly website design for easy navigation and access to information;
- Proactive sensor calibration;
- Adding more sensors to gather more data from the equipment;
- Support for a role system, that includes an "admin" that is meant to be a member of the company that will extract data from all the devices registered in the system and a "client" representing a client of the company.

1.3 Document Organization

This section offers a summary of the report's chapter organization, detailing the subjects and areas addressed. The document organization ensures a logical flow of information and facilitates understanding of the project's scope, objectives, and findings.

In the first chapter, it is provided information about the problem we are trying to solve and how our project solution is a possible approach. We also state the overall project objectives and the requirements we proposed to follow. The second chapter provides contextual information. It lays the groundwork for comprehending the project's strategies and technologies used, as well the decisions taken to choose each element of the system. The purpose of the third chapter is to provide an in-depth explanation of the system architecture, splitting the problem into different modules and make a clear explanation of how each part of the problem was approached and the challenges we encountered during the implementation process. The conclusion chapter outlines the most important aspects of solution and as well as the most difficult changes we had to face during the duration of the project.

2

Background

In this chapter will be provided the necessary information for the understanding of the elements and technologies of our proposed architecture, as well as the analysis of the different alternatives for each component selected. Additionally, the chapter offers general insights into IoT technology, providing a broader understanding of its principles and applications.

2.1 IoT Systems

The Internet of Things (IoT) refers to the network of interconnected devices, objects, and systems that are capable of collecting, exchanging, and sharing data over the Internet[2]. Regarding the physical components, one aspect of IoT systems is the deployment of sensors, as they play a vital role in gathering data from the surrounding environment. The collected data is transmitted to a central system for processing, storage and analysis. That system is responsible to centralize data management, process incoming sensor data, and facilitate communication and coordination between connected devices. With this architecture we can create a system that enables communication between physical devices, collect and analyse data from those devices, and utilize the insights gained to improve efficiency, automation, and decision-making in various domains such as healthcare, transportation, manufacturing among others.

2.1.1 Sensors

To achieve our solution, we needed to implement several sensors. The pH measurement circuit contains a pH probe that oscillates between positive and negative voltages, representing pH values. This sensor plays a role in our system as it is responsible for analysing the pH of the liquid¹. It serves as a tool for evaluating the productivity of the neutralization mechanism, providing insights into its performance.

The ambient temperature and humidity measurements were carried out by

¹Measure pH with a low-cost Arduino pH sensor board, Henry Cheung, Available online: <https://www.e-tinkers.com/2019/11/measure-ph-with-a-low-cost-arduino-ph-sensor-board/>, Accessed: 11.05.2023

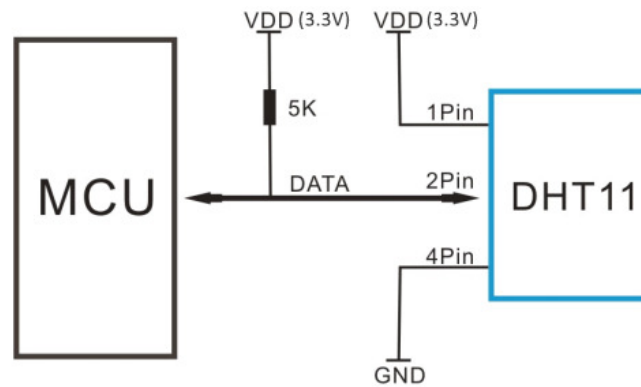


Figure 2.1: DHT11 Sensor Circuit

sensor DHT11. This sensor is also factory calibrated², making it easy to interface with other microcontrollers. Moreover, it is capable of measuring temperatures ranging from 0 °C to 50 °C and humidity from 20% to 90%, providing an accuracy of ± 1 °C and $\pm 1\%$. The values of the temperature and humidity are outputted as serial data by the sensor's 8-bit microcontroller, in pin No 2.

As shown in the Figure 2.1, the data pin is connected to an I/O pin of the MCU and a 5K Ohm pull-up resistor is used. This data pin outputs the value of both temperature and humidity as serial data.

To detect water leaks, in our scenario, small water droplets near the equipment's outlet, we chose a simple water detector sensor that is attached to the valve of the mechanism. It has a 3-pin module that outputs an analogue signal, indicating the approximate depth of water submersion. Overall, the more water it gets, the more conductivity and the voltage increases. When the sensor is dry, it outputs zero voltage. The primary consideration was the sensor's wide coverage and easy placement near the mechanism's exit.

2.1.2 Embedded Systems

Embedded systems are compact and energy-efficient computers that are integrated into various mechanical or electrical systems. These systems are characterized by their low cost and power consumption. Typically, an embedded system consists of a processor, power supply, memory, and communication ports. These communication ports enable the exchange of data between the processor and peripheral devices[3].

MCU (microcontroller unit)

The primary requirement for selecting the MCU was its compatibility with integrating a Wi-Fi module or having built-in Wi-Fi capabilities, as the device needed to transmit data over the internet. We chose to utilize Wi-Fi instead of Bluetooth as our wireless communication protocol due to several advantages. Firstly, Wi-Fi offers a

²DHT11-Temperature and Humidity Sensor, components101, Available online: <https://components101.com/sensors/dht11-temperature-sensor>, Accessed: 08.05.2023

Microcontroller Comparison				
Model	ESP32 - S3 ³	ESP32 - S2 ⁴	ESP8266EX ⁵	ESP32 - C3 ⁶
Active Mode (mA)	91 - 340	68 - 310	56 - 170	95-240
Modem Sleep (mA)	13.2 - 107.9	10.5 - 32.0	15	18 -28
Light Sleep (uA)	240	750	900	130
Deep Sleep (uA) ⁷	7-8	20 - 190	20	5
Power Off (uA)	1	1	0.5	1
Wifi	yes	yes	yes	yes
Bluetooth	yes	yes	yes	yes
GPIO	34	43	17	22
CPU	dual-core 160MHz to 240 MHz	single-core 240 MHz	single-core processor that runs at 80/160 MHz	single-core 160 MHz
Price (€)	14.1	7.52	out of stock ⁸	8.46

Table 2.1: Comparison Between different MCUs from espressif family

greater range than Bluetooth, ensuring reliable communication between the pH filter units and the access point even when they are positioned at considerable distances from each other. This is particularly advantageous if the pH neutralization units are spread across a large area or if there are obstacles that could potentially disrupt signal transmission. Additionally, Wi-Fi networks are more prevalent and established in various environments, including homes, offices, and public spaces. This widespread availability makes Wi-Fi a suitable choice for our project.

One requirement of this project was the overall low cost of the system, since we are imitating an industrial project as closely as possible. The battery consumption of the selected microcontroller was another requirement, as the device's ability to successfully automate the water inspection process relied on its ability to operate for an extended period of time. During this phase, we encountered a constraint related to the time-sensitive nature of the project, which limited our choice of microcontrollers (MCUs) available in the market since they could not be obtained within the required timeframe.

The main options we took into account were part of the *espressif* family: the *ESP32 - S3*, *ESP32 - S2*, *ESP8266EX*, *ESP32 - C3* microcontrollers. However, we also considered MCUs from other manufacturers, such as the MSP430FR413x from *Texas Instruments*. The Tables 2.1 and 2.2 reflect the main criteria under analysis, prices consulted from Mouser Online Store (URL: <https://pt.mouser.com/>) at 10/03/2023.

MSP430FR413x analysis	
Model	MSP430FR413x ⁹
Active Mode (mA)	126 μ A / MHz
LPM0	158 - 427 μ A (20 μ A / MHz)
LPM3	1.25 - 1.99 μ A, 25°C (1.2 μ A)
LPM4	0.57 - 0.75 μ A, 25°C (0.6 μ A without SVS)
LPM3.5	0.70 - 1.25 μ A, 25°C (0.77 μ A with RTC only)
LPM4.5	0.013 - 0.375 μ A, 25°C
SHUTDOWN	13 nA
Wifi	-----
Bluetooth	-----
GPIO	14.1
Price (€)	17.49

Table 2.2: MSP430FR413x analysis

During the evaluation process, we primarily considered three key factors for selecting the board. First, the presence of a built-in Wi-Fi module was crucial as the board needed to connect to Wi-Fi networks, as this feature ensured wireless connectivity for our project.

Secondly, we paid close attention to the number of *General Purpose Input/Output (GPIO) pins* available on the board. This factor is important for the future expandability of the project, allowing us to connect a sufficient number of sensors and peripherals as needed.

Lastly, our main emphasis was on the power consumption during the deep sleep mode, which serves as the primary operational state of the board for extended duration. This mode was prioritized due to the device's minimal data transmission activities. A comprehensive explanation of how we effectively utilized this mode will be provided in the chapter 3 of this report. Additionally, we considered the power consumption during active mode, which occurs when the MCU collects and transmits data from the sensors.

By carefully considering these factors, we were able to make an informed deci-

⁴Data retrieved from ESP32-S3 Series Datasheet, Espressif, Available online: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf, Accessed: 10.03.2023.

⁵Data retrieved from ESP32-S2 Series Datasheet, Espressif, Available online: https://www.espressif.com/sites/default/files/documentation/esp32-s2_datasheet_en.pdf, Accessed: 10.03.2023.

⁶Data retrieved from ESP8266EX Series Datasheet, Espressif, Available online: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf, Accessed: 10.03.2023.

⁷Data retrieved from ESP32-C3 Series Datasheet, Espressif, Available online: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf, Accessed: 10.03.2023.

⁸The values mentioned above may vary, as the activation of certain functionalities of the MCU during deep sleep mode can have an impact on power consumption.

⁹Data retrieved from MSP430FR413x Series Datasheet, Texas Instruments, Available online: <https://www.ti.com/lit/ug/slau595b/slau595b.pdf?ts=1684329937122>, Accessed: 10.03.2023.

sion regarding the selection of the board for our project.

The ESP32 - S3 is a high-end MCU that offers an impressive array of features, including a built-in Wi-Fi module, low power consumption, and a powerful processor. However, it comes with a relatively high price tag. While the ESP32 - S3 provides a wide range of possibilities for our project, one of the requirements is to maintain awareness of the final product's cost-effectiveness. Considering the overall economy of the project, we opted against choosing the ESP32 - S3 due to its higher cost.

The ESP32 - S2 is a microcontroller unit (MCU), which despite having a slightly less powerful processor compared to some other options like the ESP32 - S3, it is able to handle low CPU intensive tasks, as readings from sensors and computing basic mathematical operations. One notable feature of the ESP32-S2 is its built-in Wi-Fi module. The MCU offers the advantage of low power consumption, making it a suitable choice for energy-efficient applications. Furthermore, the ESP32-S2 was directly provided to us, by our institution, offering an advantage since this project is time-dependent.

The ESP8266EX board stands out as a choice for our project due to its low power consumption, despite having a less powerful processor compared to the other options. Its built-in Wi-Fi capability is also a valuable feature. One drawback of the ESP8266EX is its low number of GPIO (General Purpose Input/Output) pins comparing to the other options. This limitation could potentially restrict the number of sensors that can be added to the project, thereby limiting its expandability in the future. Although we were unable to select this MCU for our project at this stage due to stock shortage, it is worth mentioning as a notable competitor that we considered.

The ESP32 - C3 is a potential candidate for our project. It has a combination of features, including low power consumption, a built-in Wi-Fi module, and similar characteristics to the ESP32-S2. Although it runs on a slightly less powerful CPU, it delivers excellent deep sleep consumption. However, the ESP32-C3 is slightly more expensive compared to the ESP32-S2.

The MSP430FR413x microcontrollers has superior power consumption, making it an energy-efficient choice. However, it lacks a built-in Wi-Fi module, which presents a challenge. To incorporate Wi-Fi functionality, an additional purchase and installation would be required, adding complexity and potentially increasing costs. Considering our project's time constraints and the readily available access to the ESP32-S2, this option was ultimately dismissed due to the fact that the ESP32-S2 offered a safer solution.

Considering the project's time constraints and the availability of the ESP32-S2, along with the cost and additional requirements associated with the MSP430FR413x line-up, we have opted to prioritize convenience. As a result, we have chosen the ESP32-S2 as the preferred platform for our project.

In conclusion, the ESP32-S2 emerged as the most suitable and practical option for our project, due to its built in Wi-Fi module and immediate availability. Figure 2.2 shows the version of the chosen microcontroller.

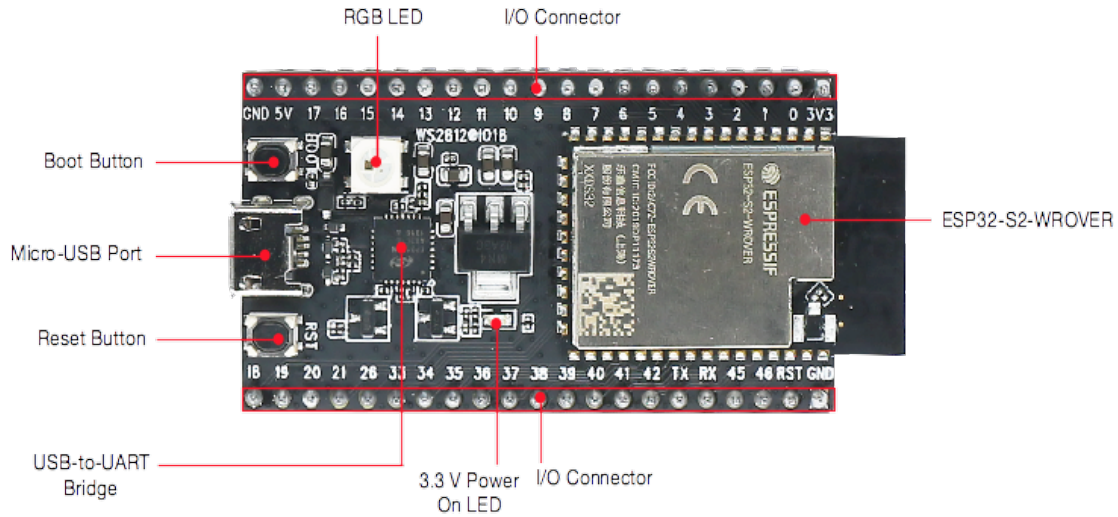


Figure 2.2: Espressif ESP32-S2-SAOLA-1M (WROOM)

MCU Battery

Three parameters were considered while selecting the MCU battery: the operating voltage compatibility with the ESP32-S2, the cost of the battery and the battery capacity. The MCU operates under 2.8V to 3.6V. Therefore, we considered the Li-Ion 18650 3.6V 3350mAh Ø18.2x65mm - INR18650 battery.

Framework

We made the decision to utilize the ESP-IDF (Espressif IoT Development Framework) framework, the development framework for Espressif SoCs supported on Windows, Linux and macOS, instead of the more commonly used solutions Arduino and PlatformIO frameworks for programming the MCU.

This choice was motivated by the project's requirement for an economic power device, as highlighted in the section 1.2. The ESP-IDF framework provides us with direct access to the underlying hardware and peripherals, giving us more control and flexibility over the microcontroller. This allows us to optimize performance in a variety of tasks.

Another advantage is the freely extensive documentation available. On the other hand, many tasks, for instance connecting to the Wi-Fi or interacting with the sensors, becomes a more complex task due to the non-existent abstraction that other frameworks provide.

2.1.3 Backend

In an IoT system, the backend serves as the component responsible for managing and processing the data collected from connected devices. The main functions of the backend in an IoT system include:

Technology Stack

In the backend development of our IoT project, we adopted a technology stack consisting of the Spring framework and the Kotlin programming language.

The Spring framework served as the foundation of our backend implementation. We leveraged features such as MVC architecture, transaction management, and integration with other components[4]. The use of this framework provided us with a scalable and flexible infrastructure for building our backend services.

Kotlin, a modern and expressive programming language, was chosen as the primary language for our backend development. This decision was based on a thorough evaluation of its features and benefits. Kotlin's concise syntax enabled us to write code that is more succinct and easier to comprehend, and, simultaneously, not as verbose as Java[5]. The built-in null safety feature significantly reduced the occurrence of null pointer exceptions, a common source of bugs in other languages. Additionally, the rich standard library provided a wide range of utility functions and classes, enabling us to implement complex functionalities more efficiently.

Due to our prior exposure to and experience with both of these technologies throughout our course, we have developed a strong familiarity with them. This familiarity significantly influenced our decision to incorporate Spring and Kotlin into our project's tech stack.

Database

To store the data, we decided to separate the data in two sections: The non-time-dependent data, which encompasses the portion of data within a database that remains constant and undergoes minimal or no changes over time or during system operation, such as information about the users, error logs and associated devices. The main considerations in selecting a technology for storing this type of data were scalability and support for intermediate querying capabilities. The open-source relational database PostgreSQL emerged as an ideal choice that fulfilled these criteria. Furthermore, our prior exposure to with this technology during our course expedited the development process of this module, enhancing overall time efficiency.

The sensor data, which is derived from the continuous collection of data by the device's sensors, such as pH values, is characterized by its time-stamped nature and represents a continuous stream of values or events. With this design in mind, we chose to utilize a *time series* database to store all the data related to the sensor records.

Time series databases are specifically designed to store and retrieve data records associated with timestamps (time series data), due to the uniformity of time series data, specialized compression algorithms can provide improvements over regular compression algorithms designed to work on less uniform data. They offer faster querying and can handle large volumes of data[6]. However, it's important to note that these databases may not perform as effectively in domains that do not primarily work with time series data.

For our project, we made the decision to utilize the *InfluxDB* time series database,

a widely recognized and used time series database solution[7]. *InfluxDB* offers a range of advantages that we previously discussed, and it integrates with Kotlin, the server-side language we have adopted.

2.1.4 Communication Protocols

After careful consideration, we opted for the MQTT protocol to allow the communication between the device and the server. MQTT is extensively utilized in the IoT industry due to its lightweight and efficient nature[8]. This efficiency enables devices to conserve energy more effectively in comparison to other protocols such as HTTP[9]. MQTT achieves this by employing a smaller packet size and lower overhead than its counterparts.

To allow communication between the web app client, we used the HTTP protocol. Our application provides an API (Application Programming Interface) that allow other applications or services to interact with them. APIs are typically based on HTTP and use various HTTP methods (such as GET, POST, PUT, DELETE) to perform operations like retrieving data, creating resources, updating resources, or deleting resources. API clients make HTTP requests to these endpoints to interact with the web application and exchange data.

2.1.5 MQTT Broker

One of the fundamental components of the solution is the presence of a broker. In a IoT system, data is consistently transmitted from the devices to the backend server. Therefore, it is important to select a communication protocol that is both lightweight and offers a dependable communication channel.

The role of the broker in the MQTT protocol is to serve as an intermediary, facilitating the transmission of data from publishing clients to subscribing clients. In the specific context of our project, the publishing clients are represented by various MCUs (Microcontroller Units), while the subscribing client is the central server tasked with analysing the data.

To implement the broker, we have decided to utilize the free version of the HiveMQ CE. This selection aligns with our requirements, particularly our focus on message security.

The *HiveMQ* broker supports TLS/SSL encryption, ensuring secure communication between *HiveMQ* and MQTT clients (both publishers and subscribers). Additionally, it provides support for handling substantial amounts of data and is compatible with Kotlin. While the free version of *HiveMQ* does have limitations, such as a maximum allowance of 100 devices, it is deemed sufficient for the scope of our project.

2.1.6 MCU Configuration

To facilitate interaction between the MCU and the equipment owner, we decided to adopt a solution provided by the manufacturer of the ESP32-S2. The ESP Touch protocol is a wireless communication protocol developed by *Espressif Systems*

specifically for their *ESP8266* and *ESP32* series of Wi-Fi enabled microcontrollers. The free ESP Touch mobile app uses this protocol to connect to the ESP device, and thus, transmitting, not only the Wi-Fi configuration, but also a string of bytes of custom data. Another advantage of this solution is the built-in encryption, using WPA2 (Wi-Fi Protected Access 2), which is relevant since Wi-Fi credentials are sensitive information.

2.1.7 Frontend

The frontend refers to the client-side part of a software application that users interact with directly. It is responsible for presenting the user interface (UI) and handling user interactions, such as inputting data, submitting forms, and receiving and displaying information. In our project, we have developed a user-friendly frontend interface that offers users visualization of the collected sensor data, in graphical form.

To develop the frontend, we had to select a web app client framework. Given our previous experience with React, a widely adopted framework for building front-end applications, we opted to utilize it in our project. It is, also, a framework which provides modularity, allowing to reuse the same components in different parts of the source code, reducing possible bugs. The majority of the website's styling was accomplished using pure CSS.

To streamline our development process, we integrated Webpack, a free and open-source module bundler for JavaScript[10]. We also took advantage of TypeScript, which provides benefits such as type checking, enhanced code editor support, and improved code documentation. Webpack was configured to employ a TypeScript loader responsible for transpiling TypeScript files into JavaScript, thereby ensuring compatibility with various browsers.

To render smooth graphs, which display collected sensor data, we opted to use an open source JavaScript charting library: Chart.js. This removes the need to build from scratch a fully functional graph component, allowing us to concentrate in other required tasks.

By adopting these technologies and tools, we successfully constructed a visually appealing and interactive front-end interface while leveraging the advantages of React, Webpack, TypeScript, and third party JavaScript modules.

2.2 Existing Solutions

CropX specializes in delivering IoT-based soil monitoring systems that incorporate pH sensors¹⁰. Their extensive experience in the market has allowed them to provide real-time data on soil conditions, empowering farmers to make informed decisions regarding irrigation and nutrient management. The fundamental principles of this industrial solution align closely with our own project, as both aim to equip users

¹⁰Field data management know to grow, CropX, Available online: <https://cropx.com/cropx-system/field-data>, Accessed: 10.03.2023

with valuable insights derived from data. They also provide an intuitive UI (User Interface) for the user to analyse the data.

In 2022, a project was developed by Bernardo Marques, under the guidance of Professor Rui Duarte. This project is called SmartDoorLock-IoT. It consists of a system composed by a microcontroller, in particular an ESP32-S2, that hosts a socket based server and an Android application that interacts with the MCU/Server to lock and unlock a door. Behind the curtains, the system uses security features, like encryption through symmetric and asymmetric keys, Hash, MAC, databases, and Wi-Fi communication. Using an IoT device, the system is able to take advantage of features accessible in microcontrollers, for instance the ultra low power mode, available in the ESP32-S2. Overall, this project/system takes advantage of the IoT technology to implement a secure IoT system capable of simplifying the life of a human being.

In addition to our project, we conducted an analysis of other similar works that aim to address similar problems. One such project that caught our attention is the study titled “Devising an IoT-Based Water Quality Monitoring and pH Controlling System for Textile ETP”. This study serves as a relevant reference for our project, as it shares similarities in terms of implementing an IoT-based system and analysing pH levels. The proposed automatic system incorporates a monitoring device and a pH-controlling device to ensure real-time data transmission, online and offline monitoring capabilities, and notifications for abnormal water quality situations[11].

In our research, we also came across a study that focuses on monitoring pH and total dissolved solids (TDS) in the context of aquaponics, a cultivation technology combining fish farming and plant growth. The research titled “Fish Feeding Automation and Aquaponics Monitoring System Base on IoT” shares similarities with our project in terms of monitoring pH and automation. The study developed a monitoring system using an Android application to monitor pH and TDS levels in real-time. Additionally, fish feeding was automated based on a predetermined schedule and feeding requirements. IoT technology was utilized for communication purposes[12].

2.3 Summary

This chapter serves as a foundation for understanding the components and technologies used in the proposed architecture. Specifically, the ESP32-S2 was chosen as the MCU, accompanied by selected sensors. The server-side implementation will utilize Spring and Kotlin, while the front-end development will utilize React. To persist the data, we have opted for PostgreSQL and InfluxDB.

During our research, we encountered various related works in this field; however, they presented certain limitations. Industrial solutions, like CropX, were identified as expensive and lacked the desired adaptability for our specific application. On the other hand, academic studies explored similar concepts but fell short in terms of adaptability and customization for pH monitoring in our specific use case. These findings further underscored the necessity and significance of our project, as it addresses the existing gaps and offers a viable solution tailored to our requirements.

3

Proposed IoT System Architecture

This chapter provides the proposed system architecture. It begins with an introduction to the system overview, which includes a description of the main components and the interaction between them. Subsequent sections focus on each individual component, highlighting their respective roles in the system and providing detailed explanations of their implementation.

3.1 System Overview

It is possible to divide the system into the following sections:

1. **IoT Sensor Node:** collects the data from the neutralization device;
2. **Backend:** receives and processes the data emitted by all the registered MCUs;
3. **Web Application:** allows the user to register devices and analyse device-collected data.

As shown in Figure 3.1, data flows from left to right. All begins with the MCU collecting data from the sensors, which is then transmitted to the *Backend Broker* module. The backend broker serves as a middleman between the IoT Node and the application server. The application server receives the sensor data from the broker to process it, accordingly. Eventually, the application server might send an alert email to the manager of the neutralization mechanism if the data indicates an abnormal

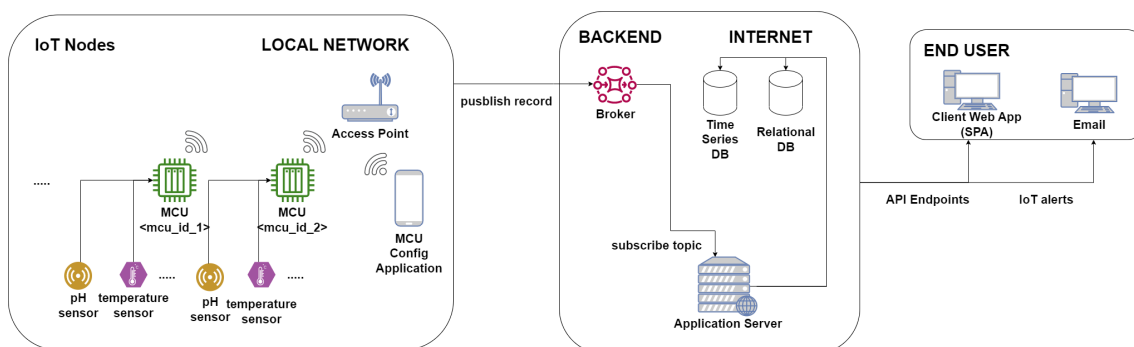


Figure 3.1: Main system architecture



3.2 IoT Node

The subcomponents have the following functions:

- With the objective of setting up the MCU for the first time, the owner of the mechanism that is associated with the MCU interacts with an android app provided by Expressif called Esp Touch to configure the MCU. This process consists in setting up a device identifier provided by the web application developed by us, in which the user must create an account beforehand, and setting the Wi-Fi network credentials. The Esp Touch application will be responsible for transmitting this data to the MCU via Wi-Fi. After, the MCU will instruct the sensors to collect data, and send it to the backend, through the access point.

In the majority of cases, the MCU operates in a low-power mode known as "deep-sleep." In this mode, the CPUs, most of the RAM, and digital peripherals that are clocked from the APB_CLK (Advanced Peripheral Bus Clock) are powered off[13].

This low-power mode is maintained throughout most of the day, as continuous readings are not required due to the nature of the neutralization mechanism. Changes in conditions occur over longer periods of time. Given the significance of power consumption, every second that the MCU remains active has a notable impact.

The MCU was programmed to be susceptible to hard resets, which can occur when it is in a sleep state. During a hard reset, the MCU's unique identifier and Wi-Fi credentials are erased, leading to a loss of configuration. In such cases, or when the device has not been initially configured, it will remain idle, waiting for proper configuration to be provided.

Upon waking up, the device performs an analysis to determine the cause of its wake-up. If the device has been powered on, it proceeds to calculate the calibration times for each sensor. These calibration times denote the duration that each sensor needs to wait after being turned on before initiating the measurement process. A detailed explanation of this process is provided later in the section 3.2.4.

Typically, the ESP will re-start execution after aborting from an exception that was not handled. When this happens, the device reports the error cause to the backend, and goes into deep-sleep mode again. It is important to collect all the error logs since they provide relevant information about the state of the equipment. After an error occurs in the system, the error information will be sent to the backend to be stored.

The standard programmed behaviour of the MCU can be thought of a cycle, which is divided by two iterations, intercalated by a 6-hour deep sleep mode. Essentially, the MCU will wake up to perform one of two actions:

- The first involves the MCU utilizing all of its sensors to collect readings, including the pH sensor, the DHT11 and the water sensor. First, all sensors are used to make readings and only then, the Wi-Fi is powered on with the objective of transmitting the readings to the broker. This iteration is the one that consumes the most energy, since the Wi-Fi will be used to send the readings to the broker.
- In this second iteration, the MCU checks for a possible water leakage, in the neutralization device. It accomplishes this by using a water sensor to detect the presence of water and subsequently sending an alarm to the backend if water is detected. Note that the Wi-Fi is only used in case water is detected, and thus, this iteration should consume less energy than the previous described one, which always uses Wi-Fi to transmit readings. This cycle typically consumes less energy compared to the previous cycle, since Wi-Fi is not actively used when no water leakage is detected. Additionally, the calibration time for the water sensor is relatively shorter than that of the other sensors. As a result, this cycle can be repeated more frequently without significantly impacting energy consumption, thereby enabling better control over potential water leakage events. Additionally, the calibration time for the water sensor is relatively shorter than that of the other sensors. As a result, this cycle can be repeated more frequently without significantly impacting energy consumption, thereby enabling better control over potential water leakage events.

It is worth mentioning that in situations where Wi-Fi is unavailable or some other issue, at the time of data transmission, the sensor readings will be stored in the device to be transmitted during the next internet connection. If the device wakes up to make new readings and still can't send the old readings, the new readings will override the ones already stored. This approach was chosen to optimize battery usage by sending smaller payloads, avoiding the possibility of transmitting several old readings that may not have much significance to the system monitorization. Since the primary objective of the system is to actively alert users, it is deemed that the most recent reading holds the utmost importance, rendering previous readings obsolete.

After the MCU completes its task (one of the two described above), it will enter deep sleep mode, and then wake up to make the other one.

3.2.2 Sensor Modules

The primary objective of this project is to develop a system that automates the inspection process for potential issues in a neutralization mechanism, but is also interesting to develop a system that is capable of providing us with meaningful insights. Taking that into consideration, we opted for the following sensors:

1. A pH sensor at the entrance of the filter system (providing data about the liquid before being neutralized);
2. A pH sensor at the end of the filter system (providing data about the liquid after being neutralized);
3. Ambient temperature sensor;
4. Air humidity sensor;
5. Water flow sensor to estimate the amount of water that flows in the system (not implemented due to project's time constraint);
6. Water detection sensor to alert if water has dripped from the discharge neutralization device exit;

The purpose behind the installation of two pH sensors, one at the filter's entrance and another at the exit, is to gather data on the product's efficiency. By comparing the pH levels, the manufacturers of the equipment can estimate the effectiveness of the system.

During the project's planning phase, we identified a potential constraint regarding the availability of physical sensors. This raised concerns about our ability to conduct comprehensive system testing. To overcome this challenge, we devised a solution by implementing mock implementations for the various "Sensor Reader" modules. These mock implementations were designed to generate synthetic sensor data, enabling us to evaluate the system's functionality in the absence of physical sensors.

Consequently, for each sensor reader component, two implementations were developed: a real implementation that interacts directly with the physical sensor, and a mock implementation that simulates the sensor procedure. This approach provided flexibility and independence from the availability of the physical sensors, allowing us to continue testing and developing various components of the system regardless of the sensor's presence.

3.2.3 Sensor readings - Parallelized vs Serialized

The approach for reading from the sensors was taken into consideration. One option is to start all the sensors simultaneously and parallelize the reading process, while the other option is to read each sensor individually and serialize the reading operations.

If we opted for parallelized reading, the main advantage is that the microcontroller would read from all the sensors at the same time. This means that the total active time of the microcontroller would be determined by the sensor that takes the longest to complete its operations. Consequently, the microcontroller can enter sleep mode faster, conserving more power.

However, parallelized reading may introduce challenges such as unstable or unreliable operation of the sensor, since all the sensors are activated at the same time, they may collectively draw a higher current, potentially exceeding the power supply capacity or putting unnecessary strain on the power source. It would also require more complex synchronization mechanisms.

On the other hand, serializing the reading operations by individually starting and reading each sensor can help mitigate these challenges. It reduces the likelihood of interference or conflicts between the sensors. Although this approach may require more time for data acquisition compared to parallelized reading, it offers greater control over power consumption, since it allows for selective activation of individual sensors and can ensure more reliable and accurate measurements from each sensor due to the lack of conflicts between the sensors. After evaluating these advantages and disadvantages, we opted for serializing the readings of the sensors.

3.2.4 Sensor Calibration

The sensor calibration involves determining the required waiting time for a sensor after it is turned on before it can start collecting accurate data from the environment. This waiting time is commonly referred to as the stabilization time. During this period, the sensor adjusts and stabilizes its internal components to ensure reliable and consistent measurements.

To address the variability of sensor stabilization time in different environments, we developed a method to calculate the required waiting time for each sensor independently, regardless of its specific location.

The approach we employed to determine sensor stabilization is based on the concept of standard deviation. Standard deviation measures the average deviation of individual values from the mean. A higher standard deviation indicates that the values are more dispersed and further from the mean, indicating that the sensor is not yet stabilized. Conversely, a lower standard deviation suggests that the values are closely clustered around the mean, indicating that the sensor readings are approaching stability.

To establish our strategy, our initial goal was to determine the standard deviation of the sensor readings when it reaches a stable state. This required conducting experimental analysis of the sensor's behaviour. We powered on the sensor and



Figure 3.3: Sliding window algorithm ¹

recorded the corresponding ADC values it produced every second. By observing the readings, we identified the point at which the sensor reached stability, characterized by low variability in its outputs, indicated by a low standard deviation.

To determine when the sensor reaches a stabilized state based on the experimental data, we employed an algorithm known as the sliding window. This algorithm involves creating a virtual window of a fixed size, denoted as 'x', which slides through the array of ADC values obtained from the sensor. At each iteration, the algorithm calculates the standard deviation of the values within the window.

By applying the sliding window algorithm, we can track the changes in the standard deviation as the window moves through the ADC value array. Initially, when the sensor is not yet stabilized, the standard deviation will be relatively high due to the variability in the readings. However, as the window progresses and includes more stable readings, the standard deviation will gradually decrease. In Figure 3.3 we can see a visual representation of how this algorithm works.

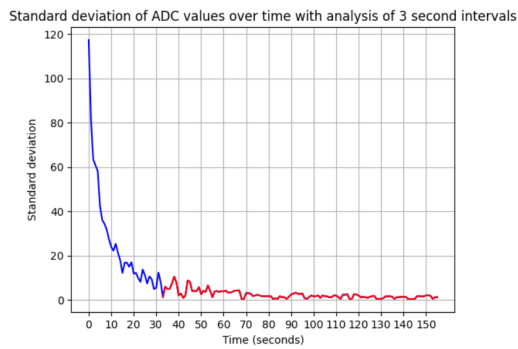
Once we observe that the standard deviation consistently remains low, we can infer that this particular standard deviation corresponds to a stabilized sensor. However, a question remains: what should be the size of the data window that we analyse?

The choice of the window size in the sliding window algorithm is necessary to accurately assess the stabilization of the sensor. If the window size is too small, it may not capture sufficient data points to provide a representative estimate of variability. Consequently, the calculated standard deviation within such a small interval might be misleading, potentially indicating stabilization at an early stage. However, this would not be a reliable indication, since the limited data within the small window may not be truly representative of the overall behaviour of the sensor.

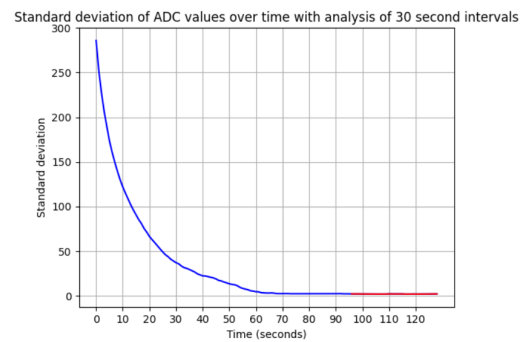
On the other hand, if a larger window size is used, the calculation of the standard deviation may take longer due to the increased number of data points being considered. It becomes more challenging to achieve a low variability in a larger array of data, prolonging the time required for the sensor to exhibit stable readings. This implies that a longer waiting time would be necessary for the sensor to reach a desirable level of variability within the analysed data window.

Therefore, finding an optimal window size consists of striking a balance be-

¹Image taken on June 15, 2023, from <https://medium.com/@ashley-peacock/advent-of-code-2022-day-6-solution-tuning-trouble-5500951d6e59>



(a) Window Size: 3



(b) Window Size: 30

Figure 3.4: Standard Deviation over Time

tween capturing sufficient data for reliable analysis and minimizing the waiting time for stabilization.

Taking as example the pH sensor:

Through empirical observation under experimental conditions, we determined that the pH sensor exhibits a stable state when the standard deviation of its ADC values approaches a value of approximately 2.

To determine the appropriate window size for calculating the standard deviation and assessing sensor stabilization, we conducted experiments using various window sizes. By analysing the standard deviations of the ADC values received by the MCU, we aimed to identify the point at which the values reached a standard deviation of 2, indicating sensor stabilization. Through this iterative process, we evaluated the effects of different window sizes on the accuracy and timeliness of detecting sensor stabilization

The graphs represented in Figure 3.4 depict the standard deviation over time obtained using two different window sizes, 3 and 30. In each graph, the standard deviation values are plotted, and the line turns red when the sensor is considered stabilized, meaning that the standard deviation is equal to or less than 2. These graphs provide a visual representation of how the sensor's stability evolves over time for different window sizes, allowing us to observe the importance of choosing an indicate size for the window.

By analysing these graphics, we can visually observe the trend that larger window size (30) require a longer time than is necessary to time for stabilization, while smaller window size (3) result in faster stabilization. However, with smaller intervals, the sensor may not be fully stabilized at that specific time, as we can observe in Figure 3.4a.

Knowing this propriety, we tested different window sizes and registered the respective time for stabilization, comparing to the window size, as shown in the Table 3.1.

After conducting the analysis, we have chosen a window size of 5. This decision was made based on the observation that window sizes 10, 15, and 20 exhibited similar values for the time to stabilize. By visually inspecting the graph, we can confirm that the values tend to stabilize at around 74 seconds. In our experiments, we found that a window size of 5 strikes a balance between not causing excessive delay in the

Table 3.1: Window Size vs. Time to Stabilize

Window Size	Time to Stabilize (seconds)
3	33
4	42
5	74
10	75
15	73
20	71
25	99
30	96
35	96
40	93
45	93
50	95

Standard deviation of ADC values over time with analysis of 5 second intervals

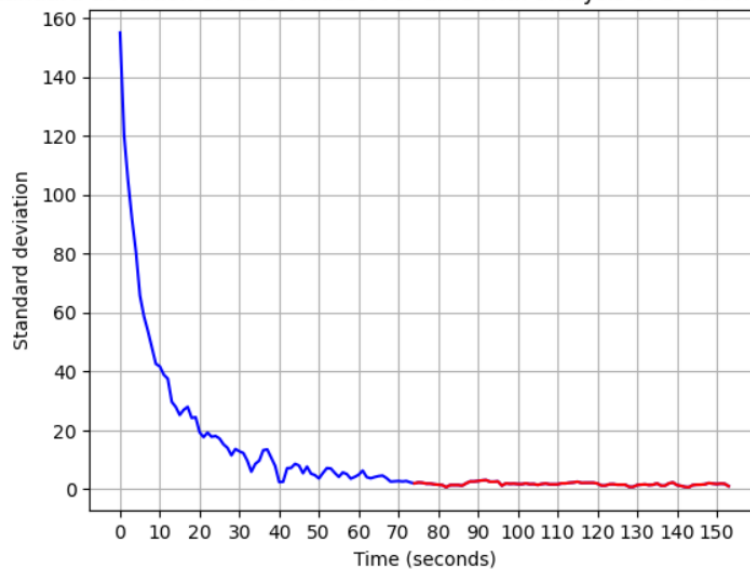


Figure 3.5: Standard Deviation over Time

stabilization process and still being representative of the system's variability. We can visually observe the stabilization of the sensor in Figure 3.4a, where the red line indicates the point at which the standard deviation of a five-element interval reaches or falls below 2. This signifies that the sensor has achieved stability.

With a fixed window size of 5, the calibration process of the pH sensor commences is the following: When the MCU is powered on, it initiates the calibration process by sequentially activating each sensor. During the calibration of the pH sensor, it collects readings from the environment at a frequency of one reading per second. To calculate the standard deviation, a sliding window algorithm is employed. Once the standard deviation of a 5-element interval reaches a value that is equal to or lower than the reference value, it indicates that the sensor has stabilized. Therefore, the time at which this stabilization occurs can be considered as the required time for the sensor to stabilize. This time will be stored persistently in the device for future readings.

To address the variation in environmental conditions and account for different scenarios, we adopted a more conservative approach in determining the stabilization of the sensor. We increased the reference value of the standard deviation by fifty percent. Although this value is significantly higher than the standard deviation in stabilized conditions, it provides some tolerance to accommodate the variability of different environments. In a real-world application scenario, we consider a standard deviation of 3 as an indicator of sensor stabilization. However, this value may not be entirely accurate if the environmental differences in the filter mechanism significantly deviate from the experimental conditions. This standard deviation reference value is stored persistently in the device. This approach is applied for every sensor that requires calibration time.

3.2.5 Sensor For PH

To deploy the pH sensor, it was first necessary to make experimental readings using this device. The ESP32-S2 includes ADC (Analogue to Digital Converter) channels to make analogue readings. This input value, the ADC, is a value with 13 bits range, which means is able to read values between 0 and 8191. Zero means null voltage, while 8191 is the maximum voltage the MCU is able to read.

The ADC units in the ESP32-S2 support configurable attenuation settings to handle different voltage levels. When higher attenuation is set, the ADC's input voltage range is increased, allowing measurement of larger input voltages.

According to the documentation, 11dB (max) allows the chip to read from 0 - 2500mV[14]. This means, that with an ADC value of 13 bits, and an attenuation of 11dB, the ADC is almost always sensible to voltage change between 0 and 2.5V, excluding extremes as with voltages near 0V and 2.5V. In reality, the values read using the pH sensor, never reach these extremes, as confirmed through experimental tests, and so, it imposes no problem to the readings.

Since the pH sensor outputs 5V (maximum), and the MCU analog channels should read no more than 3300mV, as advised in official documentation, we used a voltage divider to downgrade the analog input voltage on the chip, to 2500mV (the maximum voltage that can be read by the MCU input analog channel).

The formula for calculating the output voltage is:

$$V_{out} = V_{in} * \frac{R2}{R1 + R2} \quad (3.1)$$

To achieve a maximum output voltage (V_{out}) of 2.5V while operating with a maximum input voltage (V_{in}) of 5V, we carefully selected two resistances with equal power ratings. Our selection was guided by the consideration of minimizing current flow to conserve system energy, while simultaneously avoiding excessively high resistance values that could introduce signal interference, since the weaker signal can be easily overwhelmed or distorted by the interfering signals, leading to inaccuracies or disturbances in the system's operation.

We opted for 47kOhm for the value of both resistor($R1$ and $R2$). $R1$ is the resistor connected to the input voltage (V_{in}), and $R2$ is the resistor connected to the ground

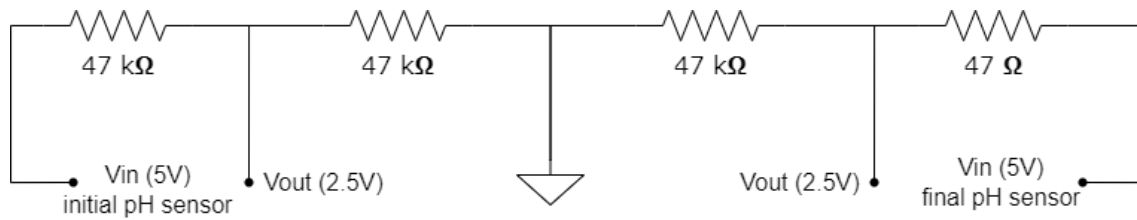


Figure 3.6: Voltage divider circuit used to achieve lower output voltage

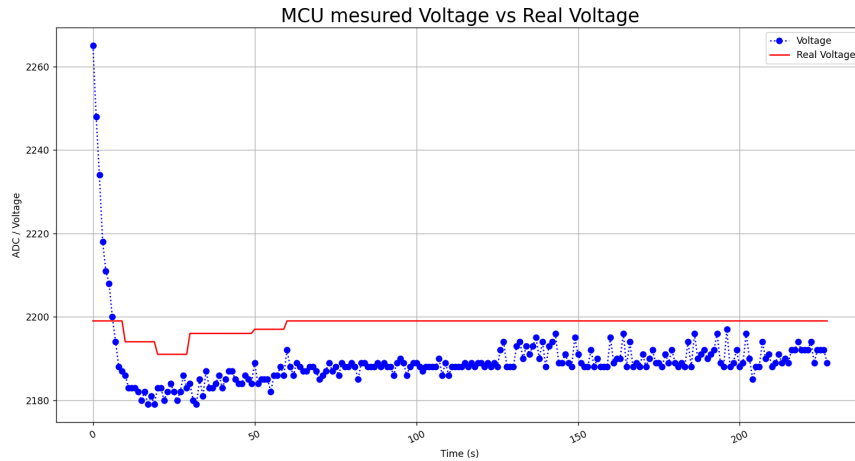


Figure 3.7: Graph illustrating the measured voltage with the MCU against the real voltage measured with an oscilloscope, of a liquid solution, pH equal to 8.

or reference point, as shown in Figure 3.6.

To interpret the ADC values obtained from the readings as pH values, we had the option of utilizing one of two different approaches to establish this correlation.

The traditional one, which consists in the analysis of the characteristic curve of the sensor, which establishes the relationship between voltage and pH. It would also be necessary to align the voltage measurements from the MCU with the reference values obtained from the external device. This calibration process would involve the analysis of the data present in the Figure 3.7, to adjust the voltage read from the MCU to the voltage read from the oscilloscope.

However, as we have liquids with known pH, we go directly from the ADC value to the pH value. According to the figure 3.8 the conversion between the ADC values and the voltage is almost linear, excluding both extremes. The voltage to pH transformation function is also linear.

Since the pH values are relatively linear over a certain range (between pH 2 to pH 10), we need two calibration points to determine the linear line, and then derive the slope of the line so that we can calculate any pH value with a given voltage output, and therefore, ADC value.

Knowing this, we experimented a solution liquid, measuring pH equal to 4 and read the correspondent ADC value in the MCU. The same was done with another solution measuring pH equal to 8.8.

PH deduction:

7955 ADC corresponds to pH 4.0

7184 ADC corresponds to pH 8.8

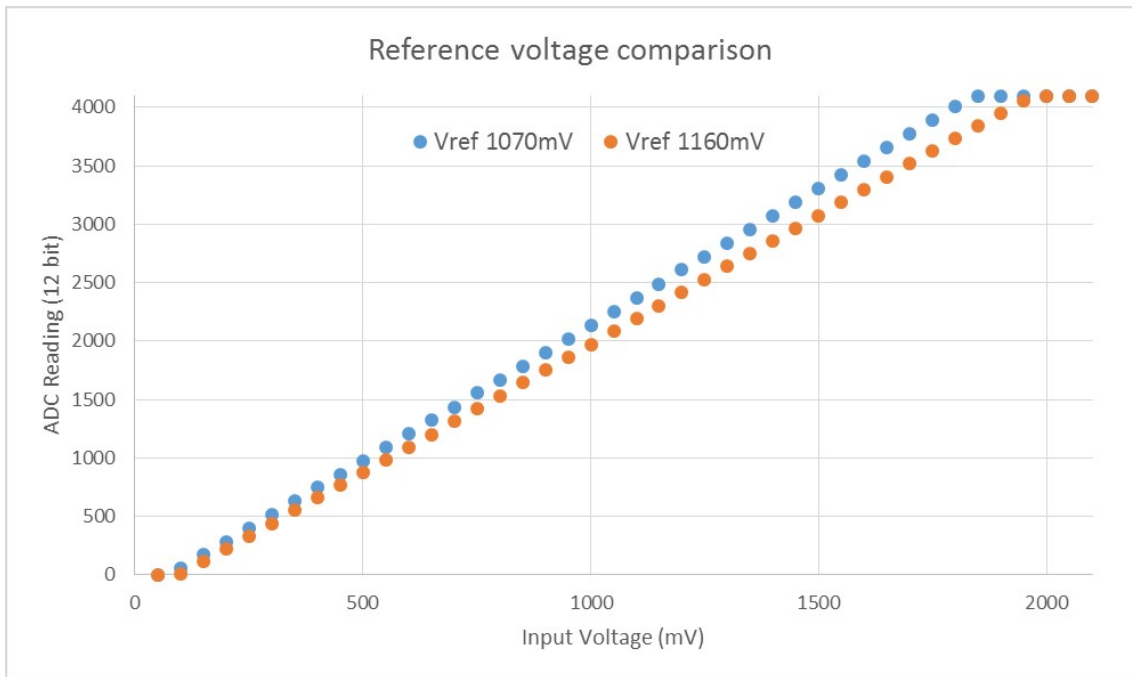


Figure 3.8: Graph illustrating effect of differing reference voltages on the ADC voltage curve.

$$m = \frac{8.8 - 4.0}{7184 - 7955}$$

$$4.0 = 7955 \cdot (-0.00622) + b$$

$$b = 4.0 - 7955 \cdot (-0.00622) = \pm 53.4801$$

$$ph = ADC \cdot m + b$$

$$8.8 = 7184 \cdot (-0.00622) + 53.4801$$

We have a high level of confidence in the accuracy of the pH values of the liquids used in our experiments, since we validated them through the use of a commercial calibrated pH meter.

With this, it is possible to trace the slope, allowing the conversion from ADC to pH. Even so, after powering up the pH sensor, it is necessary to wait for the sensor to stabilize, as the figure 3.9 shows. Starting from the data point at the 102 second (ADC: 7947), we observe a sequence of consecutive data points where the ADC values remain within the range of 7947 to 7953 (± 3 units). All the values after 102 seconds consistently fall within the defined stabilization criteria of ± 3 units. Therefore, it was decided that after 102 seconds, the reading is considered valid.

3.2.6 Sensor For Ambient Humidity And Temperature

To make the ambient humidity and temperature readings, the MCU component that handles this needs to respect the behaviour of the chosen sensor module, as described in the subsection 2.1.1.

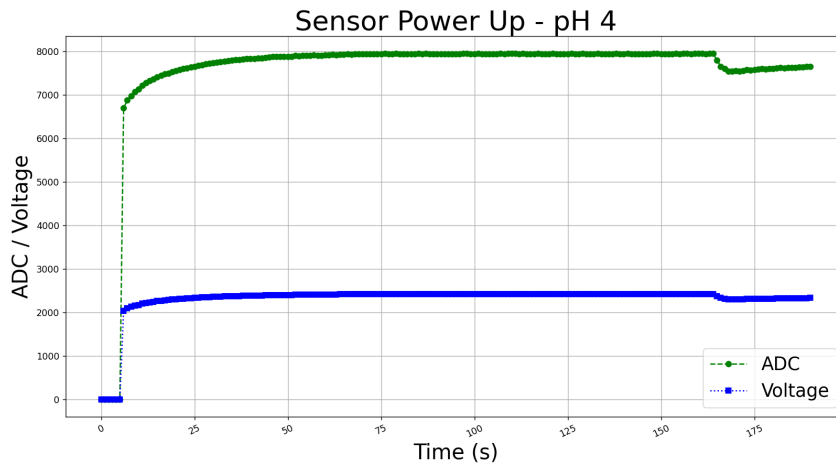


Figure 3.9: Graph illustrating the measured ADC and voltage of a liquid solution, pH equal to 4.

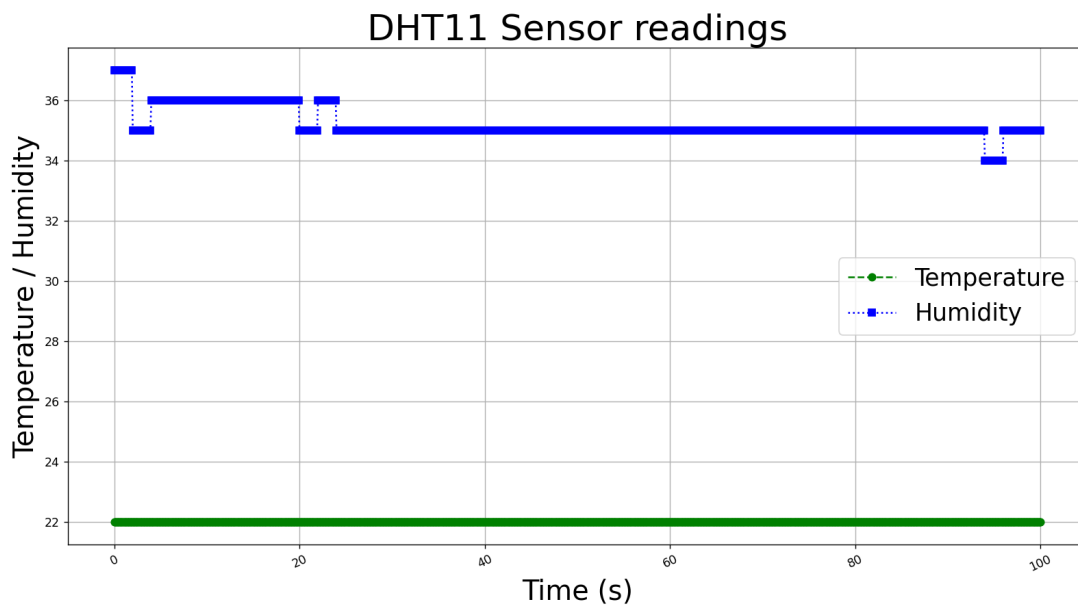


Figure 3.10: Graph illustrating the measured humidity and temperature, from with the DHT11 sensor, at room temperature.

During the development process, we conducted some research to identify existing implementations that could be leveraged, thus avoiding the need to write low-level code from scratch.

However, to determine if calibration was necessary, we conducted an experiment to verify if the values were close to the real environment condition. The test results indicated that after the 30 seconds to that took the sensor to stabilize, the values were correspondent to the room temperature and humidity, in Figure 3.10, the data relative to the reading of the sensor is presented graphically.

3.2.7 Sensor for water Leaks

Relative to the water leaks sensor presented in the subchapter 2.1.1, the sensor will output a certain voltage, each time it is in contact with water.

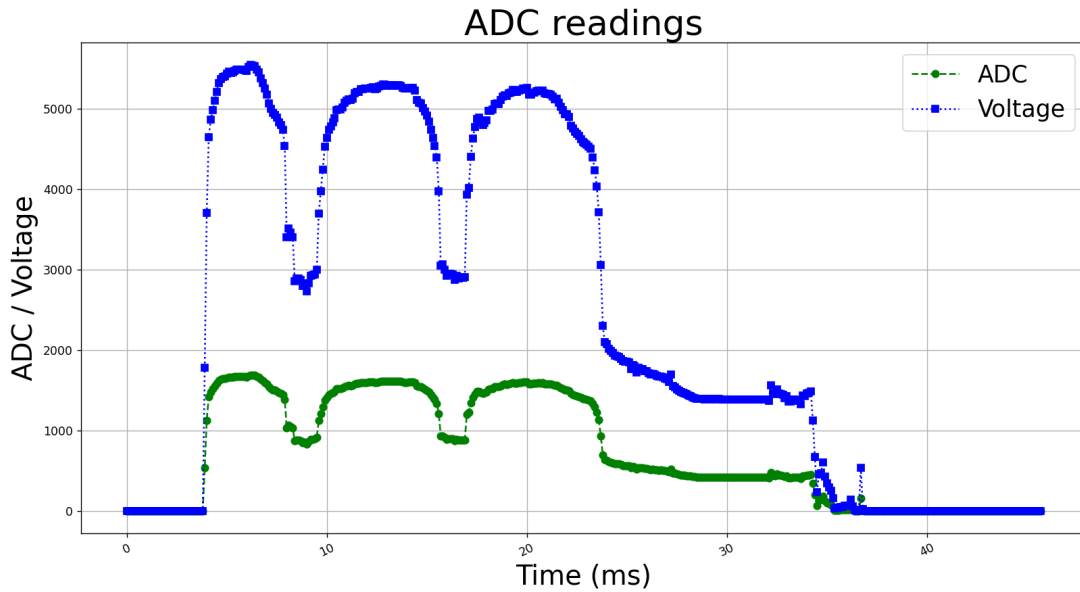


Figure 3.11: Graph illustrating the measured ADC and voltage when the sensor is immersed in water and taken out.

At first, we decided to use the ESP32-S2 functionality of waking up the microcontroller whenever an input pin receives the digital 1 value from the sensor. We identified two problems with this approach:

1. After conducting experimental tests, it was concluded that the module, even when fully immersed in water, only outputs 1.7V. This voltage level is insufficient for the MCU to recognize it as a digital logic high. The graph in Figure 3.11 illustrates the measured ADC and voltage when the sensor is immersed in water and taken out;
2. The sensor would have to be constantly on, resulting in greater energy consumption. This goes against the established requirements, where the hardware is supposed to be energy-efficient. Additionally, it is assumed that any water leakage in the system would occur at a very slow rate.

Hence, we made the decision not to keep the water leakage sensor actively monitoring for any potential leakages. Instead, the sensor would operate similarly to the other sensors. When the MCU wakes up from its deep sleep mode, it will read the sensor value. If the sensor detects any water leakage, the event will be reported.

3.2.8 Assembly

All sensors, except the pH sensor, can be powered on with 3.3V to operate. Since each GPIO can be used as digital, operating on 3.3V, when outputting digital one, they can be used as digital gates, for each sensor. Therefore, each one is connected to an individual GPIO. Using different I/O pins to supply each sensor independently offers two significant advantages. Firstly, it helps optimize power consumption by allowing each sensor to be powered on only when the chip requires their functionality. This reduces unnecessary power drain and contributes to overall energy efficiency. Secondly, this approach provides isolation in case of a circuit issue or mal-

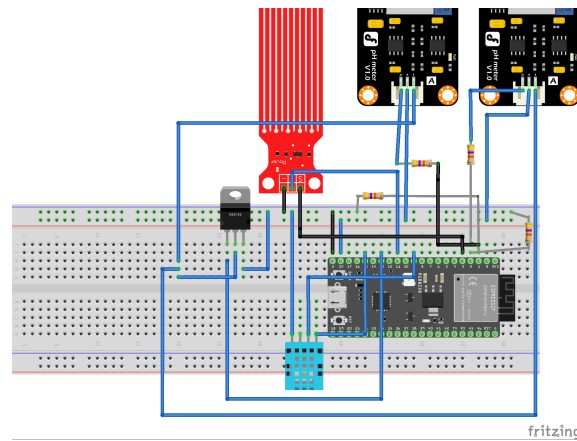


Figure 3.12: Current IoT Node assembly in the breadboard

function in any specific sensor. If a problem arises in one sensor, only that particular sensor will be affected, and the rest of the system can continue operating without interruption. This isolation allows for easier troubleshooting and maintenance, as the problematic sensor can be identified and addressed without impacting the functionality of other sensors or components. For the pH sensors, which require a total of 5V to function, we utilized the only PIN capable of outputting 5V. This specific pin was employed to supply power to both pH sensors. The issue with this particular pin is that it cannot be controlled to reduce power through software. As a result, the pH sensors would remain operational even when the MCU was not actively utilizing them, leading to unnecessary power consumption.

To address this issue, we implemented a solution using a transistor to interrupt the power supply when needed. Since both pH sensors had similar power stabilization timings and were used almost simultaneously, we decided to utilize a single transistor for both sensors. A separate GPIO pin was assigned to control the gate of the transistor, allowing us to effectively manage the power supply to the sensors. In terms of data connections, the organization is as follows:

- The initial and final pH sensors connected to the pin 3 and 4, so it's possible to read independent values from each sensor. Each of these pins has its own independent ADC channel assigned to it, the ADC1_0 and ADC1_1 channels, allowing to read analogue input voltage. Both pH sensors have their own resistances to lower down the output voltage, as explained previously;
- The water sensor, connected to the pin 3, also used as an analogue channel - ADC1_2;
- The DHT11, serving as the temperature and humidity sensor, which is digital, is connected to the pin 11 - GPIO 9, transmitting data in a series of bits.

The pin 10 - GPIO 8 - is being used to serve as an external wake up, when the ESP is sleeping. When the pin receives the digital value 1, it triggers a reset in the ESP configurations, when he wakes up. This pin is only used when the MCU is sleeping. Figure 3.12 depicts the schematics of our assembly.

3.2.9 Wi-Fi

To connect the MCU to the Wi-Fi, two properties are required: "SSID" and Password. The "SSID" and the password are the credentials used to connect to the Wi-Fi network. Firstly, the MCU tries to recover the Wi-Fi configuration and, if available, tries to connect to the Wi-Fi. If not, the MCU will initiate the *ESP Touch protocol* to allow users to utilize the *ESP Touch* Android app to transmit the Wi-Fi configuration to the MCU.

Once the user sends the credentials to the ESP32-S2, the data will be stored in the persistent memory, for later to reconnect to the same SSID. Through the Touch Android application, the user has also the ability to send a costume string, as described in the subsection 2.1.6, which in our project we define this string as the unique device "ID" that corresponds to an identifier exclusive for each microcontroller.

To incorporate the ESP Touch Protocol and Wi-Fi modules into our project, we referred to the SmartDoorLock-IoT project mentioned in section 2.2. This project had already implemented these modules, so we adapted them to suit our specific use case.

3.2.10 Time

The ESP32-S2 utilizes two hardware timers to maintain system time[15]. This functionality is particularly useful when the ESP needs to retrieve the current time, such as when reading data from sensors.

1. The RTC Timer, which is capable to persist time over deep sleeps, but is not so accurate;
2. High-resolution timer, which doesn't persist time between deep sleeps and will not persist when the device is powered off, but has a greater accuracy.

Since both sources lose track of time when the device is powered off, an external synchronization mechanism is required every time the MCU is powered on. This synchronization process is unavoidable and requires internet connection. We made the decision to use the RTC Timer as our preferred time source in the system. This choice was driven by the fact that time accuracy is not a critical requirement for our application. Additionally, considering that Wi-Fi transmissions have a significant impact on power consumption, it is not necessary to synchronize the time externally repeatedly. Our focus is on maximizing battery life, and therefore, using the RTC Timer provides sufficient timekeeping capabilities for our needs.

To ensure accurate time synchronization in our system, we utilize the Network Time Protocol (NTP). NTP is a widely used protocol designed for synchronizing the time across computer systems over a network. Its primary goal is to achieve high accuracy and reliability in time synchronization. This approach ensures accurate time-keeping while minimizing power consumption and network usage.

3.3 Backend

This component is shared by all the clients, and serves an IoT platform, supporting all IoT nodes scatter across the multiple clients using this system. It should be deployed in the cloud, so all IoT nodes have access to it. It is composed by the application server, the broker and databases.

3.3.1 MQTT: Topics And Representations

The broker acts as a mediator between the IoT nodes and the backend server, facilitating communication between them. The three topics used, and respective body representation example, are shown below:

1. The `sensor_record` is used to send a single sensor recording.

```
{
  "device_id": "AdPGFVs",
  "value": 8.0,
  "timestamp": 1656330216,
  "sensor_type": "temperature"
}
```

2. The `water_leak` is used to send an alert to the Backend, signalling a water leak in the neutralization device.

```
{
  "device_id": "AdPGFVs",
  "timestamp": 1656330216
}
```

3. The `device_wake_up_log` used by the MCU to report the reason for wake-up. Common values are deep-sleep timeout, exception and power-on.

```
{
  "device_id": "AdPGFVs",
  "timestamp": 1686330216,
  "reason": "power on"
}
```

4. `error_reading_sensor` used by the MCU to alert if there was a problem while reading any sensor.

```
{
  "device_id": "AdPGFVs",
  "timestamp": 1686330266,
  "sensor_type": "initial -ph"
}
```

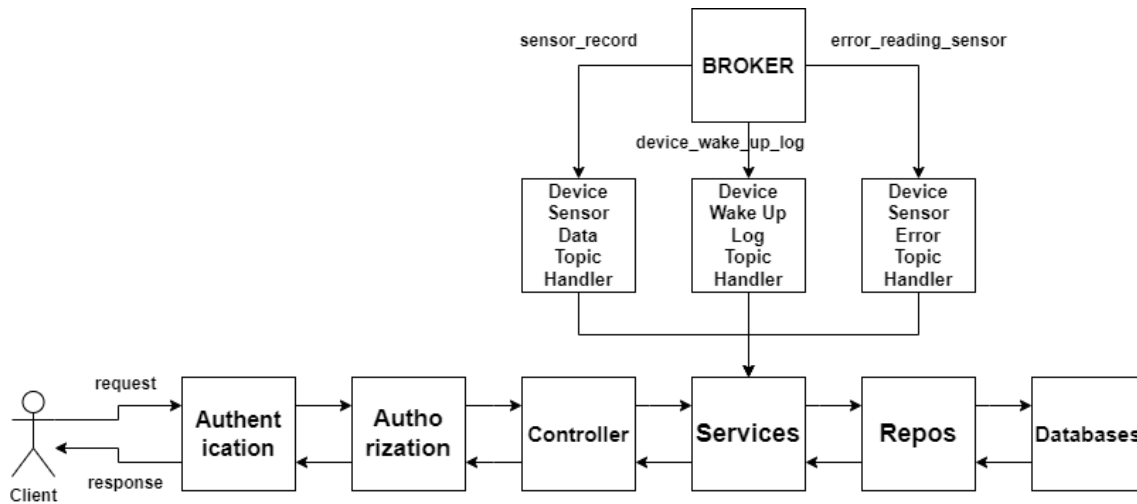


Figure 3.13: Spring server overview

3.3.2 Application Server

The server computes all system involved data. All the business logic is enforced here. This application serves two purposes:

1. Exposes a Web API, allowing client applications to interact with the system;
2. Store and compute all system data, enforcing business logic;
3. Receive, analyse and store the data that coming from the Broker module.

Figure 3.13 shows the main modules involved:

1. Authentication and authorization module will be used when the request is attempting to access protected resources;
2. The controllers to define the API operations;
3. The services display available system actions which will guarantee atomicity and enforce business logic while accessing and manipulating system data;
4. The repositories implement the database "drivers", meaning they handle all the logic associated with the database access;
5. The MQTT topic handlers, responsible to receive with data coming from the broker, and deal with them. This includes analysing sensor data, storing coming records, etc.

Each time a client sends a request, the controllers will use the services to fulfil the request. Likewise, when the services need to access or store data, they will use the repositories to interact with the physical databases. In essence, the request will follow a left-to-right flow, while the response will flow in the opposite direction.

3.3.3 Application Domain

Users

As with any other system, every user needs to have a unique identifier. We opted to use a UUID (Universally Unique Identifier), a unique identifier with 128-bit, since the probability of replications is very small. We also opted to define two types of users, each one with different capabilities:

- The "admin" role in the system has privileged access to all users and devices, including their sensor data and logs. However, this user is not authorized to create devices. The primary objective of the admin is to monitor other users and their corresponding devices. This role is typically assigned to a member of the company responsible for manufacturing the pH neutralization mechanism. Their purpose is to gather data from all devices to improve the product and provide valuable insights. This user cannot be created using the Server API. Its creation is automated by the backend Server when this is initialized;
- The "client" role, which is able to create devices and is authorized to manage his own devices. This user can be created using the backend server API.

The "admin" role was not a request made by the company, but an implementation approach that we decided to take. In addition to automating the monitoring of the neutralization filter, this project aims to extract relevant data for evaluating the neutralization mechanism's efficiency. The admin role has access to all the data collected from the various clients. This access allows an administrator to analyse and evaluate the performance of the neutralization devices, enabling improvements and optimizations based on the extracted insights, and thus, improving the quality of the company final product.

In addition to this, the administrator can also access all MCU device's stored logs. This way, the admin can act as someone with the task to monitor if the micro-controller devices are behaving accordingly.

Devices

Each device needs its own unique identifier, and a 32-bit ID (2^{32} possibilities) is adequate for our needs. This extensive range of identifiers ensures that each device can be uniquely distinguished within the system, providing a high level of identification accuracy and minimizing the chances of duplicate identifiers.

This system is designed to allow clients to configure their own devices. During the configuration process, the device identifier generated by the web app needs to be manually passed to the device using the Esp Touch Android app. Since the client will have to manually write the device ID using his smartphone, a complex ID wouldn't be suitable. In order to minimize the effort required, we generated a unique identifier consisting of 6 letters, from A to Z, lower and upper case, which is the minimum required length to represent all the 2^{32} possibilities imposed above. This results in a total of 4,294,967,296 possible combinations of different identifiers. Each device is associated with an alert email, defined upon creation. This will be used to notify the user if the filter may need human intervention.

3.3.4 Sensor Analysis

Upon receiving sensor data from the broker, the server is responsible for analysing the data. The analysis involves evaluating the received sensor records to determine if any maintenance is required for the filter mechanism. There are many possibilities

to achieve this. The backend server only needs to know the thresholds associated with each sensor type. For example, the pH lower and upper boundaries might be 6 and 7. One possibility would be for each device to send, along with each value record, the sensor thresholds. The server would receive the value and the respective thresholds and process it immediately. The main disadvantage is that it requires more bandwidth, which deeply impacts the power consumption on the MCU. The advantage is that it would be possible to customize threshold values among different devices. But our architecture assumes that all filters belong to the same company, and thus, they should function under the same conditions. Therefore, the current solution is to store the sensor thresholds inside a configuration file, in the backend server. This implies that all devices' sensor data will be treated similarly.

3.3.5 Authentication and Authorization

Most Spring Server API operations request credentials to access resources. Therefore, the backend API allows the client to register itself. To identify logged-in clients, a cookie must be present in the http request header, containing the session token.

There are two options to register a new user:

- Google authentication is implemented in the server using the OAuth2.0 protocol, allowing third-party client applications to authenticate with the Server API. When a user chooses to authenticate with their Google account, the backend server verifies the user's identity and retrieves their Google profile information, including their email address. If it is the user's first time using Google authentication, the backend server creates a new user account based on the Google profile, using the user's email as the unique identifier;
- Standard user creation. Requires an email address and user password.

To ensure control over incoming requests to the server, a Spring interceptor is utilized to handle authentication and authorization checks. The interceptor intercepts incoming requests and performs authentication and authorization validation against the request. The interceptor analyses the request containing the cookie with the access token and comparing against the database. When not valid, the interceptor determines that the request should be terminated because the user is classified unauthorized, responding with a *401 Unauthorized status code*. This means that the user's request is immediately halted, and a response indicating the lack of authentication or authorization is sent back to the client.

The 401 Unauthorized status code is a standard HTTP response status code used to indicate that the client request has not been completed because it lacks valid authentication credentials for the requested resource[16]. By terminating the request and returning a 401 Unauthorized response, the client is notified that they need to provide valid authentication credentials to access the requested resource. This approach ensures that unauthorized access attempts are promptly rejected, and the client is informed of the authentication requirement. It helps enforce security measures and protects sensitive resources from unauthorized access attempts.

To enforce authorization, the interceptor tries to authenticate the user and evaluate if the user possesses the necessary role defined in the annotation to be allowed to proceed with the request. If not allowed, a 403 Forbidden status code is returned. The 403 Forbidden status code is an HTTP response status code that signifies that the server understood the request but refuses to authorize it. It is used to indicate that the server has explicitly forbidden the requested action, even if the user is authenticated. An example of when this could happen is when someone with the role "client" tries to access resources only available to members with the role "admin". An example scenario where the Spring interceptor would come into play is when a user with the role of "client" attempts to access resources that are restricted to users with the role of "admin". The interceptor intercepts the request and checks the user's role. If the user is identified as a "client" and tries to access an "admin" resource, the interceptor would deny the request and return an appropriate response, indicating that the user does not have the necessary privileges to access the requested resource.

3.3.6 Database

Due to the nature of this project, there are two types of data to be persisted:

- Aggregate data: Users, devices, etc;
- Time based data: Sensor data.

A relational database (RDB) was chosen for storing data that involves relationships, such as users having devices and devices having logs. A relational database is specifically designed to enforce structure and maintain relationships between data entities. Additionally, when dealing with sensitive data like user information, it is crucial to handle it with care.

To construct each database, our approach began with the design of their data models. These models serve as a structure with the purpose of organizing data elements and establishing standardized relationships among them. By doing this, we are selecting the requirements for our specific domain. The Figure 3.14 represents the schema of a relational database designed to manage user, device, log, sensor error, token, and password information. The schema defines the relationships between these entities, ensuring the proper association and integrity of data within the system.

Each user can have multiple devices in operation. It has been determined that each device must be exclusively associated with the email address of the supervisor. This email address is provided during the initial registration of the device and serves as the designated recipient for all alerts related to that particular device. Each device can have a set of logs and sensor errors. This can only exist when associated to the device. The same goes for the tokens and passwords, as they can only exist tied to a user.

To summarize the relationships:

1. User and Device: One-to-Many (One user can have multiple devices);

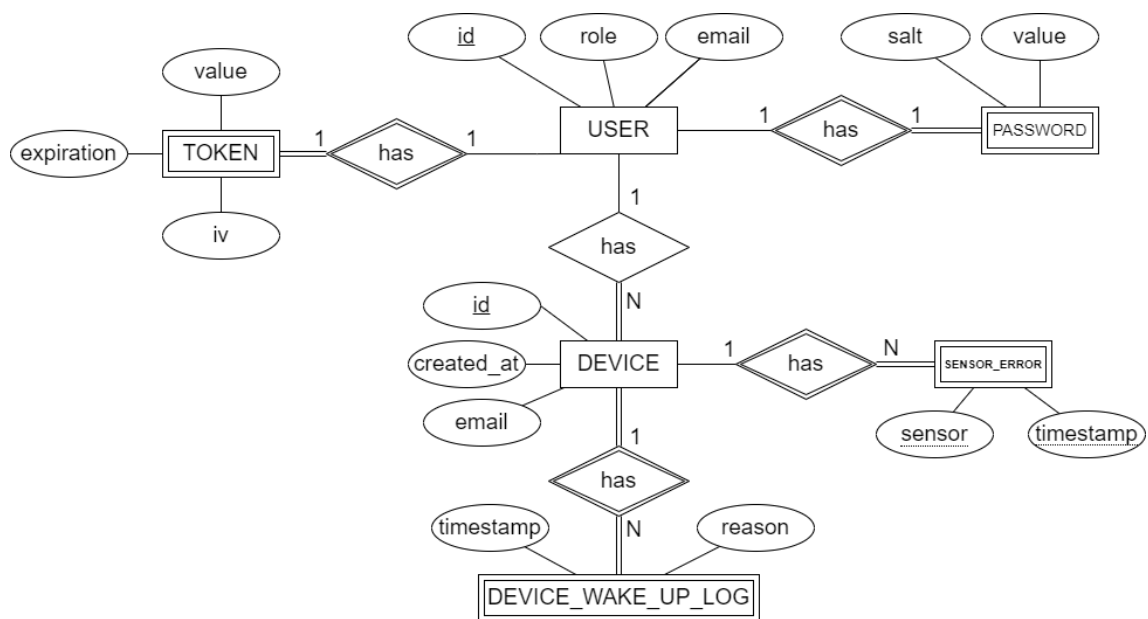


Figure 3.14: Relational DB Entity Model

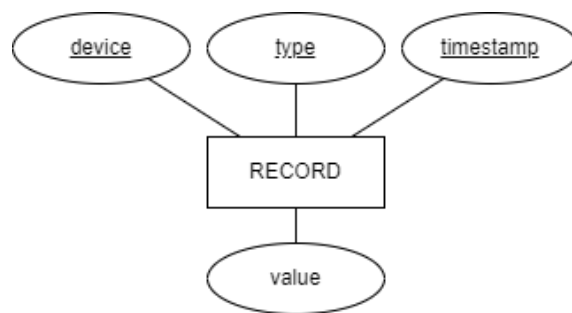


Figure 3.15: Time Series DB Sensor Record representation

2. Device and Logs/Sensor Errors: One-to-Many (One device can have multiple logs and sensor errors).
3. User and Tokens/Passwords: One-to-One (One user has one set of tokens and one password).

These relationships ensure the proper association and integrity of data within the system, allowing for effective management and organization of users, devices, logs, sensor errors information, tokens, and password information. Since InfluxDB is a non-relational database, the standard relational data representation approach does not apply. However, the diagram 3.15 offers a clear understanding of the type of data it stores. A sensor record always consists of the device, the type of the sensor, the timestamp, and the corresponding reading.

3.4 Website

The Web application appears as a user-friendly entry point to the system. It interacts with the exposed Web API, in the backend, to access and manipulate data.

The primary goal of the website is to offer an interactive user interface that enables buyers of the neutralization mechanism to visually interpret the collected data.

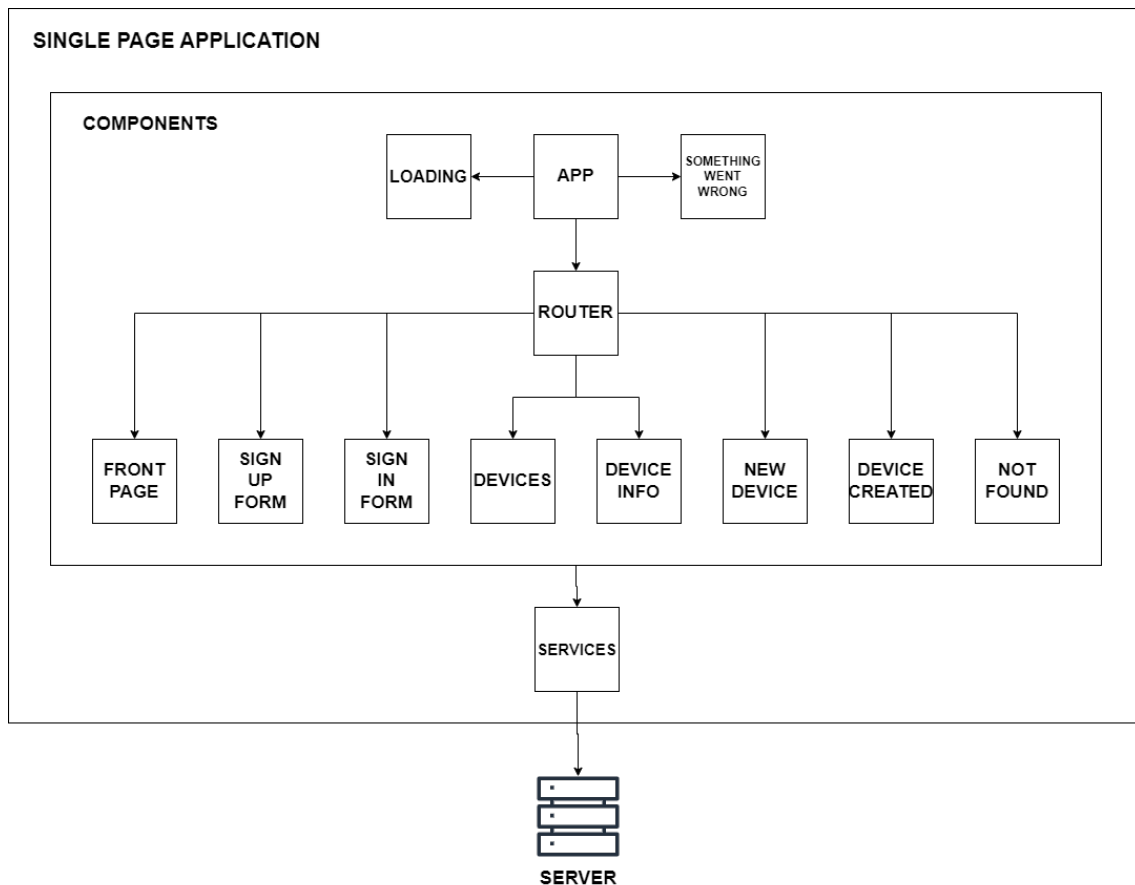


Figure 3.16: Single Page Application components diagram

Through the website, users can access and explore the data in a graphical format, enhancing their understanding and facilitating analysis of the information gathered.

3.4.1 Data Visualization and Device Logs

The main purpose of the website is to visualize the data, collected from the sensors, providing simple analytics of the neutralization mechanism under monitoring. Therefore, the website displays a graph with collected sensor data, for each registered device.

The graph allows choosing one of the existing sensors, if any, for like the pH, the temperature, humidity, etc. He can also select a time interval to sort data, ranging from hours to years. This way, the user can observe how the filtration system is progressing with time. He can also detect noise readings, for example when the graph is showing spikes. Apart from the graph, the user can also download the data, in CSV format, which may be handy when the intent is to process data, using external tools. Additionally, the user can consult the device logs, which allows him to know the state of the MCU, for instance, to know if an error occurred.

3.4.2 Website Architecture

The website architecture is represented in Figure 3.16, it presents an overview of the different modules and connections that constitute the website.

The App module is responsible for loading the router, which is responsible for selecting the component that the user will see. The majority of components within the application rely on the services layer to access the backend server. This allows them to request and manipulate system data. The services layer acts as an intermediary between the components and the backend server, handling the communication and data operations on behalf of the components.

3.4.3 Hypermedia

Using hypermedia-driven APIs, such as the Siren API, offers several advantages in the design and development of web APIs. The key advantage lies in the self-descriptive nature of hypermedia representations, which enables clients to dynamically navigate and interact with the API without prior knowledge of its structure.

In a Siren-based API, resources are represented as JSON objects with specific properties such as "class," "properties," "links," and "actions." The "links" property provides links to related resources, allowing clients to navigate between them. The "actions" property describes the available actions or operations that can be performed on a resource, along with their associated input parameters.

In the frontend implementation, the utilization of hypermedia is evident. Upon loading the "React App" component, the initial step is to retrieve all the necessary Siren API information from the backend by making a request to the known endpoint dedicated to fetching the complete Siren representation. This endpoint allows the website to dynamically explore and navigate through all the available API endpoints, eliminating the need for prior knowledge of the API's structure.

In cases where the necessary Siren API information cannot be obtained, the website gracefully handles this situation by displaying a static error message. This behaviour is designed to ensure a good user experience, as failing to retrieve the expected siren information would hinder the proper functioning of the Single Page Application (SPA). If the Siren information is successfully obtained, the Single Page Application (SPA) proceeds to set up the React Router using the retrieved data. The React Router is responsible for managing the navigation and routing within the SPA, allowing users to navigate between different pages and components

3.4.4 Protected Resources

The majority of the website resources are protected, meaning the standard users should not be able to access them, but only authorized ones. Therefore, the website provides a sign-up/sign-in page, for authentication purposes. Once a user has successfully logged in, they will be redirected to an authorized page within the application. However, there is no restriction preventing users from attempting to directly access a protected resource by manually entering the page URL in the browser's search bar, also known as deep linking. For that reason, there is a special component - "RequireAuthn" - that wraps the protected component, ensuring that the user is logged-in. His behaviour is demonstrated in the ASM chart represented in Figure

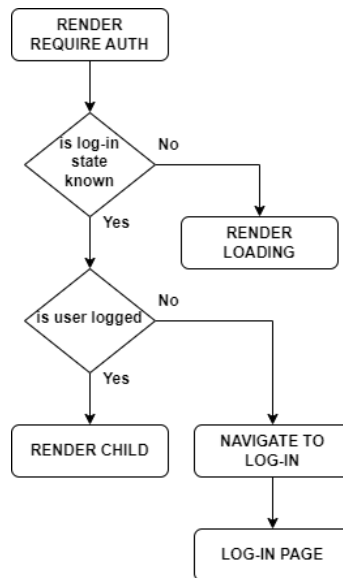


Figure 3.17: Authentication control in the SPA

3.17. By using this component, it ensures protected components are only accessed by authenticated users.

The login state is managed by the "AuthnContainer" component, and accessible to most SPA React components, so they can check if the user is logged in, and use that information, for example to make an API request.

Readers may raise the question: "Once the user has logged in and the authentication state is stored in the "AuthnContainer" component, what happens if the user refreshes the page and the authentication state is lost?" In such a scenario, it is natural to wonder whether the user would be required to log in again.

In our system, we store a token inside a cookie to persist the login state. This is used, not for the browser to know that it is logged in, but for the server to know the user is logged-in. The source of truth lies in the token validity. Since the cookie, sent by the server, cannot be accessed in JavaScript, to protect from cross-site scripting (XSS) attacks as explained in the section 3.6, the browser needs to make an API request to know if the client is authenticated or not. This happens when the user uses deep linking to access a protected resource, and the authentication state is not yet known by the browser. However, the token stored in the browser might still be valid, and he should not have to go through all authentication steps again. Similarly, to make the logout, the browser makes an API request, so the server can remove the cookie, therefore eliminating the client session.

3.4.5 Error Handling

Most unpredictable events will come from the backend server (ex: when the user requests his own devices, sensor data, etc). In these situations, after making a request, the response could be a 400s or a 500s. The response will, most times, be a *problem+json*, and have a detailed description about what caused the error outcome. Despite this, it is not always practical for an SPA to know all possible errors, and know how to deal with them in particular. To handle potential errors in the application,

various scenarios were considered. While certain errors, such as invalid credentials during the login/sign-in process, are expected and can be handled appropriately by the respective React component, other types of errors, like a 500 internal server error, may occur during different types of requests, leaving the website unsure how to handle them. To address this, a dedicated React component called the "Error-Container" was implemented. This component wraps around another component, which represents the desired user interface in case of successful operations. The "ErrorContainer" is designed to re-render itself whenever an error occurs, enabling control over the user experience, particularly in situations where unexpected errors arise, ensuring that errors are not ignored or overlooked.

3.5 System Tests

3.5.1 Backend

The central component in the backend is the Spring Server, and so, extensive tests were implemented to access its well execution. In our development process, we have placed a strong emphasis on automated testing, particularly for the server-side layers of our application. This includes testing the services, repositories, domain, and controllers through integration testing. There is also a python script that emulates the hardware device to send synthetic sensor data to the backend broker, so it is possible to test the server sensor data analysis without relying on the real hardware.

3.5.2 MCU

To test the MCU firmware, we opted for simple programs that need to be loaded in the hardware. These are not automatic tests, unlike in the Spring Server. The objective is to test each module independently without having to rely on the main ESP program. For instance, to test the pH reader module, a simple program that only uses that module, is loaded into the ESP hardware and the developer uses the console logs to make sure the module is behaving as expected. This simple programs are also used to collect sensor data, in order to perform further analysis, for instance to plot the data in a graphical form. The same goes for the other sensor modules.

3.5.3 Power Consumption

As emphasized throughout this report, power consumption plays a crucial role in the design of this system. This holds true not only for IoT devices in general, but particularly for this specific project, as per the requirements outlined by the company.

During the development phase, utmost importance was placed on implementing various power-saving techniques. However, the effectiveness of these techniques could not be determined without conducting proper experimental evaluations. To address this, a device called the UM24C USB tester was utilized to measure the power consumption of the ESP32-S2 during different execution stages. This equipment

Current measurement in different states	
	Consumption
Active mode idle	27mA
Power on DHT11	29mA
DHT11 readings (spaced by 1 second)	29mA
Power on water sensor	27mA
Read from water sensor (no water detected)	29mA
Read from water sensor (water detected)	38mA
Read from pH sensor (not altered)	34.4mA
Read from pH sensor (altered)	31mA
Deep Sleep	10mA
Wi-Fi and MQTT setup	61mA
Wi-Fi reconnect timeout (max tries)	78mA
Active mode Wi-Fi idle	31mA
Active mode Wi-Fi sending MQTT	39mA

Table 3.2: Results for the different MCU consumption stages

boasts a current measurement resolution of 0.001A, which provided sufficient precision for conducting meaningful experiments.

To evaluate the power consumption of our system, a dedicated program was developed to simulate various operational modes of the MCU. Each mode was tested individually to assess its power requirements. Upon starting the program, the specific running mode was initiated and monitored for a duration of 30 seconds. This approach allowed us to accurately measure and analyse the current consumption of the system under different operating conditions. The collected values, as presented in the table 3.2, represent the instantaneous current readings obtained during the evaluations.

The tests yield readings way above the ones indicated in the respective data sheet. This difference can be explained by the presence of power LED's in the ESP32-S2 module and associated sensors, which cannot be turned off. During development, it is convenient to have them turned on to indicate when the sensor is being used, for debug purposes. However, from the experimentations made, it is noticeable clear the existent differences between distinct MCU profiles.

As explained in the subsection 3.2.1, the MCU will wake up and do one of two reasons:

- To read from all sensors. This iteration will always use Wi-Fi to make data transmissions;
- To only check for water leaks. This iteration will only make transmissions if water leakage is detected.

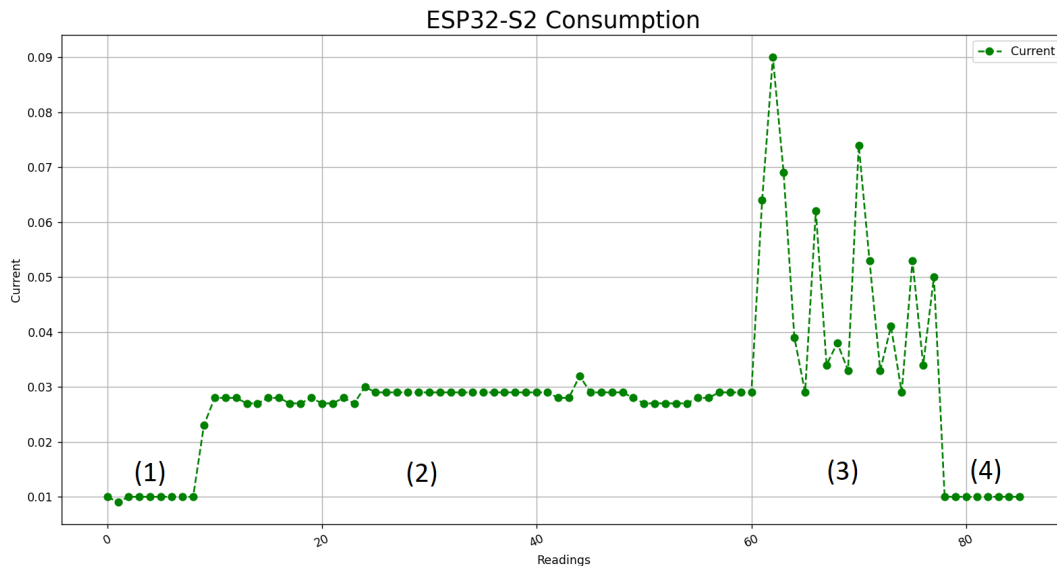


Figure 3.18: Graph illustrating the ESP32-S2 energy consumption measured when the MCU wakes up to read from all sensors

The image 3.18 shows the instantaneous current consumed at different stages of consumption. In the initial and final stages, (1) and (4), the MCU is in a sleep mode, resulting in relatively low power consumption. Upon waking up, stage (2), the MCU engages the sensors to perform readings, leading to a moderate level of power consumption. However, when the Wi-Fi functionality is activated for data transmission, stage (3), the power consumption significantly increases, indicating that this stage predominantly influences battery usage.

Power Range

To estimate how much the device battery is going to last, a test was conducted, in which the MCU stayed in a constant loop for three hours, doing its normal active mode execution. In one iteration, the MCU only reads from the water sensor. In the next one, it reads from all sensors and performs data transmission. The execution is explained in the 3.2.1 section.

To accurately estimate the total power consumption, it is necessary to consider the energy usage during a single cycle.

A cycle consists of an MCU in active mode plus 12 hours in deep sleep: (1) + (2) + (3) + (4) (Represented in Figure 3.19).

Some stages that only happens once (ex: sensor calibration and time synchronizing) were left off from this test. The accumulated consumption was recorded, along with the total number of cycles. Next, to estimate the total energy consumption during deep sleep, a similar test was performed. The ESP stayed in the deep sleep mode for three straight hours, and the accumulated consumption was recorded.

Based on the project requirements, the ESP32-S2 is programmed to wake up at a 6-hour interval. The active mode is when the MCU is performing readings or making data transitions. With a 3550mAh battery capacity, it is crucial to ensure that the total

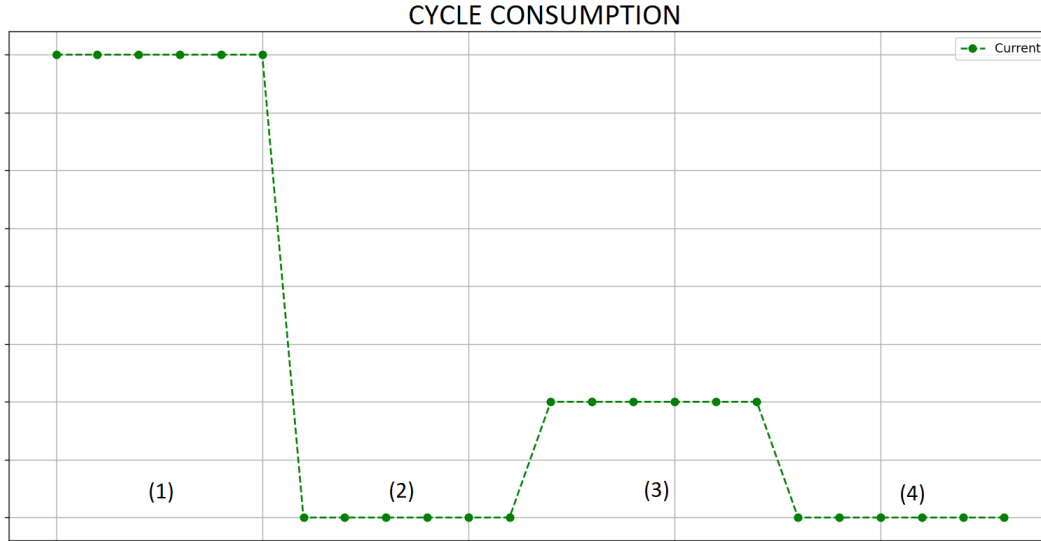


Figure 3.19: Image showing the 4 stages in a reading cycle

consumption does not exceed this limit.

$$EnergyPerCycle * NumberOfCycles = Total \quad (3.2)$$

$$EnergyPerCycle = (1) + (2) + (3) + (4) \quad (3.3)$$

$$Total = 3350 \quad (3.4)$$

The results showed a total consumption of 80mAh for three hours in active mode, (((1) + (3)) * 75). The deep sleep mode consumed a total of 30mAh during the three hours in deep sleep mode. This means that "EnergyPer12HoursDeepSleep" is equal to 1200 mAh ((2) + (4)). The consumption calculation is as follows:

$$(1) + (3) = 1.1(mAh) \quad (3.5)$$

$$(2) + (4) = 120(mAh) \quad (3.6)$$

$$NumberOfCycles = +/- 28 \quad (3.7)$$

With 28 cycles, it corresponds to 14 days of operation. This calculation enables us to ensure that the battery capacity is sufficient to support the desired operation of the system within the specified timeframe.

According to the datasheet, the EPS32-S2 in deep sleep mode drains an instantaneous current of 0.025mA. If we replace this value in the variables (2) and (4), the "NumberOfCycles" would be equal to +/-2978, which, dividing by 2 is equal to +/-1489 (days). This is roughly equivalent to 50 months.

3.6 Security Measures

Ensuring the security of the system is of paramount importance in any application, especially when dealing with sensitive data and user authentication. In this

section, we will discuss the security measures implemented to protect the system from unauthorized access and potential vulnerabilities.

3.6.1 Password Storage

. In our system, we prioritize the security of user passwords by storing them as hashes instead of clear text. A hash is a computed value derived from the original password, which adds another layer of protection. By securely storing passwords as hashes, we can maintain a high level of confidence in protecting user data within our system. In the event of unauthorized access to our system, the use of password hashing ensures that passwords remain secure. Additionally, hashing passwords enhances user privacy, as none of the database administrators or personnel can view the passwords in clear text form. To further enhance password security, we have implemented a technique called "salting" in addition to password hashing. When computing the hash value of a password, a randomly generated value (known as a salt) is added to the password before hashing. This salt provides an extra layer of randomness to the resulting hash. As a result, even if two users have the same password, their hashed values stored in the database will be different due to the unique salts applied. This approach significantly strengthens the security of passwords and mitigates the risk of password-based attacks, such as *rainbow table attacks* or *precomputed hash attacks*. In a rainbow table attack, a large precomputed table of hash values and their corresponding plaintext passwords is used. The attacker compares the hashed password from the target system's database with the entries in the rainbow table to find a match. Precomputed hash attacks, on the other hand, involve the creation of a precomputed table of hash values for a large set of possible passwords. This table is generated in advance, allowing the attacker to quickly look up the hash value in the table to find the corresponding password. By salting the passwords, we introduce a random and unique value for each password before hashing. This prevents the effectiveness of both rainbow table attacks and precomputed hash attacks. Even if an attacker has access to the hashed passwords in the database, they would need to generate separate rainbow tables or precomputed hash tables for each unique salt, making the attack significantly more difficult and time-consuming. We can see a visual representation of how salts act in the figure 3.20

3.6.2 Website

When a user attempts to create an account on our website, they are presented with two options: signing up through the Google authentication option or using the built-in registration mechanism provided by our web application. If the user chooses to sign in via Google authentication, a separate page will be displayed, requesting their permission to access their data. If the user grants permission, our application will retrieve their Google email and store it in our system, creating a new account

²Image taken on July 6, 2023, from <https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/>





				
Password	p4s5w3rdz	p4s5w3rdz	p4s5w3rdz	p4s5w3rdz
Salt	-	-	et52ed	ye5sf8
Hash	f4c31aa	f4c31aa	1vn49sa	z32i6t0

Figure 3.20: Visual representation of the function of the salt ²

using that email. If the user chooses to create an account through the website mechanism, they will be prompted to provide an email and password. To ensure password security, the password must meet certain requirements to be considered valid, such as having a minimum length of 8 characters, containing at least one digit, and at least one uppercase letter. This approach is implemented to discourage the use of weak passwords, as research has shown that common or easily guessable passwords are more susceptible to being compromised[17]. To prevent the creation of fake or unauthorized accounts and to add an extra layer of identification verification it is required that the user must confirm their email address through a verification code sent to the email supplied by the user, this process helps ensure that the email address provided during registration or account creation is valid and belongs to the user. This confirmation process also provides a barrier to scripts or bots with the objective of creating multiple accounts on the system, since the confirmation code is sent to the user's email address, automated processes would struggle to complete the verification step.

To maintain user authentication in our application, a token is utilized, as explained in the previous subsection on Protected Resources. This token is securely stored in a cookie and serves as a unique identifier for the user. However, it is crucial to protect this token from potential attackers who may attempt to hijack it through Cross-Site Scripting (XSS) attacks. XSS attacks involve injecting malicious scripts into web pages to gain unauthorized access.

To mitigate this risk, we employ the HTTP-only attribute to safeguard our cookies. By setting the HTTP-only attribute, we ensure that JavaScript running on the page cannot access or modify the value of the HTTP-only cookie. This fortifies the security of the token stored within the cookie, making it significantly more challenging for attackers to exploit XSS vulnerabilities and impersonate users.

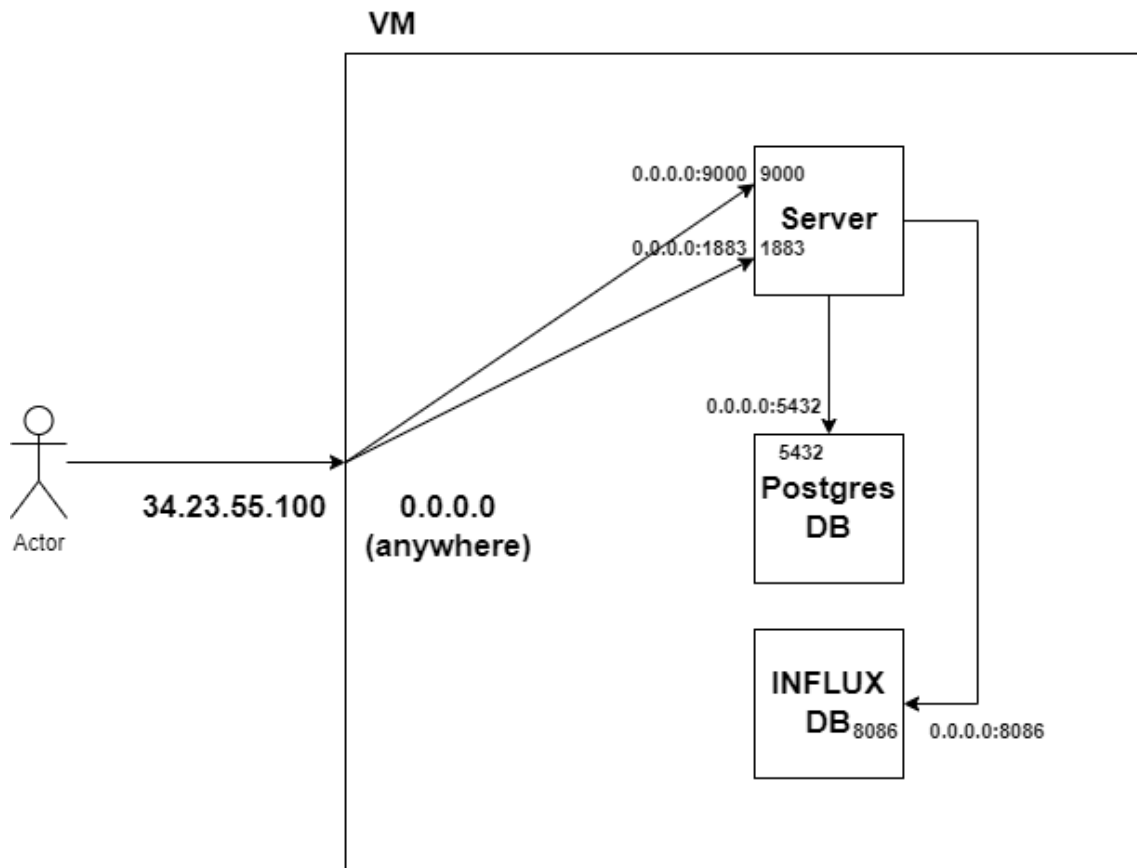


Figure 3.21: Containers for each system service layer

3.7 Deploy

This system is composed of three main components, described in the section 3.1. The backend and frontend components should be accessible to all the system users. To provide isolation between the different system layers, we opted to use docker engine and build containers for each service:

- The Postgres and Influx DB are two individual containers, built from an available open source image, supporting data storage;
- The Spring Server plays a dual role in the system, serving as both the receiver of backend requests and the provider of the static website and JavaScript file. In order to streamline the deployment process, Webpack was utilized to bundle multiple JavaScript files into a single file, which is then served by the Spring Server. Additionally, the MQTT broker is currently deployed from the same Spring Server, consolidating the various components of the system within a single deployment environment.

Given the lack of information regarding the scale requirements of the system, a decision was made to implement a simplified solution utilizing a single Virtual Machine (VM) to host the entire cloud system, including the necessary Docker containers. This approach offers cost benefits as deployment expenses are minimized with a single VM. However, it also presents drawbacks such as a single point of failure,

meaning that if the VM experiences an issue, the entire system could be affected. Additionally, scalability options are limited with this setup, potentially hindering future expansion efforts. Due to the absence of free deployment services, a pay-to-use service was selected for hosting the system. The choice was made to utilize the GCP (Google Cloud Platform) Compute Engine service, primarily due to the availability of a \$50 coupon that allowed for the allocation of a virtual machine (VM) without incurring additional costs. This decision enabled the deployment of the system on GCP at no immediate expense, utilizing the allocated resources efficiently.

3.8 Summary

In this chapter, we provided a comprehensive overview of the architecture of the proposed system, delving into the intricate details of its key components and the underlying technologies. We presented a detailed analysis of the prototype, highlighting the extensive testing and evaluation conducted to ensure its robustness and effectiveness. The architecture of both the backend and frontend was meticulously outlined, elucidating the intricate mechanisms and implementation decisions driving their functionalities. We successfully achieved this by designing an ultra-low power consumption device capable of accurately detecting and alerting any environmental anomalies within an industrial mechanism. This detailed understanding of the architecture will facilitate a deeper exploration of the implementation, performance evaluation, and future enhancements of our innovative system.

4

Conclusion

During the development of the project we had the opportunity to exercise the technologies we learned throughout the course such as React and Spring Framework and that are widely used in the industry.

We also had to learn to use other technologies and protocols such as time series database and MQTT protocol and beyond that we had contact with hardware, more specifically the programming of the ESP32-S2 microcontroller, which was one of the main challenges during the project since both elements of the group doesn't have any experience in this area.

Despite these challenges, the group managed to successfully meet the requirements that were proposed for the project. Overall, all the purposed requirements were fulfilled in time. The prototype was subject to tests made to ensure the well function of the system, not just at the software level, but also at the hardware level. The documentation was produced accordingly. Therefore, the project is considered finished, for a first production version, but there is still space for improvements and new features.

4.1 Future Work

In this section, we discuss potential areas of improvement and future work for the system.

4.1.1 Hardware Adaptation

The current implementation utilizes the development kit of the esp32-s2, which includes LEDs that were useful during development but are not necessary for the final hardware system. As part of future work, the hardware design should be adapted to remove these LEDs and any other components that are not essential for the system's functionality. This will help optimize the system's form factor and reduce any interference caused by unnecessary components.

4.1.2 Dynamic Configuration

Currently, the URL of the broker, which serves as the connection point between the MCU and the server, is statically referenced in the code. To enhance flexibility and configurability, a future improvement would be to implement dynamic configuration for the broker URL.

4.1.3 pH Transformation with Temperature Compensation

The transformation of pH values is influenced by the temperature of the liquid being measured. In the current implementation, the system does not extrapolate or adjust the pH readings based on the temperature of the water. As a future enhancement, incorporating temperature compensation algorithms can provide more accurate and reliable pH measurements by compensating for the temperature effect. This would improve the overall precision and reliability of the system's pH measurements.

4.1.4 Hermetic Enclosure

To ensure the longevity and reliability of the hardware components, it is recommended to protect the device in a hermetic enclosure. This will safeguard the system from environmental factors such as dust, moisture, and physical damage. By creating a sealed enclosure, the hardware components will be shielded from external elements, ensuring the system's durability and longevity in various operating conditions.

4.1.5 Scalability

To deploy the MQTT broker, it is recommended to use a dedicated container to enable independent scaling of the broker from the server. However, at the time of writing, this has not been implemented. The broker is currently being served from the Spring Server, making it dependent on the server's availability. During development, integrating the broker directly with the server was more convenient, as it eliminated the need to launch it as a separate service.

4.1.6 Water Flow sensor

Although the project requirements did not initially necessitate the inclusion of a water flow sensor, we decided to integrate it to further analyse the behaviour of the neutralization system. By collecting data on water flow, we can assess whether it aligns with recommended values and gain a deeper understanding of system performance.

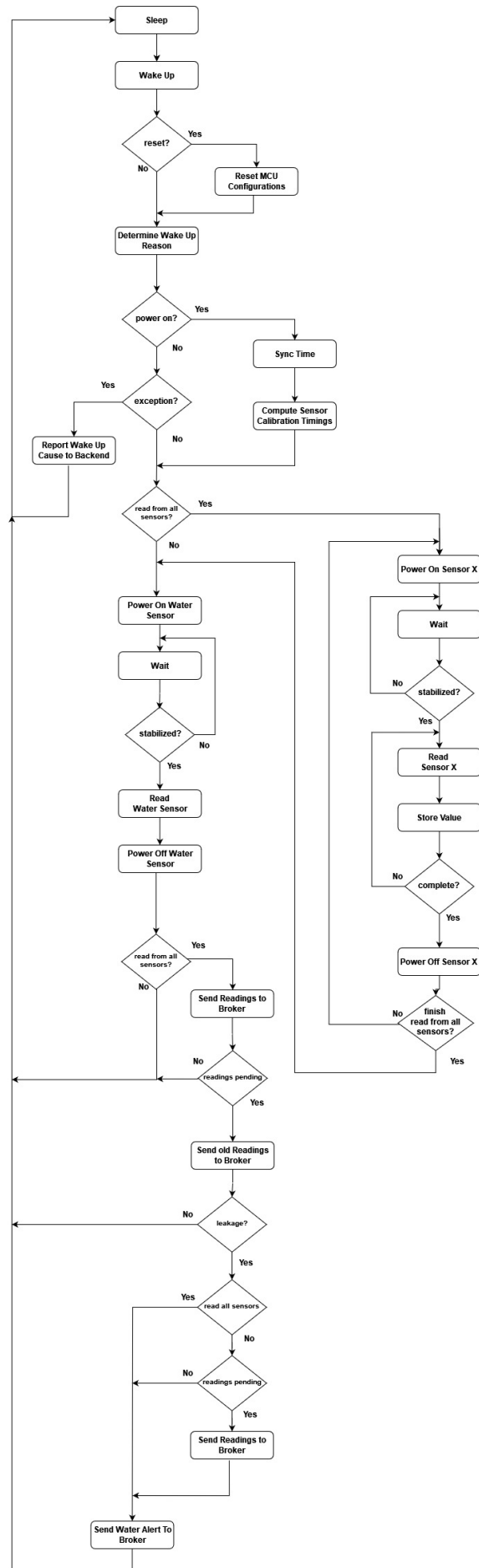
Although the system has the capability to incorporate a water flow sensor, currently the MCU is only sending synthetic water flow data. The remaining task involves implementing the necessary code to enable the MCU to interface with the physical sensor.

4.1.7 System future support

When this system goes to production, a virtual machine will have to be running in order to support the backend and frontend layers. Without this, the data will not be collected, analysed and stored. The MCUs don't need more development support, since they only need power to work properly. The firmware should not need any update, unless a bug is detected, or additional features are required.

In order to deal with possible existent glitches, the MCU should have a mechanism for checking updates, automatically. If detected, the MCU would update itself.

Appendix A - MCU Behaviour ASM Chart



Appendix B - API documentation

Bibliography

- [1] K. Addy, L. Green, and E. Herron, "ph and alkalinity," *University of Rhode Island, Kingston*, 2004.
- [2] K. Rose, S. Eldridge, and L. Chapin, "The internet of things: An overview," *The internet society (ISOC)*, vol. 80, pp. 1–50, 2015.
- [3] W. Wolf, "What is embedded computing?" *Computer*, vol. 35, no. 1, pp. 136–137, 2002.
- [4] "Introduction to spring framework," Available online, Spring Framework Reference Documentation, accessed: 10.03.2023. [Online]. Available: <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html>
- [5] S. Bose, M. Mukherjee, A. Kundu, and M. Banerjee, "A comparative study: java vs kotlin programming in android application development," *International Journal of Advanced Research in Computer Science*, vol. 9, no. 3, pp. 41–45, 2018.
- [6] "Relational databases vs time series databases," Available online, Influxdata, accessed: 11.03.2023. [Online]. Available: <https://www.influxdata.com/blog/relational-databases-vs-time-series-databases/>
- [7] "Db-engines ranking of time series dbms," Available online, DB-Engines, accessed: 11.04.2023. [Online]. Available: <https://db-engines.com/en/ranking/time+series+dbms>
- [8] M. A. Khan, M. A. Khan, S. U. Jan, J. Ahmad, S. S. Jamal, A. A. Shah, N. Pitropakis, and W. J. Buchanan, "A deep learning-based intrusion detection system for mqtt enabled iot," *Sensors*, vol. 21, no. 21, p. 7016, 2021.
- [9] E. Al-Masri, K. R. Kalyanam, J. Batts, J. Kim, S. Singh, T. Vo, and C. Yan, "Investigating messaging protocols for the internet of things (iot)," *IEEE Access*, vol. 8, pp. 94 880–94 911, 2020.
- [10] "Webpack documentation," Available online, Webpack, accessed: 12.03.2023. [Online]. Available: <https://webpack.js.org/concepts/>
- [11] A. D. Gupta, M. M. Islam, M. R. Islam, Z. Sadek, T. R. Toha, A. Mondol, and S. M. M. Alam, "Devising an iot-based water quality monitoring and ph controlling system for textile etp," in *2023 International Conference on Electrical, Computer and Communication Engineering (ECCE)*. IEEE, 2023, pp. 1–6.

- [12] A. Riansyah, R. Mardiaty, M. R. Effendi, and N. Ismail, "Fish feeding automation and aquaponics monitoring system base on iot," in *2020 6th international conference on wireless and telematics (ICWT)*. IEEE, 2020, pp. 1–4.
- [13] "Sleep modes," Available online, Espressif, accessed: 20.03.2023. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html
- [14] "Analog to digital converter," Available online, Expressiff, accessed: 24.05.2023. [Online]. Available: <https://docs.espressif.com/projects/esp-idf/en/v4.4.1/esp32s2/api-reference/peripherals/adc.html>
- [15] "System time," Available online, Expressiff, accessed: 02.07.2023. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/system_time.html
- [16] "401 unauthorized," Available online, Mozilla, accessed: 29.06.2023. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/401>
- [17] J. Renders, "Why implementing a strong password policy is important for your business's data," Available online, brightlineIT, accessed: 20.06.2023. [Online]. Available: <https://brightlineit.com/why-implementing-a-strong-password-policy-is-important-for-your-businesss-data/>