Bachelor's Thesis

Part of Knightec Group

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Kongsberg

Course: TS3000 Bacheloroppgave
Date: May 20, 2025
Title: Kaffeknekt

Project group: 3
Group members:

Sokaina Cherkane
Martin Taraldstad
Didrik Aas Bergan
Mikolaj Marek Szczeblewski
Ivan Bergmann Maronsson
Abdulkadir Kabuk

Internal supervisor: Joakim Bjørk, Zoran Dokic
External supervisor: Sofie Aarnes

Project partner: Semcon Part of Knightec Group

Acknowledgments

We would like to begin by expressing our deepest gratitude to our supervisor, Joakim Bjørk, whose support in navigating the project management process has been invaluable. Your guidance helped us in turning a complex vision into a structured and achievable goal. You made sure we stayed on track, reflected on our directions and decisions, and moved forward with clarity and confidence and for that, we are truly thankful.

To Zoran Dokic, our secondary supervisor, we would also like to extend our sincere thanks for their valuable technical guidance and input, particularly in support of the electrical engineering aspects of the project. You helped bring our ideas to life, and gave us the confidence to tackle real world challenges head-on.

All of us are deeply grateful to Semcon Part of Knightec Group, our project partner, for not only providing resources, but for your engagement, insight and support in areas beyond our own expertise. On several occasions, when we encountered technical or logistical challenges that put our project at risk, your guidance helped us move forward, your willingness to engage with our project contributed in a meaningful difference, and we sincerely appreciate the trust you've put into us and the collaboration extended to us.

To the faculty and staff at the University of South-Eastern Norway, we are grateful for providing us with the academic environment and resources necessary for the completion of this work.

We are grateful to all of our fellow group members for the shared learning and interdisciplinary cooperation that made this project rewarding and educational.

This bachelor's thesis represents not just a technical result, but the culmination of months of dedication, discovery, problem-solving, cooperation, persistence, and growth. It will remain a defining milestone in our academic and personal lives.

Abstract

The lack of real-time feedback in commercial espresso machines creates challenges in achieving consistent quality. Key variables such as temperature and pressure, which significantly influence extraction, often remain hidden from the user. Without access to this data, it becomes difficult to explain variations in results or adjust the brewing process with precision.
To address this issue, this project introduces a smart monitoring system capable of capturing and visualizing brewing conditions during espresso extraction. The system enhances process transparency and empowers users with insights that traditional machines do not provide, bridging the gap between manual operation and data-supported brewing.

Contents

List of Figures

List of Tables

Abbreviations

AC Alternating Current. 157, 177, 179

ADC Analog-To-Digital Converter. 8, 11, 48, 50, 74, 159, 161, 162, 164, 167, 172, 173, 255

CPAF Cross-Platform Application Framework. 36

CSS Cascading Style Sheets. 39

DC Direct Current. 50

ECAD Electronic Computer-Aided Design. 35

Faema Fabbrica Apparecchiature Elettromeccaniche e Affini. 24, 50, 179

GCC GNU Compiler Collection. 76

GPIO General-Purpose Input Output. 44, 73, 159

GUI Graphical User Interface. 19, 37, 53, 55, 65, 77, 268

HDMI High-Definition Multimedia Interface. 44, 46

HTML HyperText Markup Language. 39

I/O Input/Output. 44

I2C Inter-Integrated Circuit. 10, 12, 45, 47, 50, 53, 62, 72, 73, 159–161, 164–166, 168, 254, 276, 278

IC Integrated circuit. 164, 166, 170

IDE Integrated Development Environment. 36, 37

IoT Internet of Things. 19, 45, 57

JSON JavaScript Object Notation. 64

JST Japan Solderless Terminal. 11, 160, 161, 167, 168, 173

MCU Microcontroller. 19, 45, 50, 52, 71, 159

MISO Master Input Slave Output. 160, 165

MOSI Master Output Slave Input. 160, 165

MVP Minimum viable product. 39, 71, 80

PBI Product Backlog Item. 10, 30, 31

PCB Printed circuit board. 7, 9, 11, 35, 61, 72, 81, 159–167, 169–171, 173, 174, 184

PSU Power Supply Unit. 11, 157, 173, 184

RAM Random Access Memory. 44

RPI-5 Raspberry Pi 5. 37, 44, 46, 50, 54, 71, 73, 75, 76, 157

RTC Real Time Clock. 8, 11, 159, 162, 166, 170, 172

RTOS Real-time operating system. 45

SCL Serial Clock Line. 53, 73, 160, 164–166, 168

SDA Serial Data Line. 53, 73, 160, 164–166, 168, 270

SDIO Secure Digital Input Output. 45

SDK Software Development Kit. 45

SMD Surface-Mounted Device. 11, 159–161, 170, 171, 173

SPI Serial Peripheral Interface. 45, 50, 72, 159, 160, 165

SSD Solid State Drive. 44, 46

SSH Secure Shell. 75

TCP Transmission Control Protocol. 280

UART Universal Asynchronous Receiver / Transmitter. 45

UML Unified Modeling Language. 38

USB Universal Serial Bus. 46, 174

USN University of South-Eastern Norway. 39

VM Virtual Machine. 55, 70, 76, 77

Terminology

7Pace Timetracker Azure DevOps plugin for timetracking.. 10, 31, 32

BibTeX Bibliographic flat-file database file format and a software program for generating lists of references [113].. 34

Bluetooth Short-range wireless technology standard for data exchanging between devices [114].. 45

C++ A powerful general-purpose programming language which utilizes both object-oriented and procedural programming [41].. 36, 44

CERN Short for "Conseil Européen pour la Recherche Nucléaire", is an intergovernmental organization that operates the largest particle physics laboratory in the world [115].. 35

Convection Process involved with carrying heat from one place to another by the bulk movement of a fluid or gas [116].. 26

Grouphead An assembly of static and dynamic components that receives the portafilter and delivers brewing water to the coffee bed at a preset temperature and flow rate [85].. 23–25, 52, 77, 80, 175

Heat exchanger System for transferring heat from one medium to another. 24, 26, 77

InfluxDB Database about InfluxDB. 13, 38, 57–60

Kernel The core part of an operating system. Acting as a bridge between some software application and the hardware of a computer [36].. 45

Linux An open source operating system, with a Linux kernel maintained by a worldwide community [78].. 46

Load cell A device which converts a force such as tension, compression, pressure, or torque into a signal [120].. 81

Percolation The brewing time/process. 20, 21

Portafilter Removable, handheld component that performs the function of retaining the filter basket and securing it into the grouphead during the extraction process [85].. 21, 22, 52

Qt An open-source framework for creating cross-platform applications with a graphical user interface [87].. 36, 37, 44, 75

Strain gauge A device which converts a force such as tension, compression, pressure, or torque into a signal [120].. 81

Thermosiphon A device which utilizes the motive forces of natural convection, creating a cyclic fluid flow from areas of high heat to low heat and back [5].. 25, 26

Transducer A device that converts energy from one form to another [122].. 156

Chapter 1

Introduction

This thesis presents the development of a smart monitoring system for a commercial espresso machine, focused at visualizing key brewing parameters. The motivation behind the project is the growing demand for deeper insight into traditionally manual processes, and how real-time monitoring technology can contribute to that insight. The goal is not to automate the process, but to enable real-time data visualization of the system which provides users and stakeholders with a deeper understanding of the brewing process and machine behavior.

This project was carried out by an interdisciplinary team of engineering students specialized in computer and electrical engineering, in collaboration with Semcon, which are interested in showcasing Industry 4.0 capabilities in new domains. This report introduces the motivation, scope and constraints that shaped the project, in addition to outlining system design, development methodologies and lessons learned throughout the process.

1.1 Project Overview **SC** | *MMS*

Coffee, as one of the most valuable agricultural commodities in the world, is a widely consumed beverage which fuels a vibrant industry, and that industry contributes immensely to the economies of mostly tropical and developing countries. At a global scale, it is estimated that more than a 100 million farmers benefit from the coffee supply chain in some way, additionally, roughly 2,2 billion cups of coffee are consumed at a daily basis worldwide [34].

Our project is a coffee monitoring system that will be able to capture data in real time from sensors, and thereafter store the data in a computer, after the data is analyzed it will be displayed on a multi-touch dashboard. Finally, the data will be displayed in form of graphs and statistics.

This paper will be discussing the project's process, in order to achieve its final result. The project goes by the name "Kaffeknekt". This highlights the main objective of the project. By developing a system that operates as a monitoring system for an espresso machine, enabling real-time tracking of daily usage and performance. The aim is to showcase Semcon Part of Knightec Group's technologies in a subtle yet effective manner that communicates their value to clients.

Initially, this report presents the project's background and the issue identified for resolution by the group. It then provides an overview of related work, highlighting how this project builds upon, what project framework was followed and contributes to existing research. Following this, the document presents the group's project methodology, and will eventually conclude with a detailed description of technical development part both software and electrical-wise employed throughout the project.

At the end of this report, introduce the risks that both the project an system could've experienced, and also the challenges that were met throughout the entire journey.

1.2 Bachelor group

1.2.1 Group Members **MMS** | *SC*

The Kaffeknekt project consists of an interdisciplinary team of both electrical and software engineering students, a group overview illustration can be found in Fig. 1.2.1

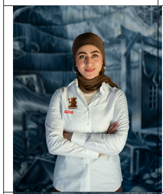
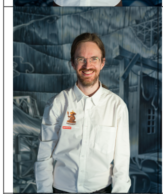
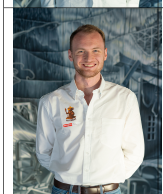
	Name	Discipline	Initials
	**Sokaina Cherkane**	**Software Engineering Student**	**SC**
	**Martin Taraldstad**	**Software Engineering Student**	**MT**
	**Didrik Aas Bergan**	**Software Engineering Student**	**DAB**
	**Abdulkadir Kabuk**	**Software Engineering Student**	**AK**
	**Mikolaj Marek Szczeblewski**	**Electrical Engineering Student**	**MMS**
	**Ivan Bergmann Maronsson**	**Electrical Engineering Student**	**IBM**

Table 1.2.1: Group members of Kaffeknekt

1.2.2 Initials **MMS** | *SC*

Throughout this document, the initials of both the author and a proofreader will be presented on the headings of every section. The document will be written by in total six members of the team, and it is vital that clarification and verification of every individually contributed text is satisfied, given that the project is graded individually. The format is of the following scheme: The author's initials are in a **bold** format, following along with a vertical line | which separates the author from the proofreader's initials.

1.3 About Semcon Part of Knightec Group **SC** | *MMS*

Semcon Part of Knightec Group (now officially, Knightec Group) is an internationally renowned technology company. Founded in 1980 in Västerås, Sweden, the company combines engineering expertise, digital services, and sustainability know-how to create solutions that prioritize value for people and the planet. Semcon operates in various industries, including life sciences, energy, industry, mobility, and the public sector [88]. Their focus is on product development based on human behavior which has been integral for multitude of projects. While it would be impractical to demonstrate every method they have employed in a single case study. Nevertheless, their emphasis on human behavior makes it ideal to illustrate these approaches with a product that a user interacts with daily in one form or another.

In 2024, Semcon merged with Knightec Group, forming the Semcon Part of Knightec Group, which positions itself as Northern Europe's leading strategic partner in product and digital service development. This merger expanded their expertise and delivery capabilities, enabling them to better support customers across various industries [90]. Through these initiatives and collaborations, Semcon demonstrates its commitment to developing technology that matters, focusing on sustainable innovation and human-centric solutions [88].

Over the years, Semcon Part of Knightec Group has undertaken numerous projects that exemplify its multidisciplinary approach. For instance, the company has worked on developing simulations for more realistic dimensioning of solar panel mounting systems, contributing to more efficient renewable energy solutions. In the life sciences sector, Semcon has been involved in projects like OnDosis, which revolutionizes the way patients take their medication by integrating digital solutions with pharmaceutical treatments. Additionally, Semcon/Knightec has contributed to the development of automated docking systems for airport equipment, enhancing efficiency and safety in airport operations [89].

1.4 Project Objective **SC** | *MMS*

The aim of this project is to create and deploy a real-time monitoring system for a coffee machine that continuously gathers key sensor data, specifically water temperature, pressure and power consumption, and presents these metrics on an intuitive, interactive dashboard. By capturing and visualizing live operational parameters, the system will allow users to observe the machine's performance and detect irregularities early. Serving as a proof of concept for Semcon, this implementation will illustrate how Industry 4.0 technologies, such as the Internet of Things (IoT), data analytics, and human-machine interfaces, can be harnessed to boost machine efficiency, improve the end-user's brewing experience, and support more sustainable practices through informed resource management. For more in depth information about the task description can be found in 2.1.

1.4.1 Coffee and technology **SC** | *MMS*

Developing a modern espresso machine involves integrating multiple technologies to improve both precision and the user's brewing experience. First, a network of sensors continuously measures essential parameters such as water temperature, boiler pressure, and flow rate; these signals are then digitized and sent to a central processing unit. In particular, a Microcontroller (MCU) that interprets the incoming data, running control algorithms that gives the user insight about the current state of the espresso machine. In order to maintain optimal extraction conditions. Which involves typically the temperature at 90–95 °C and a pressure of 9 bar. This will ensure each shot of espresso delivers consistent flavor and crema [66].
Beyond local control, IoT connectivity links the machine to a back-end server—either on a local network or in the Cloud—where user profiles, brewing logs, and diagnostic information are stored. This infrastructure allows users to preset drink strength, temperature profiles, and shot volumes via a web or mobile application, and to retrieve real-time status updates during operation [12].

An intuitive Graphical User Interface (GUI) is critical. It visualizes live sensor readings, displays recommended maintenance alerts and guides the user through custom drink recipes. Software modules manage secure data exchange, and execute real-time adjustments, thereby minimizing deviations in extraction yield and maximizing energy efficiency.
Altogether, the seamless combination of precision sensing, embedded control, IoT connectivity, transforms a traditional espresso machine into a smart appliance. This delivers a continuous high quality espresso while offering insights that contributes to sustainability and ease of maintenance.

Chapter 2

Background

The growing influence of Industry 4.0 has made real-time monitoring, data-driven analysis, and system transparency increasingly relevant, not only in large-scale manufacturing, but also in specialized domains such as espresso brewing.

Traditional espresso machines, while admired for their mechanical precision and craftsmanship, often operate as black boxes, systems the user must rely on without fully understanding the underlying brewing conditions. Yet, the espresso brewing process relies on tightly controlled physical parameters, where small deviations in temperature or pressure can significantly affect quality. By gaining a deeper understanding of the machine's thermal and mechanical behavior, the need for a modernized monitoring solution becomes evident.

This chapter explores the background for the development of a sensor-based monitoring system tailored to a commercial espresso machine. It presents the limitations of traditional brewing equipment, identifies the key parameters influencing extraction quality, and highlights the benefits of sensor integration. Finally, it situates the project within the broader context of Industry 4.0 — extending digital innovation into a field traditionally dominated by manual expertise.

2.1 The Role of Coffee in Modern Culture **SC** | *MMS*

The system will process data in the form of statistics and graphs, representing key factors that influence the final taste of espresso, e.g. the Percolation process which refers to the movement of water through the coffee grounds, influenced by key factors such as water temperature, pressure, and flow rate etc.

Additionally, the system is designed to be user-friendly, ensuring that everyone can easily understand and follow the provided guidance.

In the sections that follow, we will first look at the history of coffee, then explain the espresso extraction process, examine the ideal brewing time (generally 25–30 seconds), and finally detail the key parameters—extraction time, average flow rate, and espresso volume—that experts identify as critical for achieving a balanced, high-quality shot [3].

2.1.1 The History of Coffee **SC** | *MMS*

Coffee has been consumed for centuries, with its origins traced back to Ethiopia, where the bean was originally eaten, and Yemen, where it was first roasted and brewed, before eventually spreading into Europe [29].

Previously, coffee had a significant impact on several countries. Still, this beverage plant has a huge influence in all industries in today's society. Economically, culturally, socially, and nonetheless in the tech-world.

The coffee quality depends on a significant amount of factors, although for the project its narrowed down to the focus upon beverage preparation. The preparation process involves using either **Arabica** or **Robusta** bean, which are the most prominent coffee species in the market. The beans undergo a roasting process and become ground, this increases extraction surface which helps the water to absorb all the flavors.

Coffee has truly evolved throughout its long history, resulting in a variety of brewing methods which involve turning beans into beverage. The simple extraction created by heating water and grounds in a pot over a campfire. The art of brewing coffee is one that demands precision and finding the "sweet spot" that is difficult to realize without a visualization of the conditions occurring at the very moment of brewing [52].

Manufacturers noticed an opportunity in this market and responded by implementing profiling of pressure, thermal stability and flow rate along with its regulation. There is certainly a skepticism in the coffee world surrounding digital technology implementation in espresso machines, because it is considered a craft and art which is fueled by a barista's passion. In spite of that, this addition to the system would improve the overall user experience, by including the user in the brewing process, in addition to providing insight and control [63].

2.1.1.1 The espresso method **SC** | *MMS*

The espresso brewing method is involved with forcing a small amount of hot water through a tightly packed coffee-puck using high pressure. This process allows the water to be absorbed within the coffee grounds, contributing to the extraction of rich flavors, aromatic compounds, and essential oils. The result is a concentrated and well balanced espresso with its signature depth and intensity [2]. The process behind how an espresso machine brews by using this method is explained in 2.2.

2.1.1.2 Brewing time **SC** | *MMS*

Commercial espresso machines usually brew at a time interval between 20-30 seconds [52]. Like other brewing parameters, extraction time is influenced by multiple factors beyond just duration. Variables such as water pressure, temperature from the tank, and coffee grind size. All these factors play an important role in the final outcome. If the coffee grounds are too fine, the extraction process will take longer, resulting in an over-extracted espresso with an overly strong or bitter taste. However, if the grind is too coarse, water will flow too quickly, leading to an under-extracted espresso that is weaker and more acidic.

2.1.1.3 Water pressure **SC** | *MMS*

Water pressure plays a crucial role in determining the final taste of the coffee. For instance, if the pressure is **lower** than ***9 bars***, it can lead to **under-extraction**, resulting in a weak and unbalanced flavor profile. The espresso may taste sour, thin, and lacking in depth. This occurs because the water passes too quickly through the coffee grounds in the Portafilter, preventing sufficient extraction of essential flavors and oils.[3] The optimal pressure is at around 9 bar $\pm$ 1, [p.2, [3]]. However, if the water travels through the coffee with the correct speed and pressure, this will result in a **balanced extraction** of sweetness and acidity, which will serve an aromatic and complete coffee [3].

However, if the pressure **exceeds 10 bars**, it can lead to **over-extraction**, resulting in a burnt and overly bitter or harshly sour taste. This occurs due to the water forcing too many flavors from the coffee grounds, extracting undesirable flavors and negatively impacting the overall balance of the espresso [3].

2.1.1.4 Percolation **SC** | *MMS*

Percolation is a process where a liquid such as water slowly runs through a porous material. For instance, ground coffee extracts soluble substances. In the "coffee" context, percolation represents the process where the hot water presses throughout the coffee-puck in order to extract the taste, aromas and oils that will eventually result in the "brewed" coffee. The speed of water flow significantly affects the quality of extraction. If the Percolation is too fast, the coffee will be under-extracted as mentioned in 2.1.1.2. In contrast, if it is too slow, over-extraction occurs, resulting in a bitter and overpowering flavor. Achieving the right balance in flow rate is critical for ensuring a well-rounded and flavorful espresso.

2.1.1.5 Flow rate **SC** | *MMS*

In the context of "espresso extraction", the flow rate could be defined as the speed at which water passes through the compacted coffee puck in the Portafilter, typically measured in grams per second (g/s) or milliliters per second (ml/s) [35]. The pump generates higher pressure, which results in increased flow rate of the water, and conversely, a lower pressure corresponds to slower flow rate. The permeability of the tamped coffee ground plays a key role in determining the actual operating pressure and the resulting flow rate through the coffee puck, since it allows the water to easily pass through it. This in turn, affects the brewing time that the water's residence time within the coffee grounds, and also the overall quality of the brewed espresso [35].

It is not easy to define the flow rate among other factors since they, themselves, are dependent on a chain of parameters, such as the type of the espresso machine, coffee beans, and the grinder.

Table 2. Domain-based quality thresholds.

Quality Variable	Low	Optimal	High
extraction time (s)	<20	[20–30]	>30
volume (mL)	<20	[20–30]	>30
flow rate (mL/s)	<0.67	[0.67–1.50]	>1.50

Figure 2.1.1: DBAT

According to the figure 2.1.1, if the flow rate is less than 0,67 mL/s, it is considered too low. This could mean that the coffee is too finely ground or packed too tightly, making the water struggle to pass through.

If the flow rate is between 0.67 and 1,50mL/s it is optimal , which corresponds to the preferable range. If it is higher than 1.50, this means that the coffee is coarsely ground or not packed enough, allowing water to rush through too quickly, resulting in a weak espresso.

2.1.1.6 Water temperature **SC** | *MMS*

Based on the study done by the Graduate School of Bioresources in MIE University in Japan, their research focused upon comparing real brewed coffee with an already brewed model, while showing similar spectral tendencies.

pH alongside with the temperature affects the infrared absorption spectra, in this case the pH value of the brewed coffee is at 4.7. The spectral shifts occur due to temperature change which is impacting the molecular interactions within the coffee. The results of the study-test suggests that the pH, along with temperature, plays a role in how the coffee components interact and how its characteristics, e.g flavor and aroma, has the ability to change the results in different circumstances and conditions. [39].

2.2 Espresso Machines: From Craft to Engineering **MMS** | *IBM*

Espresso machines have a long tradition associated with craftsmanship, the kind which relied on the barista's intuition, manual control, and of course, years of experience to adjust key parameters such as temperature, pressure and extraction time. The espresso culture is evolving there is no doubt about that, but how does this intertwine with newly appearing technologies?

This section will present an overview of the espresso machine technologies – in general and in the context of the specific machine used in this project. Additionally, we'll delve into the internal structure of them, what type of components contribute in brewing espresso, and their functionalities.

2.2.1 General Overview **MMS** | *IBM*

The very definition of what an espresso machine does is the following:

> An espresso machine brews coffee by forcing pressurized water near boiling point through a "puck" of ground coffee and a filter in order to produce a thick, concentrated coffee called espresso [117].

Now, there are different varieties of espresso machines, but in essence, they all brew the same coffee, although in different ways. Despite some of these machines being highly complex which include features that do not characterize a traditional espresso machine, the basic mechanism of all espresso machines is narrowed down to the following schematic illustrated in Fig. 2.2.1.

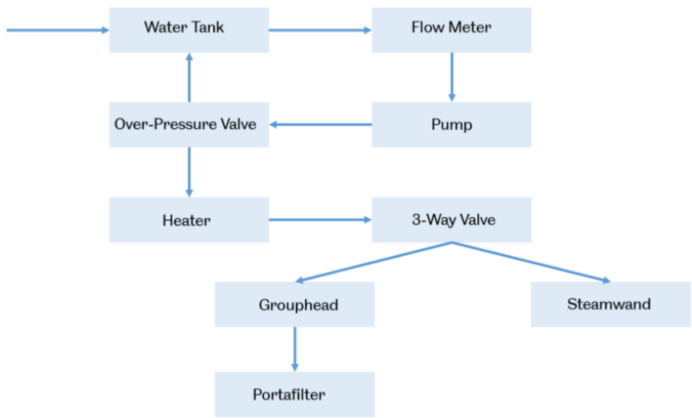


Figure 1.14 Espresso machine schematic

Figure 2.2.1: Basic schematic of a general espresso machine [126].

2.2.2 Design **MMS** | *IBM*

There is a puzzle at the root associated with the espresso machine designer's dilemma. Two vital components, espresso and milk, each prepared at different temperatures, how can they be delivered reliably and quickly? Espresso machines differ based on their Grouphead and boiler designs, and this is where the root lies.

Over time, various design solutions have emerged to balance performance, speed, and thermal stability, resulting in distinct categories of espresso machines based on their internal structure. This section explores the design principles behind these systems, focusing on how boiler types, Grouphead connections, and heating mechanisms impact usability.

2.2.2.1 Single Boiler **MMS** | *IBM*

An espresso machine of this variant has two temperature settings which correspond to either the brew thermostat or the steam thermostat as can be seen in Fig. 2.2.2. The boiler itself is utilized for both brewing and steaming, although not simultaneously, therefore for this design, the transition between brewing and steam temperatures necessitates a moment of waiting [17].

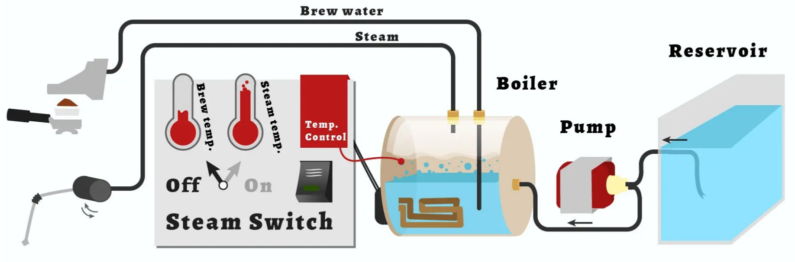


Figure 2.2.2: Notice that there is a switch which can set the temperature for either brewing or steaming, taken from [14].

2.2.2.2 Heat Exchanger — **MMS** | *IBM*

This design is a creative solution to the "two-temperature problem" associated with frothing milk and brewing espresso. The boiler as seen in Fig. 2.2.3 is filled up slightly above half of its volume, the remaining empty vacuum inside serving as the layer of steam. The Heat exchanger, which is a copper tube passing through the boiler, is responsible for heating fresh water from the reservoir to a very close brew temperature. The accuracy behind the desired temperature is influenced by the E61 Grouphead design, and how the temperature inside of it is affected when coupled with a heat exchanger [40].

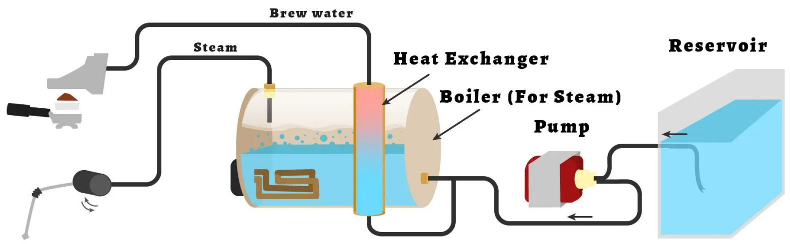


Figure 2.2.3: Notice the heat exchanger, the tube that crosses the boiler, and allows water to heat up to the desired temperature for espresso preparation, taken from [14].

[1]

2.2.2.3 Double Boiler — **MMS** | *IBM*

A more expensive solution, includes two separate boilers, each with their corresponding purposes for steaming and brewing, additionally, they have their own separate temperature controls and can be viewed in Fig. 2.2.4. This is an ideal candidate for continuous dispensing. Despite being the most reliable design, the heat exchanger solution usually is preferred as the most balanced one in terms of price and performance [14].

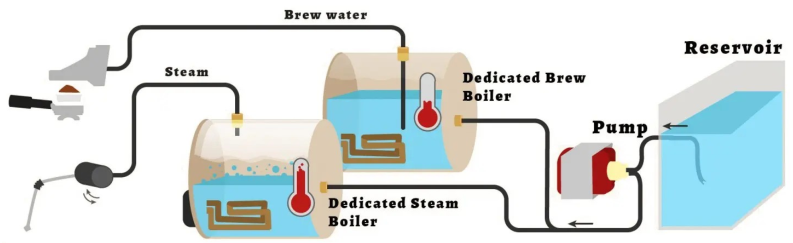


Figure 2.2.4: The double boiler

2.2.2.4 E61 Design — **MMS** | *IBM*

In 1961, Carlo Ernesto Valente revolutionized espresso machines when he introduced the Fabbrica Apparecchiature Elettromeccaniche e Affini (Faema) E61. The machine was fitted with an electric

[1] Throughout the report, you will see expressions such as either group or grouphead, note that they correspond to the same entity, check the glossary for more information.

pump which was operated by a simple on/off switch, this replaced the use of the lever which earlier corresponded to more manual work and a tiresome process.

Incidentally, the machine was described as ”semi-automatic”, now it was possible for the barista to maintain control over less in the whole brewing process, and focus instead on both the length and the parameters of the extraction. Instead of taking the water directly from the boiler, a pump was now providing it, pressurizing it and passing it through the heat exchanger before it reached the Grouphead.

The machine therefore was capable of an uninterrupted dispensing, it could produce a espresso shot after another without the necessity of reheating the boiler – every single espresso is produced under a pressure of 9 atmospheres, which is the standard today [28].

Some 30 years later, FAEMA's technical department, initiated along with Giugiaro Design, the development of an advanced product, the E91. At the time, it featured state-of-the-art technology such as programming functions for ease of use and yield, but it maintained its tradition which stretched from the E61 [13].

2.2.3 Operation **MMS** | *IBM*

Espresso machines are designed to carry out a time-sensitive and precise brewing process, which involves water being heated and pressurized before being pushed through finely ground coffee. Typical brewing cycle works by drawing cold water into the machine, heating it to the desired temperature, and finally, delivering it at a consistent pressure.
In this subsection, the general mechanical and thermal operation of espresso machines is explained for context on sensor-based monitoring for the user's understanding and insight.

2.2.3.1 Thermosiphon system **MMS** | *IBM*

While the former summarized explanation captured the basics about how the E61 machine brews espresso, it overlooked a critical part of the espresso machine which is concerned with heat management – that is the Thermosiphon system which can be viewed in Fig. 2.2.5.

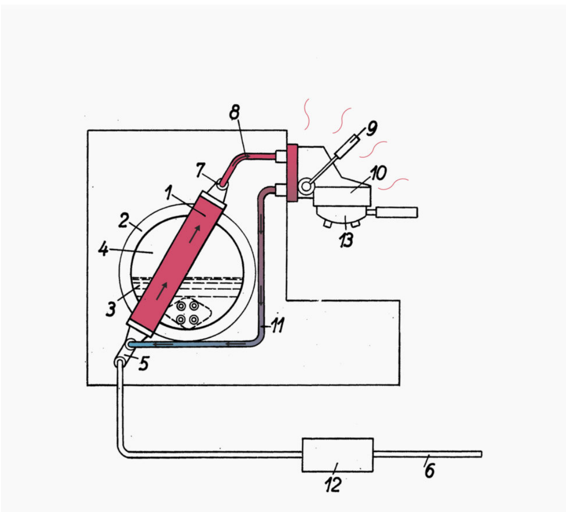


Figure 2.2.5: Illustration of Ernesto Valente's 1960 patent on the thermosiphon system [42].

The Thermosiphon system is relatively comprehensible: Cool water has a lesser density than warm water. In a closed system, water that is heated rises and water that cools, will sink. The very first known patent for a Thermosiphon was realized by Thomas Fowler in 1828, he pioneered the basis for the first central heating systems where they were involved with water being heated in a basement of a building, rising and delivering heat to radiators in the rooms above. As the loss of heat energy from the radiators to the air occurs, this results in the cooler water in the system becoming denser, sinking back to the basement to restart its loop through the boiler again [10].

A Thermosiphon is defined as a device that utilizes a method of passive heat exchange based on natural Convection, this involves circulating a fluid without the necessity of involving mechanical, or electrical pumps to do all the work of moving the fluid through an open or closed-loop system. In other terms, a Thermosiphon relies on the varying pressures created by density differences in the water. The procedure is as such: cold water is pumped into the bottom of the Heat exchanger, the valve in the group opens. However, because water flows towards the group through both pipes, both hotter and cooler water combine. Prominently in a lot of espresso machines, specifically our own, the groupheads are made in part with brass [42].

Brass, which is an alloy of both copper and zinc, has the properties which contribute to brewing temperature stabilization. As an alloy its durable, but it also has a considerably favorable thermal conductivity. It is also resistant to corrosion, unlike both aluminum and copper which are prone to it in wet environments [16].

Chapter 3

Project Management

This project represented a complex and interdisciplinary undertaking — combining electrical and software engineering with mechanical understanding, and requiring a structured yet flexible approach. The diversity of technical domains involved introduced both challenges and opportunities, highlighting the need for a project management approach that could adapt to evolving requirements while maintaining structure and momentum.

To navigate this complexity, the team employed the Agile Scrum methodology. Its iterative nature allowed the team to work incrementally, respond to uncertainty, and continuously refine both technical solutions and project goals. Weekly sprints, task boards, and retrospectives provided the foundation for transparent collaboration and regular progress evaluation.

This chapter describes how the project was managed from planning to execution. It details the distribution of responsibilities, the evolution of user stories into system requirements, the management of risks and dependencies, and the strategies used to align the team's work with the project's broader vision. In a project where both hardware constraints and software logic had to work hand in hand, the project management strategy became a critical part of the engineering process itself.

3.1 Planning Phase **MMS** | *IBM*

The planning phase was essential in shaping a shared direction for the project and creating a structure that could support interdisciplinary collaboration. At the outset, the project scope was only partially defined, and several technical unknowns — such as the condition of the espresso machine and the feasibility of sensor integration — made it clear that flexibility would be critical. To manage this uncertainty, the team focused on defining high-level objectives, aligning on expectations with the task-giver, and outlining a strategy for how the project could evolve over time.
One of the first priorities during planning was to identify the disciplines involved and distribute responsibility in a way that leveraged each team member's area of expertise. Parallel to this, the group held discussions around methodology selection and concluded that an Agile-inspired approach would best support the project's iterative nature. This decision allowed the team to embrace change, test early ideas, and refine the system design incrementally.
The planning phase also included setting up collaborative infrastructure — such as Azure DevOps for task management and shared documentation environments — and developing a high-level project timeline. These planning efforts provided the team with enough structure to begin working efficiently, while leaving room for the system and project model to evolve as needed.

3.1.1 Choosing a Methodology **MT** | *DAB*

When it came to selecting a methodology for the project we had some key requirements. We wanted to have room for failure and have opportunity to experiment and learn from mistakes. We wanted transparency so we could closely monitor our progress so we could have the ability to quickly adapt to changing circumstances or new insights. We wanted focus on sustainability to avoid burnout (this was maybe chasing a dream, but we survived).
These criterias effectively ruled out rigid models like waterfall so we turned our attention towards agile methodologies.

3.1.2 Agile And Why **MT** | *DAB*

> "Agile is a method of project management that focus on dividing tasks into short phases of work, with frequent reviews of the project and adaption of planning mid-execution as needed. An agile approach is flexible and less rigid than other methods of project management." [96].

This mattered to us because we were learning as we went. We had to perform tests and gather feedback as the project progressed. It was a collaborate effort in which both our understanding of the problem domain and our knowledge of various technologies evolved. Doing all the planning up front would have limited our flexibility, and we would not have been able to adapt if the design requirements had happened to change.

3.1.3 Scrum **MT** | *DAB*

Agile is a broad philosophy. Scrum is a popular implementation of agile philosophy. Scrum provides a framework for doing agile work. The main pillars of Scrum are:

- Sprints
- Sprint planning
- Daily Scrum/Daily Standups
- Sprint Reviews
- Sprint Retrospective

> "Scrum is a lightweight framework that helps people, teams, and organizations generate value through adaptive solutions for complex problems." [86]

On our journey to learn Scrum, we initially incorporated a selection of elements of the Scrum framework while omitting others. Introducing new roles such as Product Owner and Scrum Master created confusion as we already had an established group structure and a designated group leader. Maintaining and refining the backlog was a collaborative effort throughout the project.

3.1.4 Scrum Activities

3.1.4.1 Sprints **MT** | *DAB*

> "Sprints are the heartbeat of Scrum, where ideas are turned into value. They are fixed length events of one month or less to create consistency. A new Sprint starts immediately after the conclusion of the previous Sprint. All the work necessary to achieve the Product Goal, including Sprint Planning, Daily Scrums, Sprint Review, and Sprint Retrospective, happen within Sprints." [86]

In our case we chose to have sprint lengths of usually two weeks, sometimes extending to three ie. exam week and easter holiday. In the start of the project we tried to have a one week sprint, but since we had other courses to attend and did not work on the project mondays, we effectly had only four days for a sprint, which was not enough time to get anything meaningful progress done.

3.1.4.2 Sprint Planning **MT** | *DAB*

> "Sprint Planning initiates the Sprint by laying out the work to be performed for the Sprint. This resulting plan is created by the collaborative work of the entire Scrum Team." [86]

During the sprint planning, we discussed and reviewed which tasks were most imporant and which tasks that we thought we could complete within the given timeframe (Sprint).
We tried to keep our sprint planning sessions short, but at times they took quite a while and felt quite unproductive. This was due to lack of clear requirements and user stories, which often caused the planning sessions to blur into broader discussions focused on defining and refining these. Once we established clearer goals it was easier to shape a more concise and prioritized backlog. Which then reduced the time having to be spent planning. This improvement not only made our sprints planning more efficient but also allowed the team to do actual development and execution of tasks instead of discussing the time away.

3.1.4.3 Daily Scrum **MT** | *DAB*

> ”The purpose of the Daily Scrum is to inspect progress toward the Sprint Goal and adapt the Sprint Backlog as necessary, adjusting the upcoming planned work.” [86]

We have routinely held our Daily Scrum meetings at 9:15. Early in the project, this proved to be an effective way to ”take the temperature” of our progress and identify problems arising. It helped us support each other when encountering challenges. Although being contractually obliged to attend these meetings felt rigid, it ultimately became a good way to update each other and aligning the teams effort towards what tasks currently has highest priority.
However, as the project progressed and the team members became more independently focused of their specific tasks, the consistency of our Daily Scrums declined. When each of us knew what we were going to work on the Daily Scrum felt like a waste of time and instead we just had ad-hoc check-ins if problems arose. Nonetheless, the early dicipline helped us set a strong collobarative tone and helped us build a routine of respecting the projects as our job.

3.1.4.4 Sprint Review **MT** | *DAB*

> ”The purpose of the Sprint Review is to inspect the outcome of the Sprint and determine future adaptations. The Scrum Team presents the results of their work to key stakeholders and progress toward the Product Goal is discussed.” [86]

We have had regular meetings with Semcon where we have continuously refined the requirements of the system and how to progress toward the goal of fulfilling these.

3.1.4.5 Retrospective **MT** | *DAB*

> ”The Scrum Team inspects how the last Sprint went with regards to individuals, interactions, processes, tools, and their Definition of Done. Inspected elements often vary with the domain of work. Assumptions that led them astray are identified and their origins explored. The Scrum Team discusses what went well during the Sprint, what problems it encountered, and how those problems were (or were not) solved.” [86]

At the end of each sprint, we had retrospective discussions. These meetings were not always particularly productive. In the early stages of the project, reflecting on our previous discussions and planning felt like another opportunity on understanding the tasks at hand. However, as the project progressed and our task was clearly defined, the retrospectives began to feel like a waste of time - removing us from focused work just because we had a rule about having a retrospective every sprint.

3.1.4.6 How we started implementing scrum **MT** | *DAB*

The first weeks of getting used to scrum was quite chaotic and unorganized. Partly because we were still getting used to the ins and outs of scrum, and partly because we didn’t have a properly structured backlog. The biggest challenge we had was that our tasks was not derived from user stories and system requirements. This was natural since it was our job to figure out the system requirements, but we saw that we needed better structure so that none of us would be confused by what tasks needed to be done and which tasks to prioritize.

Azure DevOps had quite a learning curve in the beginning of the project. We struggled a bit with finding a way with tracking the time we spent working, and how much time we spent on specific tasks.

3.1.4.7 How we adapted to these problems **MT** | *DAB*

The first step toward organizing this chaos was to introduce the concept of epics and features in Azure DevOps (Fig. 3.1.1). This allowed us to group related work items under overarching categories, providing a clearer structure and better visibility into the project’s scope, as well as organizing other academic related tasks.

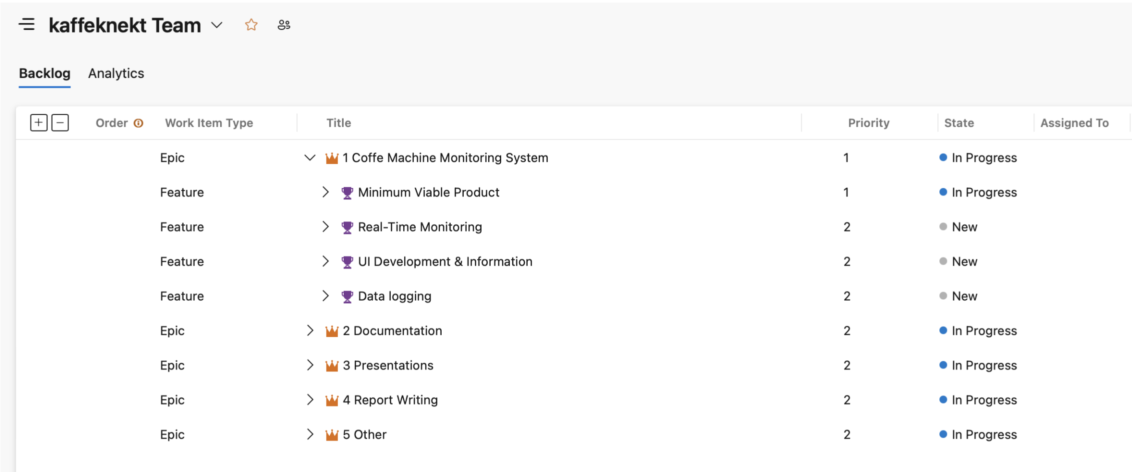


Figure 3.1.1: Epics and Features in Azure Devops

The next change we made was to represent our user stories as Product Backlog Item (PBI) in Azure DevOps (Fig. 3.1.2). When configuring the PBIs, we were able to add both a description and acceptance criteria (Fig. 3.1.3), mirroring the information found in our User Stories (Appx. A.2), Requirements(Appx. A.3), and Requirements Traceability Matrix tables(Appx. A.5.

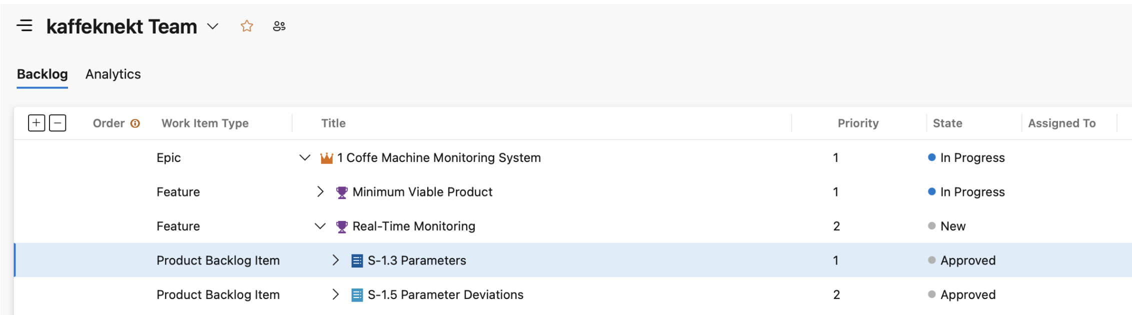


Figure 3.1.2: Product Backlog Item in Azure Devops

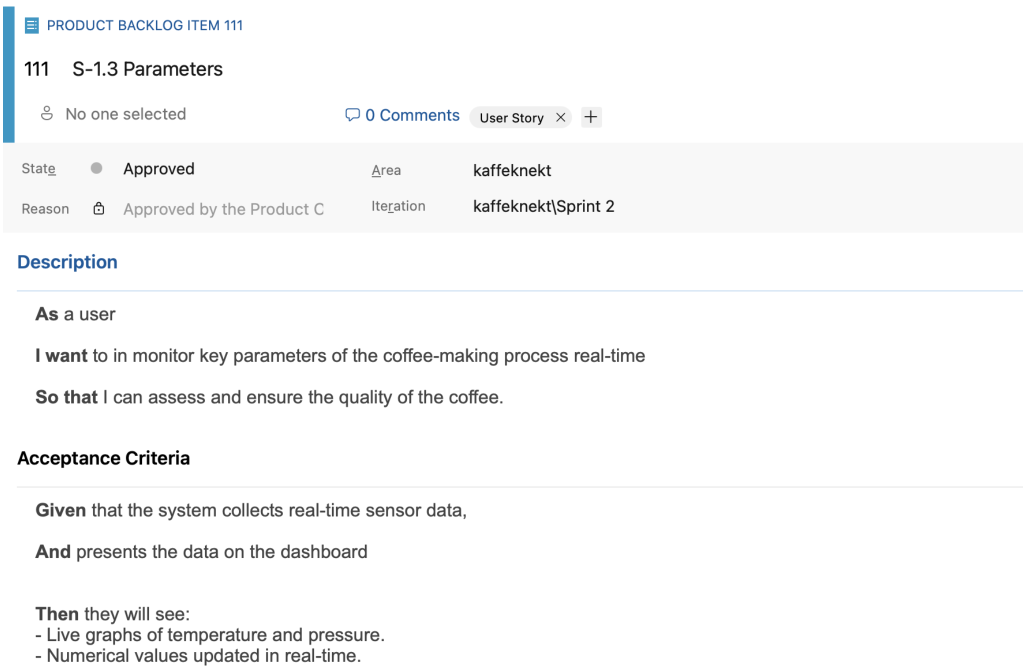


Figure 3.1.3: Description and Acceptance criteria of a PBI

In our task hierarchy, we have different kinds of PBIs. We have user stories represented as PBIs and we have work packages represented PBIs (work packages are just collections of tasks which not necessarily derive directly from user stories, but are tasks needed to be done to work towards the goal of fulfilling a system requirement). To differentiate these items we configured Azure DevOps to represent this using tags (Fig. 3.1.4).

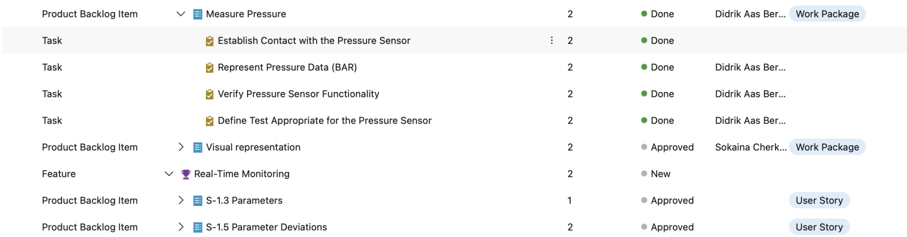


Figure 3.1.4: Tags to differentiate PBIs

Our first system of tracking time spent on tasks was based on tracking the time tasks spent inside any given state (Approved, Started, In Progress, Done). This felt like a clever and innovative way in the start, but we discovered it had its flaws. For one thing, members of the group forgot to drag tasks back to the Started state and instead left them in the In Progress state. This whole system was cumbersome and flawed in many ways so we had to rethink this approach because it became a root of frustration. After a period of researching different alternatives we found a solution that suited our needs.

Introducing 7Pace Timetracker, an extension for Azure DevOps.
By installing this extension in our Azure DevOps portal we were able to easily track time spent on backlog tasks. And we can easily generate statistics and overview of the teams efforts. It integrates neatly with both a desktop application (Fig. 3.1.5) and a mobile application (Fig. 3.1.6).

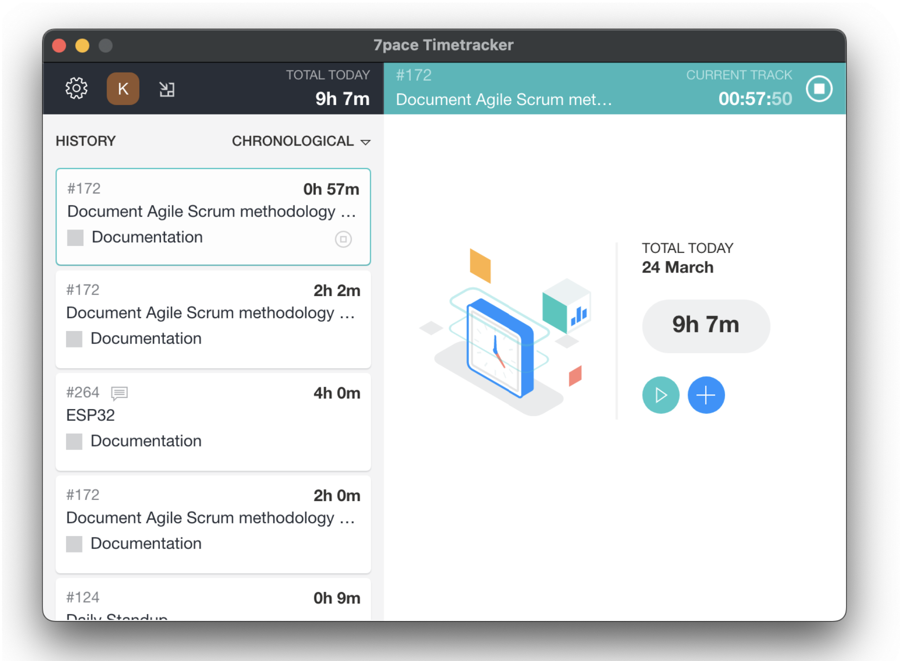


Figure 3.1.5: 7Pace Timetracker desktop application

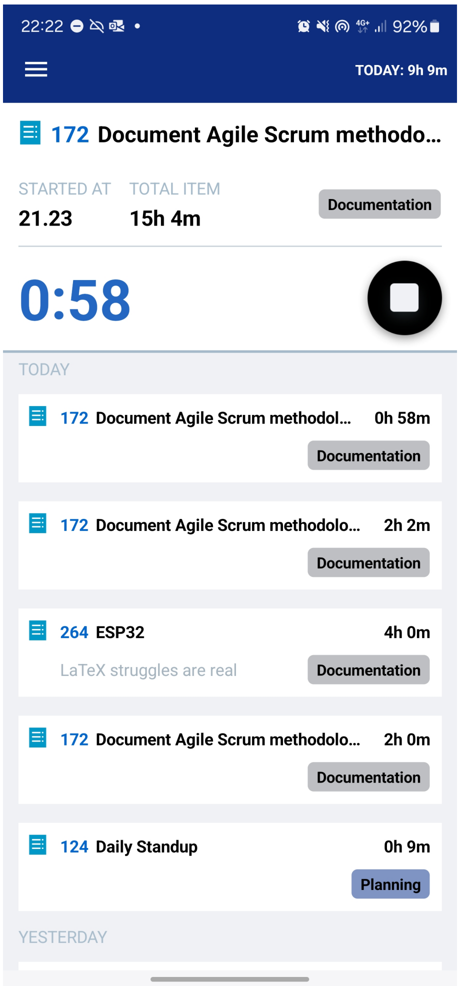


Figure 3.1.6: 7Pace Timetracker mobile application

Having done these changes, we now feel like we have a robust platform that will keep us on track towards our goals and provide us with transparency in our work efforts and traceability in the tasks being done.

3.1.5 Meetings — MMS | *IBM*

Meetings are a frequent and essential aspect of project collaboration. Their purpose ranging from enabling key decisions to be decided upon in a single session, to providing regular updates on the progress. It is by no means an understatement that meetings are a cornerstone of all project management schemes, they ensure that clear communication is maintained, that stakeholders are being included in the decision-making, such that the end result is in their interest. Without this scheme, customer's confidence and trust would be neglected towards the project's development and questions would arise in concern of the team's reliability.

3.1.5.1 External supervisor / Company — MMS | *IBM*

Semcon who is our stakeholder, maintains direct communication with us through Microsoft Teams 3.3.3. Concerns or feedback on the project's progress are addressed there, the formal meetings take place on Tuesdays either at Semcon's offices or at the campus.

3.1.5.2 Internal supervisor — MMS | *IBM*

The meetings with an internal supervisor are concerned with updates on the group's chemistry, progress, and how the project management inside the group operates. Weekly meetings are held every Tuesday.

3.2 Risk Analysis **IBM** | *MMS*

Risk assessment is the processes of determining risks which can occur while working on projects. Possible risks are shown in Tab. A.1.4, and Tab. A.1.5 for the product. And in Tab. A.1.1, Tab. A.1.2, and Tab. A.1.3 for the project. These tables are then assessed into categories with varying levels. The categories include the likelihood (odds of the risk happening), and the consequence of the risk. Both of these categories span over 5 levels each, ranging from very low and up to very high.

When a risk has been categorized, it will receive a risk rating utilizing Tab. A.1.10, which determines whether the risk is going to be an actual problem or not. If a risk rating reaches the upper limit, then the group should re-evaluate the rating that has been given to said risk.

3.2.1 Agile Risk Management **IBM** | *MMS*

We've tailored our project management to be fitted with an agile risk management scheme. The scheme is specifically tailored to conduct risk analysis at the iterative level, rather than the incremental level. For elaboration, the incremental level looks at a single significant feature milestone reached, unlike the iterative which is concerned with smaller releases at the end of every sprint. The iterative approach is better suited for the early phase of the project, given that there's insufficient information to evaluate risks properly. [8]. The iterative approach gives us the opportunity to edit and change our risk analysis at any time. Making updates to either the probability and consequences levels update periodically. This being whether we notice that some values were at unacceptable values, or that new risks were able to be added.

3.2.2 Risk Assessment Matrix **IBM** | *MMS*

The risk matrix uses both the consequence level matrices, and the probability level matrix to create a new table, that cross-references both tables as shown in Tab. A.1.10. Cross-referencing these tables gives us risk values that span from 1 to 25 (very low - very high). Our risk matrix is divided into five grades. VERY LOW, LOW, MEDIUM, HIGH, and VERY HIGH. Five grades were selected so that the table could be dealt into a condition, where in theory, it would be more workable.

This is how the grades of the matrix can be defined as:

- **Very Low:** Mitigation strategies don't need to be considered.
- **Low:** Mitigation strategies can but don't need to be considered.
- **Medium:** Mitigation strategies should be considered.
- **High:** Mitigation strategies should be taken.
- **Very High:** Mitigation strategies must be carried out.

Also seen in Tab. A.1.11. Again the amount of grades and the definition of said grades are all up for interpretation, since it usually changes between projects.

3.2.3 Consequence Level Matrices For Product And Project **IBM** | *MMS*

The utilization of a consequence table is useful when it comes to creating a risk analysis for a project. The consequence table is used to show what would happen to the project/product when a certain level is reached. These levels spanning from minimal consequences, and up to catastrophic consequences. There will be two consequence tables used in this report. One for the final product, and one for the project, as shown in Tab. A.1.6 & A.1.8. These tables show the level severity, description, and what we define these descriptions as.

The reason we utilize two different consequence tables is because they operate on different levels if both probability and risk. For example if our product stops working the project might still be a success as the product can be replaced, while if the project gets cancelled the product serves very little purpose. So it made sense to separate them to have better control over the risk and mitigation we can implement.

3.2.4 Probability Level Matrices For Product And Project **IBM** | *MMS*

The probability matrix is concerned with categorizing the level scale ranging from 1 to 5 (1 being unlikely, 5 being frequent), it is an essential risk assessment tool for acknowledging likelihoods and the severity of risks. The most important advantages of this tool are to both enhance safety and quantify risks. In this document we have created two probability matrices. Product shown in Tab. A.1.7, and project in Tab. A.1.9. Both tables share the same definition to level values, but deviate when it comes to the probable frequencies for both project and product. This being that the frequency of a probability for a functioning product, differs from a team working together, *i.e.* the probable frequency for a project, is more likely to happen then the product.

3.2.5 Risk Mitigation Strategies **MMS** | *IBM*

We've implemented mitigation strategies for our risks related to the project, by utilizing Agile Scrum for managing deadlines but also ensuring cooperation and cross-discipline knowledge sharing. Risks concerned with security have also been taken into account, it is imperative that we follow a guideline which protects our files, documentation, and of course our devices which we will be using for the duration of this project.

3.2.6 Agile Scrum Integration With Risk Assessment **MMS** | *IBM*

In Agile Scrum, it is inevitable that new risks will be discovered and identified, therefore it is necessary to acknowledge and track them on every Sprint planning. Daily-stand ups will be concerned with addressing immediate risk updates. And finally, at the retrospectives, they will be reviewed and reassessed, based on tests conducted. Risk assessment in relation to the project management scheme opens transparency within the group in terms of uncertainties.

3.3 Tools **IBM** | *MMS*

Tools are an important factor when it comes to working on projects. The word tools, does not necessarily mean physical tools. But as a term used to describe what was used to help the project move on. The tools that were used during this period, varied on whether the whole team used them, or that few members used them. Here are the software tools that were used during this project. The Tools span from documentation, software development, figures, design, and communication between the team.

3.3.1 Overleaf **IBM** | *MMS*

Overleaf is a document editor that is based on the usage of LaTeX[62]. Overleaf is what we used to string all parts of the report together. Overleaf can be used online, and multiple people can work on the same document at the same time if a subscription is paid. Which Semcon Part of Knightec Group has covered for us. Overleaf is very helpful in its document process, where almost everything can be labelled and called upon between documents. In Overleaf, there is the possibility of creating tables and figures, with certain commands. And there are many tools on-line to help people with commands and how to make the "perfect" document.

3.3.2 Google Scholar **MMS** | *IBM*

Google scholar is a free search engine which stores a vast array of published scholarly literature. Its index includes peer-reviewed academic journals, books, theses, dissertations, conference papers and technical reports. It is intuitive to use and has easily accessible BibTeX formats [118]

3.3.3 Microsoft Teams **IBM** | *MMS*

Microsoft teams is a messaging application that has multiple uses. But we mostly used it to communicate with our external supervisor and Semcon Part of Knightec Group team when it concerned info and certain updates. Microsoft teams has more functions, as in document sharing, video calls, and calender planning within a team. [56]

3.3.4 Github

MT | *DAB*

> "GitHub is the single largest host for Git repositories, and is the central point of collaboration for millions of developers and projects. A large percentage of all Git repositories are hosted on GitHub, and many open-source projects use it for Git hosting, issue tracking, code review, and other things." [15]

We have opened a git repository at Kaffeknekt GitHub Repository. This will be our primary way to store and collaborate on our code-base. We also connected Github to draw.io so we can edit our diagrams and keep version control of them as-well.
We also use github integration in Overleaf so that we periodically get backups off all tex files and figures in case Overleaf servers go down.

3.3.5 Timeshift

MT | *DAB*

As a backup solution for both Linux workstations and the Raspberry Pi we used Timeshift.

> "Timeshift is a system restore utility which takes snapshots of the system at regular intervals. These snapshots can be restored at a later date to undo system changes. Creates incremental snapshots using rsync or BTRFS snapshots using BTRFS tools." [50]

This tool proved invaluable throughout the project allowing us to quickly recover from system failures caused my misconfiguring system files or accidential deletions. (For example: Fig. C.1.20)

3.3.6 ChatGPT

SC | *AK*

ChatGPT is an Artificial Intelligence tool that serves as a source of inspiration for our team. It is not intended for producing final written content and should not be regarded as an authoritative nor trustworthy source. It is only meant for refining ideas, offering suggestions, and providing preliminary guidance. Any information it generates must be supported by citations from reliable sources [61].

3.3.7 Discord

IBM | *MMS*

Discord is another communication platform. This platform is usually used in the gaming community. But is robust enough to be used for everything, we have opted to use discord as one of our main communication methods when it comes to the project.
In discord you can make "servers", and in these servers you can make voice/text channels. Here we can communicate with one another even though we are not at campus, or when we are at home. Additionally, we are able to send pictures that are either meant for our social media or our report. [26]

3.3.8 KiCad

MMS | *IBM*

KiCad is a free and open source program for Electronic Computer-Aided Design (ECAD), which utilizes both design and simulation of electronic hardware for Printed circuit board (PCB) manufacturing. It is used across a large developer base as a hobbyist electronic design program. CERN noticed the potential of this program and contributed resources to foster open hardware development, such that KiCad could be as competitive as the most expensive ECAD programs out there [119].

This program is chosen by us because it is user-friendly, intuitive to learn and above all, free for all users. Since the project consists of two electrical engineering students that are eager to further advance their skills in ECAD design, this is a very powerful tool to realize ambitious projects involved with that domain.

3.3.9 Azure DevOps

MT | *DAB*

> "Azure DevOps supports a collaborative culture and set of processes that bring together developers, project managers, and contributors to build software. It allows organizations to create and improve products at a faster pace than they can with traditional software development approaches [57]."

We used Azure DevOps to manage our backlog and to track our hours spent on backlog items. We only touched the tip of the iceberg of what Azure DevOps provides. We were only interested in the project management parts like Boards and Sprints .

3.3.10 PlatformIO **MT** | *DAB*

Throughout our education we have primarily used Arduino IDE for developing code on microcontrollers. While the Arduino IDE is compatible with ESP32, we wanted to explore a more modern and flexible alternative.

> ”PlatformIO is a cross-platform, cross-architecture, multiple framework, professional tool for embedded systems engineers and for software developers who write applications for embedded products [65].”

Since much of our development took place directly on a Raspberry Pi, the ability to write code using a lightweight text editor, flash the ESP32 via the terminal, and monitor the serial output — proved to be both efficient and refreshing compared to the more clunky Arduino IDE.

We also installed PlatformIO as a plugin in Visual Studio Code (VSCode). This runs quite slow on the Raspberry Pi put performed good on the linux PC’s. While being able to use PlatformIO through the command line was beneficial because of its light weight nature a more intuitive way to use PlatformIO was using the VSCode plugin. This plugin provided all the features we needed and every member of the team did not have to memorize all the commands needed to get a workflow going.
The configuration file of PlatformIO project is named ”platformio.ini”. It defines the target platform (in our case ESP32), baud rate and other parameters if needed. The most powerful feature is the lib_deps keyword. This keyword handles the management of libraries, be it external libraries from github or local files on the development system. For example importing the the library for DFRobot-DS323X (the rtc module used in the system) - was done by adding the DFRobot’s github link under the lib_deps keyword as shown in Fig. 3.3.1. PlatformIO then handles the importing of this library automatically and the RTC functionality is available to use.

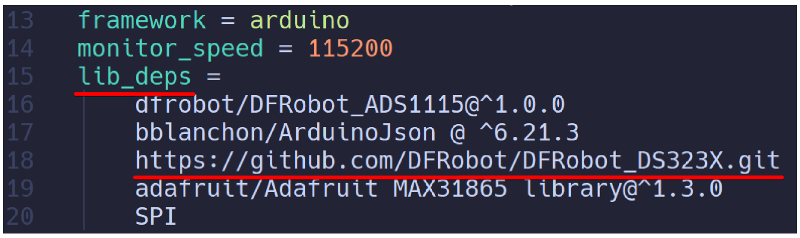


Figure 3.3.1: Importing library.

3.3.11 Qt Creator **SC** | *AK*

Qt Creator is a feature-rich, cross-platform integrated development environment Integrated Development Environment (IDE) that supports embedded software development and offers a wide range of tools for mobile application development [73].

3.3.11.1 What is Qt **SC** | *AK*

Qt is a complete development framework that provides a set of tools designed to simplify and speed up the creation of applications and user interfaces across desktop, embedded, and mobile platforms [71].
Qt, as a Cross-Platform Application Framework (CPAF), was developed in 1990 by Eirik Chambe-Eng and Haavard Nord, Norwegian programmers under the company name Trolltech. The framework became later available to the public in 1995. The company changed its name to The Qt Company and is based in Finland [103].

Qt C++ is the best fit for this project because it is a powerful, cross-platform framework that is specifically designed for developing graphical user interfaces (GUIs). It offers several advantages and features [72] such as:

Performance and Efficiency, C++ is known for its high performance and low level control over system resources. This is particularly important for real-time applications like your espresso machine dashboard, where fast data processing and visualization are crucial.

Cross-Platform Development, In Qt, one has the ability to write code once and deploy it on multiple platforms without significant changes. [46] In this project, it ensures that the same application code can be used both on the Raspberry Pi 5 and on the Windows PC, making the development process more efficient.

Rich in GUI Elements, Qt provides a wide range of pre-built components and tools for designing interactive, responsive, and visually appealing user interfaces. This is especially helpful for creating the real-time dashboards that will display sensor data and statistics in an intuitive way. For more information about GUI, check this section 5.3.4.1.

Real-Time Capabilities, Since this project involves real-time data, Qt's efficient handling of graphical rendering and real-time updates is definitely beneficial. The use of signals and slots in Qt allows for seamless integration of sensor data updates to the GUI without freezing or lagging the interface.
In Qt, signals and slots are a safe and flexible way for objects to communicate with each other. A signal is emitted when something happens (for instance a button is clicked or sensor data updates), and a slot is a function that reacts to that signal, allowing one part of the program to respond to changes in another, without the two being tightly connected [92].

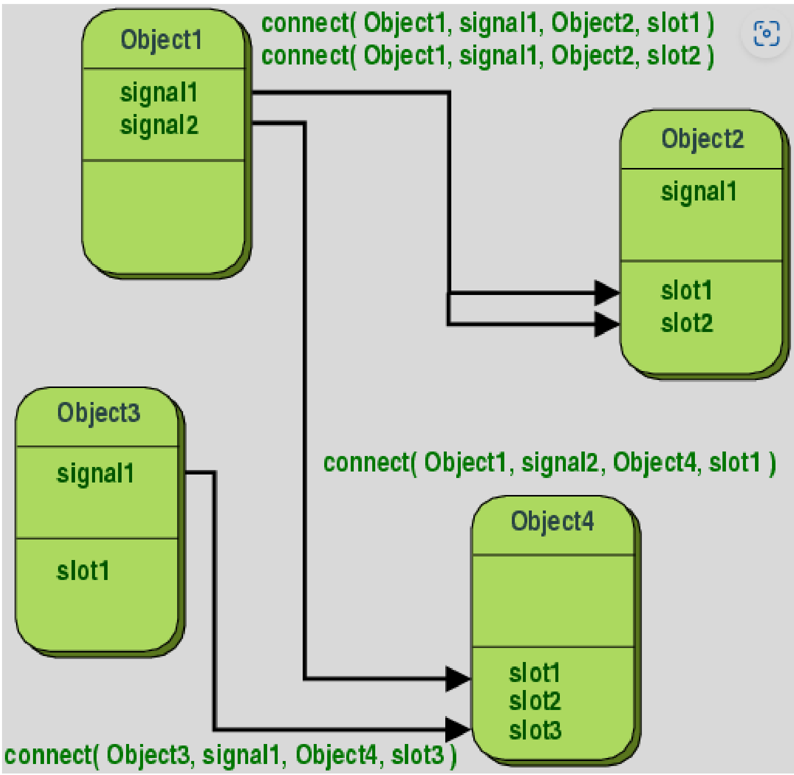


Figure 3.3.2: Visual Representation of Signals and Slots Communication in Qt, taken from [92].

3.3.11.2 Why use Qt Creator? **SC** | *AK*

Qt Creator is the official IDE for Qt, and it provides a range of features that make developing Qt applications easier such as:
1. **Integrated Environment**, where Qt Creator combines the features of a text editor, compiler, and debugger in one place, allowing the user to write, compile, and test the code efficiently.
2. **Cross-Platform Support**, Qt Creator simplifies cross-compiling for platforms such as Raspberry Pi 5 (RPI-5).
3. **Code Completion and Debugging**, With features like intelligent code completion, error detection, and step-by-step debugging, Qt Creator assists its user to catch issues early, making sure that the application runs smoothly. It also selects suitable debugger for each kit based on the

existing one in each system. This is especially beneficial when working with complex projects that require precision and efficiency [18].
4. **Visual Designer**. Qt Creator features a visual design tool that enables users to build user interfaces by dragging and dropping widgets and components directly onto the layout. This simplifies the process of designing and prototyping the dashboard, reducing the need for extensive manual coding. It is particularly beneficial when creating an interactive touch interface, as it allows for intuitive placement and arrangement of elements.

- **Qt classes for Qt Widgets 6.8.2** In Qt, the core building blocks for developing applications are provided as C++ classes. These classes offer the tools necessary for creating windows, responding to user actions, arranging interface elements, and presenting content on screen. For example, the core class, **QApplication** is responsible for running the event loop that keeps the application responsive to user input such as mouse clicks, key presses, and window events. [67]. While **QMainWindow** is a specialized class designed to create a main window with built-in support for elements like menus, toolbars, a status bar, and a central display area. [68].
QWidget, however, is the base class for all user interface (UI) objects in Qt. It represents a rectangular area on the screen and provides the basic features needed for building graphical elements in an application. Almost every visible element in a Qt GUI, like windows, buttons, labels, etc, is either a direct QWidget or its extension. [75]. **QWidget**, handles painting, event processing such as keyboard input, and lasso layout management. [75]

- **Widgets and Windows in Qt** The Qt Widgets Module offers a comprehensive collection of user interface (UI) elements designed for building traditional desktop-style applications. In Qt, widgets serve as the foundational components of a user interface. They can display information, reflect status updates, accept user input, and even act as containers for grouping other widgets logically. [74] When a widget is not placed inside another widget (i.e., has no parent), it is referred to as a window. [124] Typically, a window includes standard decorations such as a frame and a title bar. However, these elements can be customized or removed using specific window flags. Common examples of top-level windows in Qt include QMainWindow and subclasses of QDialog, which are often used to create the main application window or dialog boxes, respectively.
In this example from the project's code:

$$graphView = new\ GraphView(this);$$

GraphView is instantiated with this as its parent, which refers to the **MainWindow**. This means **GraphView** is a child widget, and it will appear inside and be managed by the main window.

3.3.12 Draw.io **SC** | *AK*

Draw.io or now known as diagrams.net is an open source web-based diagramming tool that can be used for creating flowcharts, network diagrams, Unified Modeling Language (UML) diagrams, ER diagrams, database schemas, and more. It offers a versatile platform for designing a wide range of visual representations [27]. In this project Draw.io was used to design and draw several diagrams that represents the functionalities and the system architecture of our project.

3.3.13 Google Sheets **IBM** | *MMS*

Google sheets [37] in short terms is, Microsoft excel but online. The main purpose we use google sheets is for the tables that we can create easily. These tables being Budget & Component list Appx. A.8, Requirements Appx. A.3, User Stories Appx. A.2, Test table Appx. A.4, Risk Analysis Appx. A.1, and Traceability Matrix Appx. A.5.

3.3.14 InfluxDB **AK** | *MMS*

InfluxDB is a time-series database that is designed for high ingestion rates of continuous data, making it well-suited for our operation.

More information about InfluxDB can be found in Sec. 4.11.1

3.4 Website **SC** | *AK*

In order to represent our project, including the stakeholders, the progress and also team members. A website was designed as a requirement from the university. Therefore, a lightweight site was developed and deployed on University of South-Eastern Norway (USN)'s servers, as seen in footnote 1. Where it operates with minimal maintenance. This was the chosen template "2137_barista_cafe" from the available collections on the website [108]. The template was later on modified and adapted to align with Kaffeknekt's visual identity by adjusting its HyperText Markup Language (HTML) structure and Cascading Style Sheets (CSS) styling.
[1]

3.5 System Requirements **DAB** | *MT*

In order to make sure that the product we deliver fulfills as many of our customers' wishes, we need to define them into testable requirements so that we know what can achieve within the scope of this project in terms of time, budget, and acquirable expertise.

3.5.1 User Stories **DAB** | *MT*

User stories were made to keep our customer in focus when developing our project and then infer what our requirements would be as user stories are made by placing ourselves in the shoes of the user. After doing some research, we also discovered that it is also a proven agile work method for a self-organized team like ours.

> "A self-organizing team, on the other hand, does not have an explicit requirements or design phase. When a team is self-organizing, it means that they work together to plan the project (instead of relying on a single person who "owns" the plan), and they continually come together as a team to revise that plan. A team that works like this typically breaks the project down into user stories or other small chunks, and starts working on the ones that deliver the most value to the company first. Only then do they start thinking about detailed requirements, design, and architecture[97]."

As we started to define our user stories, it soon became apparent that this needed to be a collaborative task. Our first attempt where only one member worked on it led to user stories that had undefined acceptance criteria and were too generic.

Our second iteration worked better as a result of this change, which we then used as the basis for the first requirements that we had our customer approve.
As the project developed, we realized that our most important user story needed to be broken down so that we could tackle it in more manageable parts resulting in our third; and smallest, iteration.(See Appx. A.2)

We could then use the new user stories to formulate a Minimum viable product (MVP) that would give our customer their most valuable features first before working on our more detailed features.

3.5.2 Requirements **DAB** | *MT*

Defining requirements for every project is of vital importance, as making a mistake could lead to the system we deliver not performing the functions our customer requested.

The first attempt at defining our projects requirements did not utilize the user stories as the group split their work effort into individual tasks, which predictably led to no cohesion between them. Realizing that this went against one of the main benefits for defining user stories, the course was quickly corrected for our next iteration.

After we had iterated on our previous user stories, those were used to further refine our requirements before we sent them to our customer for verification.

[1]The project's website can be accessed at: (https://itfag.usn.no/grupper/D03-25/)

Their response was overall positive, but they had some minor changes and wishes that we implemented while also refining our user stories to match our new insight of our end user.

Our final change to our requirements came with our customer wanting a scalable system that would allow for both visual expansions and physical additions. (See Appx. A.3)

3.5.2.1 Validation of Our Requirements **DAB** | *MT*

To assure that our system delivers functionality that fulfills our requirements tests need to be designed such that they validate one or more of our acceptance criteria.
Test Table: For our test table we opted to organzine them in two ways. The first was by adding a three letter code so that you can easily identify which module of the system you are looking at, while the other way was to separate our test into three types.
• The first type are the standard UNIT test, where a single function of a module were to be tested.
• The second type is functional test were the whole functionality of a system, but not any other modules that can change the result of the test.
• The third type is an integration test where one or more modules are tested together.
(See Appx. A.4 for the Test Table)
Traceability Matrix
To keep track of all requirement artifacts, we decided to create a traceability matrix which connected all artifacts in one clear table giving you a clear thread of which user story and requirement the test was derived from.
(See Appx. A.5 for the Traceability Matrix)

Chapter 4

System Development

This chapter explains how the different parts of the monitoring system were developed before they were combined into a complete solution. The project was divided into smaller sections, each focused on one area of the system, such as collecting sensor data, handling data in real time, and creating a user interface to display information clearly.

Each part was developed separately to make sure it worked as expected before connecting it to the rest. This approach helped the group work in parallel, reduce errors, and make adjustments as needed during development. Special care was taken to ensure that the system could collect reliable data and show it in a way that gives the user useful insight into what happens during brewing.

This chapter presents the rationale behind the architectural decisions, the selection of hardware and software technologies, and the methods used to ensure that each component fulfilled its functional role prior to full system integration.

4.1 System Specification and Architecture **DAB** | *MT*

This chapter delves into how we defined and validated our system requirements through user stories, testable requirements, and test strategies. Furthermore, it also describes the system architecture through various diagrams, from connections between key components to the entire system as a whole.

4.2 System Architecture **SC** | *DAB*

Software architecture is the high-level, executable description of how a distributed system's components interact, capturing concurrency, synchronization, data flow, and timing relationships, to enable early simulation, analysis of system behavior, and verification against formal constraints prior to full implementation. [51]

4.2.1 Overall Architecture **DAB** | *MT*

When designing a system, having an understanding of how the system interacts with each element in its environment is of vital importance. Keeping that in mind will help in terms of scalability and allow for solutions that tackle problems directly and beneficently.

4.2.1.1 Defining System Modules **DAB** | *MT*

As the requirements started to fall into place, the system could now be formulated into modules that handles specific tasks that our system needs to fulfill those requirements.

Main Modules: As a number of requirements for this system specify the need to measure specific values some module that handles the retrieval of this data would be needed and the need to store data locally has been specified so a module that handles this would also be required. Furthermore a module for displaying the data in some form to fulfill several other requirements.

Additional Modules: As the project developed some more modules were added. A visualization module was added to improve the user experience and to improve the clarity of the data represented. The other module came about as the need for a system that supports scalability so an embedded module was implemented to handle more sensors as the system expands.

4.2.1.2 Defining System Interfaces **DAB** | *MT*

When the modules were defined the next step would be to define how they should communicate with each other. Defining these interfaces between modules allows our team to work on a module and not having to gain a full understanding of the other modules it interacts with as long as the defined interfaces are upheld.

Embedded: The embedded module takes in sensor data of different protocols depending on their individual make. As for the interfaces between the embedded module and the data handling module utilizing a USB connection or wireless seemed like strong contenders. This interface was also later used by the visualization module.

Data Handling: For the interfaces between the data handling module and the data storage module depends on the type of data storage utilized, but the most common type would be a database. Later in the project a UNIX domain socket interface was utilized. (See Appx. C.2 for more information regarding this decision)

Data Storage: As the data storage modules function is to store data that will later be displayed an interface between these two seems appropriate. And as one of the user stories implies that the data will need to be accessible after the system is turned off, the device that retains this data would need to retain this for this situation as well. This would then set the need for some form of hard drive, further clarifying what specific devices the system needs.

Visualization: This module needs a way to display curated data to a user so an interface to a device that can fulfill this role and at the same time there should be a device that receives inputs from the user.

4.2.1.3 Our Solution **DAB** | *MT*

With all the interfaces and modules defined a tentative suggestion that evolved as the project matured. What is seen in figure. 4.2.1 above would be the final version of our system. (See Appx. B.9 for other diagrams that were used under the development of the system)

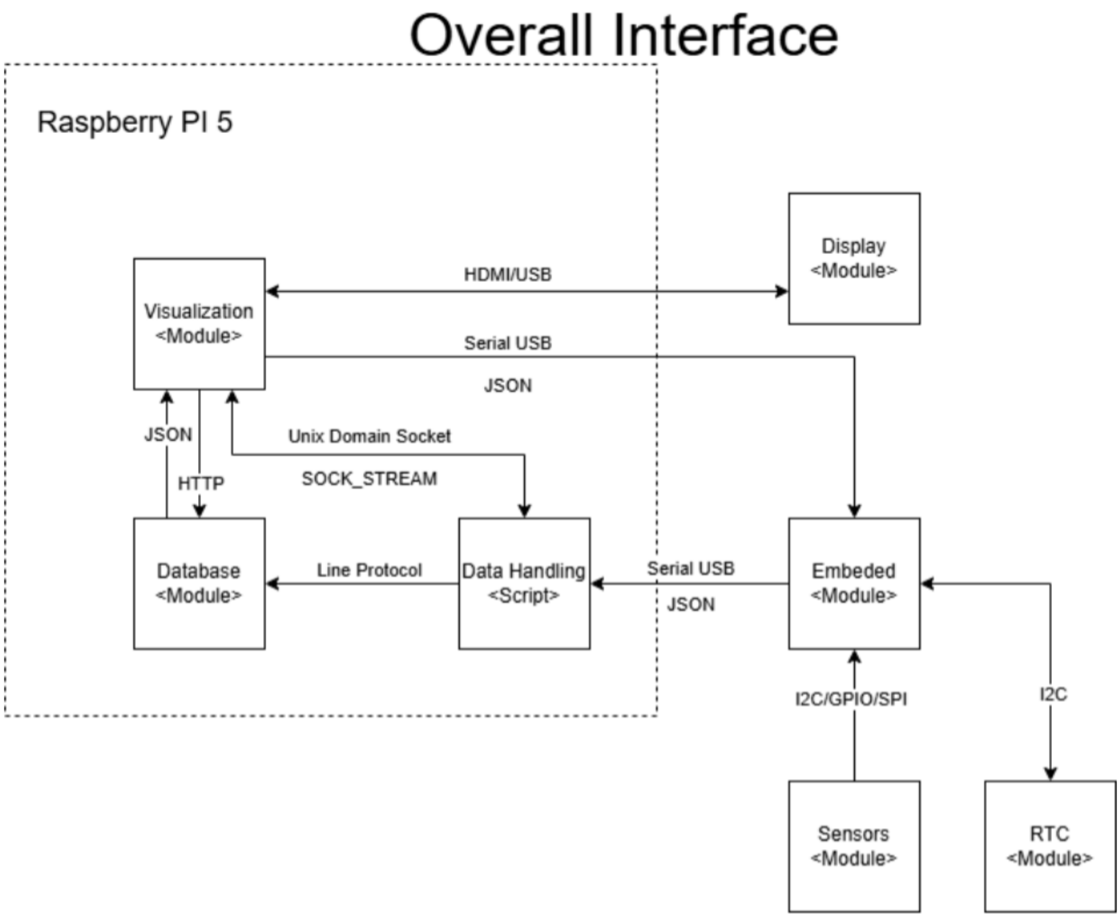


Figure 4.2.1: Final Overall System Interface

4.3 Hardware Overview

MMS | *IBM*

The hardware necessary for the monitoring system consists of following important components which are involved with visualizing, processing, and storing data.

4.3.1 Raspberry Pi 5

SC | *AK*

Figure 4.3.1: Raspberry-Pi 5 single-board computer [49].

The RPI-5 is the latest and most powerful single-board computer in the Raspberry Pi series. It features a 2.4GHz quad-core ARM Cortex-A76 processor, up to 8GB of LPDDR4X Random Access Memory (RAM), dual 4K High-Definition Multimedia Interface (HDMI) outputs, and significantly improved Input/Output (I/O) capabilities compared to its predecessors (Raspberry Pi, 2023). [82]. In this project, the RPI-5 serves as the central processing unit, collecting real-time data from water pressure and temperature sensors installed in an espresso machine. The data is stored locally on a Solid State Drive (SSD) 4.3.3 and is simultaneously visualized on a connected 15.6" QLED touchscreen using a Qt-based graphical dashboard. The RPI-5 upgraded performance makes it ideal for handling sensor input, local storage, and live graphical rendering without lag. Its compact size, low power consumption, and General-Purpose Input Output (GPIO) pins for sensor integration make it a cost-effective and efficient choice for embedded systems. Additionally, its compatibility with Linux-based environments and support for C++ development aligns perfectly with the goals of this Qt application. Overall, RPI-5 provides the hardware flexibility, processing power, and real-time responsiveness required for building an interactive, standalone dashboard system.

4.3.2 ESP-32

MT | *DAB*

Figure 4.3.2: Firebeetle 2 ESP32-E [20].

The ESP32 MCU series was first released in September 2016 [31].
ESP32 features a robust design. It is capable of functioning reliably in industrial environments with an operating temperature ranging from -40°C to 125°C [30]. It is engineered for mobile devices, wearable electronic devices, and IoT applications. Because of this it has very low power consumption [30].
ESP32 can interface through Serial Peripheral Interface (SPI) / Secure Digital Input Output (SDIO) or Inter-Integrated Circuit (I2C) / Universal Asynchronous Receiver / Transmitter (UART) interfaces, and it can interface wirelessly through Bluetooth or Wi-Fi [30].
ESP-IDF (Official IoT Development Framework)
ESP-IDF is Espressif's official IoT Development Framework for ESP32. It provides self-sufficient Software Development Kit (SDK) for development using programming languages such as C and C++ [30].
ESP-IDF supports a large number of software components, including Real-time operating system (RTOS), peripheral drivers, networking stack and various protocol implementations[30].
The most important aspect for us is that ESP-IDF features a RTOS Kernel (it uses FreeRTOS).

> "RTOS is a type of computer operating system designed to be small and deterministic. RTOSes are commonly used in embedded systems such as medical devices and automotive ECUs that need to react to external events within strict time constraints." [84]

> "FreeRTOS is a real-time Kernel (or real-time scheduler) on top of which embedded applications can be built to meet their hard real-time requirements. It allows applications to be organized as a collection of independent threads of execution. On a processor that has only one core, only a single thread can be executing at any one time. The kernel decides which thread should be executing by examining the priority assigned to each thread by the application designer." [81]

Using an RTOS on a RTOS capable MCU made us able to ensure accurate timing and scheduling and it ensured synchronization of tasks.

We chose ESP32 since it has a low cost, it has wide availability and it has strong community support among hobbyists and professionals. Our team is familiar with development on Arduino and ESP32 is in essence an Arduino with added real time capabilities like tasking and

4.3.3 Solid state drive (SSD) **MT** | *DAB*

Figure 4.3.3: Raspberry Pi 256GB SSD [76].

In pursuit of designing a robust system we had to consider the risk of data getting corrupted. Sudden power outages when data is getting written to SD-cards can cause corruption of data. This is currently based on anecdotal evidence and general knowledge that SSDs are able to withstand a much higher rate of read/write cycles of data. Since our system will continuously read and store sensor data we decided quite early in the process to go for this solution since the price comparison between a SSD and a industry grade microSD-card was not much difference. SSDs are also much faster than microSD-cards.

4.3.4 Touch-screen **SC** | *AK*

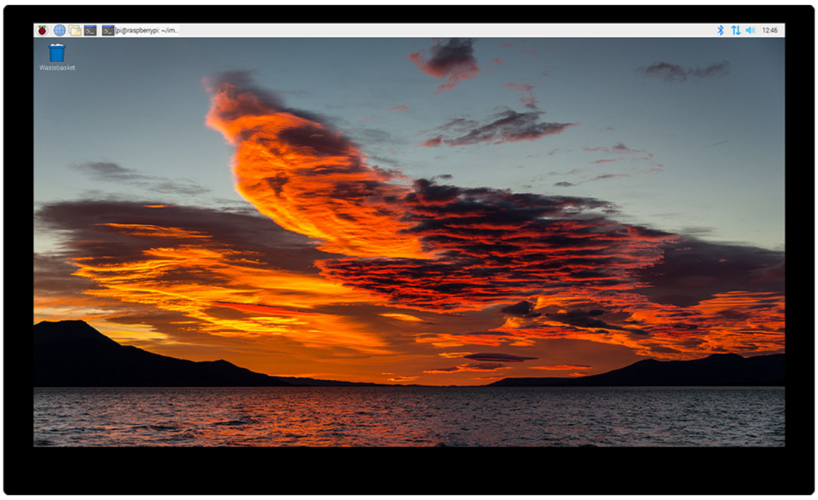


Figure 4.3.4: Raspberry Pi compatible 15,6“ HDMI QLED Touchscreen Display [77]

The 15.6” HDMI QLED Touchscreen Display for RPI-5 is an excellent choice for this project due to its size, display quality, and seamless compatibility with the RPI-5. This screen provides a full HD resolution (1920x1080) with QLED technology, offering vibrant colors, deep contrast, and wide viewing angles. It is ideal for visualizing detailed sensor data such as pressure and temperature graphs. Its large 15.6” size allows the dashboard interface, built with Qt, to present clear, readable graphs and interactive elements without crowding the screen. [109]
In terms of compatibility, the display connects via standard HDMI for video and Universal Serial Bus (USB) for touch input, making it plug-and-play with the RPI-5, which includes dual micro-HDMI outputs and USB 3.0/2.0 ports. The display is powered via USB-C, and no special drivers are required when running RPI-5 OS or other common Linux distributions, ensuring quick integration with the system.

4.4 Sensor Hardware

MMS | *IBM*

Our sensor hardware was selected based on the need for accurate and reliable measurements of key parameters. Each sensor was chosen based on its compatibility, user-friendliness, intuitive programming and strategic placements inside of the espresso machine.
A summary of the functional role of each sensor is presented below, although a more detailed account of component choice strategy – which involves among others, costs and redundancy plans, and can be found in Appx. B.1

4.4.1 Digital Pressure Sensor

MMS | *IBM*

Figure 4.4.1: NPI-19 I2C Digital Pressure Sensor [25].

The NPI-19 I2C Digital pressure sensor is a remarkable component capable of measuring highly stable, amplified, and calibrated pressure output, and it can be seen in Fig. 4.4.1.

It has an excitation voltage of 3.3V which allows for direct connection without the necessity of shifting the logic voltage levels down, given that most of the system runs on 3.3V. This type of sensor delivers digital data on the output in the form of the industry standard I2C protocol, which is elaborated in more depth in Sec. 4.6.0.1.

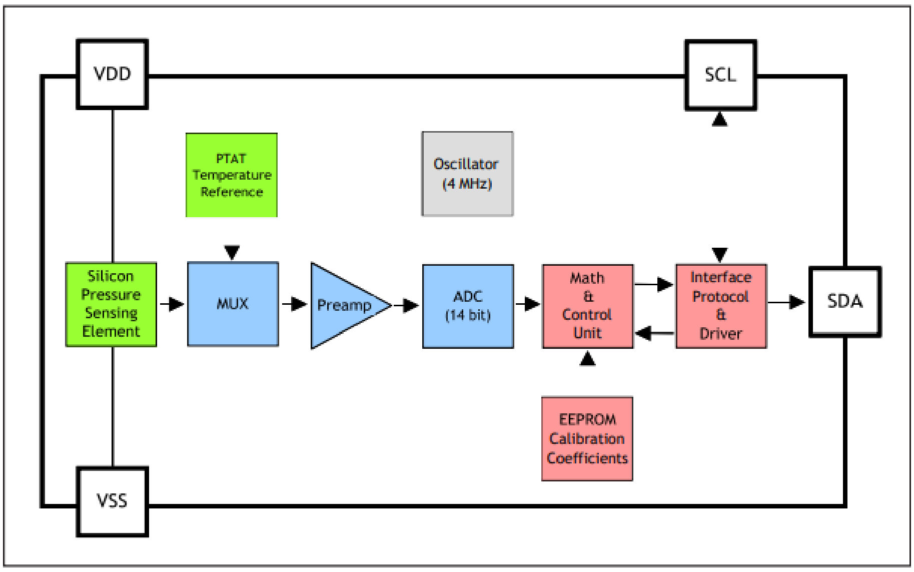


Figure 4.4.2: Block diagram of the NPI pressure sensor taken from [91]

The sensor is equipped with an inbuilt Analog-To-Digital Converter (ADC) with a 14-bit resolution; this is important as more bit-resolution contributes to more accurate output measurements. It tolerates temperatures from -40°C up to 125°C, which is ideal given that the temperature in the heat exchangers is above the boiling point.

Our specific sensor model is an NPI-19J-200G2, the operating range of measurement for the pressure is 1378.95 kPa, which roughly corresponds to almost 14 bar.

4.4.2 Resistance Temperature Detector PT100 **MMS** | *IBM*

Figure 4.4.3: The TSP-1PAG10305MZ Resistance Thermometer, taken from [110].

The Pt100 seen in Fig. 4.4.3 is a temperature sensor which contains a resistor that changes resistance which is proportional to the temperature change.

Their application is prominent in industrial environments, however, they are suitable wherever high accuracy is demanded. The measuring accuracy also depends strongly on the accuracy class, since this one in particular is of Class B. The classes tell what resistance limit deviations are allowed at certain temperature ranges and can be viewed in Fig. 4.4.4.

		allowed deviation in class A		allowed deviation in class B	
Measuring Temperature °C	Resistance value in Ω (Ohm)	Ohm Ω	Temperature °C	Ohm Ω	Temperature °C
-200	18,52	±0,24	±0,55	±0,56	±1,30
-100	60,26	±0,14	±0,35	±0,32	±0,80
0	**100,00**	**±0,06**	**±0,15**	**±0,12**	**±0,30**
100	138,51	±0,13	±0,35	±0,30	±0,80
200	175,86	±0,20	±0,55	±0,48	±1,30
300	212,05	±0,27	±0,75	±0,64	±1,80

Figure 4.4.4: Limit deviation table for pt100 for both class A and class B.

[1]

[1] The Pt in Pt100 stands for Platinum, while the "100" stands for the resistance value at 0 C

4.4.3 AC Current Sensor

MMS | *IBM*

Figure 4.4.5: AC Current Sensor 20A, produced by DFrobot, taken from [21].

The SEN0211 AC current sensor is a split-core current transformer, a unique design which allows for the core to be opened and closed around a conductor, without interrupting a circuit. This component is very user-friendly, non-invasive and easy for both installation and removal. Despite the hinged mechanism behind this component, air gaps introduced into the core, usually are not a significant problem, because of secure clamping, which also reduces disruptions to the magnetic path and the measuring accuracy. Once the core is closed, the primary current induces a magnetic flux, this in turn induces a proportional current at the secondary winding for the monitoring [111].

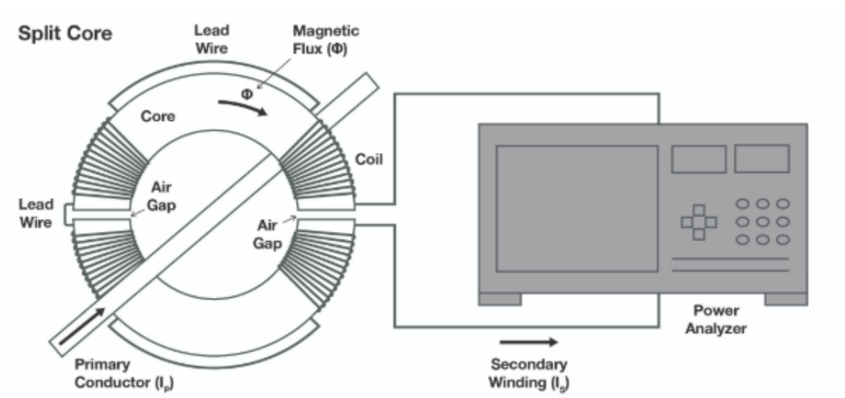


Figure 2. Illustration of a split core CT (with characteristic air gaps) in operation

Figure 4.4.6: Illustration of the split core current transformer, taken from [111].

4.5 Electrical Overview

MMS | *IBM*

To understand the monitoring system's function, it is important to look at the electrical system that connects all the key components. It connects all the components needed to measure, process, and send data from the espresso machine and can be vied in Fig. 4.5.1. This section gives an overview of the system using an electrical diagram and explains the role of each component. The goal is to show how everything is connected and why each part is important for the system to work.

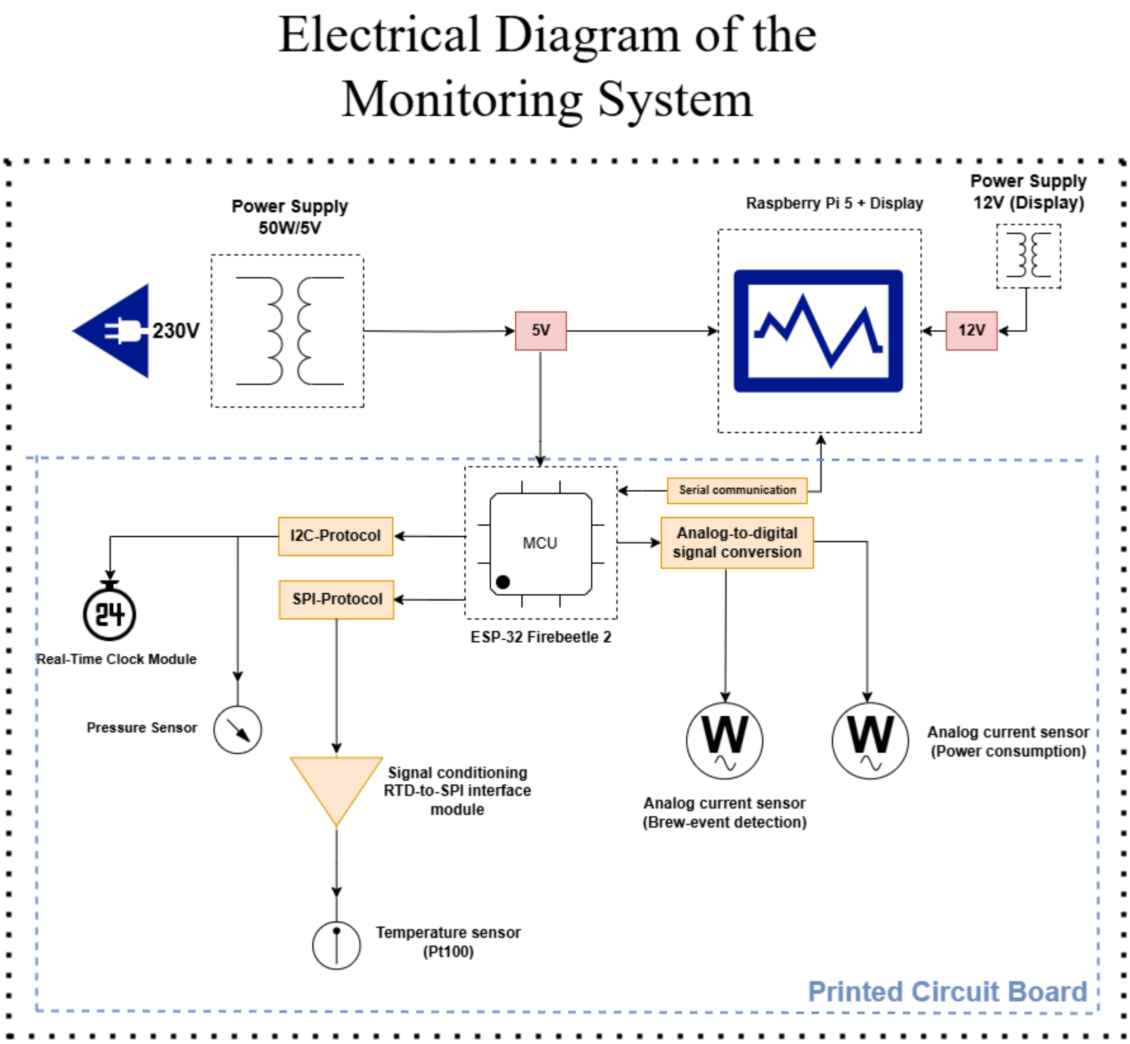


Figure 4.5.1: Electrical diagram of the monitoring system, with all components and modules. Designed in Drawio. Note the blue dotted area which is also on a printed circuit board layout.

The system is powered through a power supply which is connected from the espresso machine's two selected phase wires for 230V AC connection, transforming it down to a stable 5V Direct Current (DC). This further powers both the ESP32 MCU which handles the sensor data, and the RPI-5, which is the computer that stores that data and visualizes it on a display.

The ESP32's interface translates both I2C and SPI protocols to their interface-compatible sensors, additionally, the ESP32 is connected to an ADC with 16-bit resolution, capable of reading very accurate data from the analog sensors.

The blue dotted area covers the components that are on a designed printed circuit board, which the layout and design is presented in B.3.

4.5.1 The Espresso Machine

MMS | *IBM*

The espresso machine is a Faema due D92N/A-2H, it can be configured in different electrical configurations, either single phase, 3-phase 230V or 3-phase 400V. For this project its specifically configured for a 230V 3-phase connection and can be seen in Fig. 4.5.2.

Figure 4.5.2: Faema Due D92 Espresso Machine, the same model as our own machine [33].

4.5.1.1 Electrical Setup **MMS** | *IBM*

It was discovered that there is an electronic control board which is connected to the button panel for brewing espresso shots, in addition to other sensors, solenoids, relays, etc. Our customer emphasized strongly that despite the team consisting of two electrical engineering students, it is not worth both the risk and effort to try merging the electronic control board with our own system, as this may pose hazardous for both the person and the machine itself. More information about the control board is documented in Appx. B.6.

This particular configuration restricts the system to very specific power outlets which unfortunately are a rare sight at our university. The only place in the university where 3-phase-230V power outlets are to be found, are at the workshop.

4.5.1.2 Solenoid Valves **MMS** | *IBM*

Solenoid valves are prominent in espresso machines, their purpose to either fill the boiler, relieve pressure from different connections in the hydraulic system, or overall, control the flow of water.

There are both 3-way and 2-way solenoid valves appearing in espresso machines. They function nearly the same way, although they are installed at different places.

(a) Three-way parker solenoid valve [32]. **(b)** Two-way parker solenoid valve [32].

Figure 4.5.3: Solenoid valves typically used in espresso machines.

2-way solenoid valves have only an inlet and outlet, pressurized water is supplied to the inlet when the coil activates, the water is allowed to pass and exits through the outlet side. Unlike the 3-way solenoid, which includes a pressure relief discharge function once the valve deactivates, commonly used near Groupheads to relieve the pressure before removing the Portafilter [32].

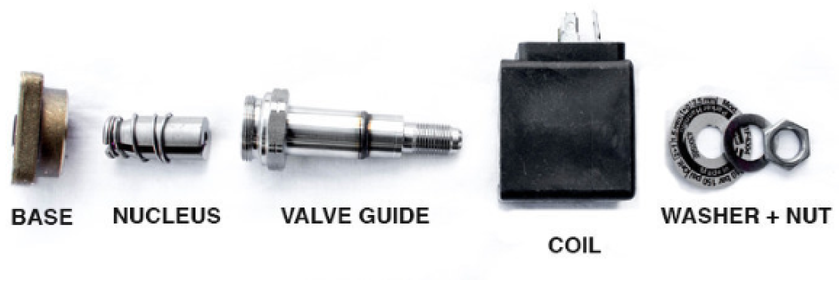


Figure 4.5.4: Components inside of a solenoid [32].

When the coil of a solenoid becomes magnetized, it pulls up the nucleus which allows the liquid to flow from the inlet to the outlet. The piston which seals the way from inlet to outlet is a nucleus.

4.6 Embedded Development

MT | *IBM*

This section explains the subsystem that is responsible of acquiring sensor data from the different sensors placed in the coffee machine. The system consists of an ESP32 MCU a RTC module and different sensors for measuring temperature, pressure etc.

The main responsibility of this subsystem is handling all the sensor data that comes in simultaneously and packages the sensor readouts in timestamped packages which then gets sent over USB. The RTC ensures the system has an accurate reference clock so the timing does not drift. Developing the embedded software has been an incremental endeavor to ensure the robustness of the system. We focused on testing one sensor at a time. After confirming the functionality and accuracy of sensors (and the RTC-module) - we started a system where we combined the sensors. We integrated one sensor at a time, testing for each step. The tests can be found in the test table Appx. A.4. The development log of the embedded software can be found in Appx. C.1.

In our system we have multiple sensors gathering data at any given time. We solved this by implementing multiple tasks running concurrently. Though it appears that tasks are being executed concurrently, they are in fact executed one at a time. see Fig. C.1.7 . ESP32 with FreeRTOS uses priority based scheduling preemptive scheduler. When tasks have the same priority it utilizes round robin scheduling algorithm [81].

Figure 4.6.1: Concurrency in ESP32. Picture taken from FreeRTOS [81].

The ESP32 also functions as a traffic officer, ensuring the flow of data does not create any conflicts. FreeRTOS utilizes mutexes to lock down critical sections of the code.

> "Any time memory shared between tasks is read or written, it creates a critical section of code, meaning code that accesses a shared resource (memory or device). The shared resource must be protected so only one task can modify it at a time. This is called mutual exclusion, often shortened to mutex. [60]"

The most critical section in the project was during the usage of the I2C bus. We had to lock down the I2C bus ensuring only one task could utilize it at a time.

4.6.0.1 I2C Protocol **MMS** | *IBM*

Under the careful consideration of components to decide on, it was important to find those that wouldn't pose difficulties in communication between devices. There are a variety of different communication protocols to choose from, although I2C is one of the most intuitive to use for software development.

I2C in simple elaboration, is a two-wire serial communication protocol which utilizes both Serial Data Line (SDA) and Serial Clock Line (SCL) lines. It was developed in 1982 by Phillips Semiconductor (now NXP Semiconductor) initially as a low-speed communication protocol. Since the protocol requires only two lines for communication it is widely used in the industry, in addition to that, its simple and economical for manufacturers to implement. SCL is the line concerned synchronous clocking of data either in or out of the target device, unlike SDA which transmits data to or out of some target device. I2C's most attractive attribute is its ability to establish communication between a controller device and a target device through a unique I2C address [125].

4.7 GUI **SC** | *AK*

This section presents the concept of a Graphical User Interface (GUI), explains its core functions, and demonstrates why an intuitive, touch-driven interface is essential for monitoring and controlling the espresso machine data in this project.

4.7.1 What is a GUI? **SC** | *AK*

In order to display the retrieved data, we need a touch screen, and for that we will be using a GUI.A Graphical User Interface (GUI) is an interactive platform that allows users to communicate and operate computers or digital systems using graphical elements such as windows, icons, menus, and touch-sensitive controls, rather than command-line interfaces (CLI) or text-based inputs [9]. A GUI is a user-oriented environment that leverages visual components, such as windows, icons, menus, buttons, and dialog boxes. This facilitates interaction with software without requiring knowledge of underlying command syntax or programming details. These elements typically respond to user actions (e.g., mouse clicks or touch gestures) in an event-driven manner, triggering corresponding functions when activated . In contrast to command-line interfaces, where users must enter precise textual commands at a prompt, GUIs present a more intuitive, discoverable means of control, reducing the cognitive load associated with memorizing commands and syntax. By abstracting low-level operations into graphical widgets, GUIs make complex tasks accessible to a broader audience and enhance overall usability [54].

4.7.2 Relevance to the Project **SC** | *AK*

The multi-touch desktop application's GUI was implemented in C++ using the Qt framework within the Qt Creator development environment. For this project, employing a GUI is critically important. The primary functionality of the application involves real-time visualization of data such as water pressure and temperature from a FAEMA espresso machine 4.5.2. By leveraging graphical representations that users can quickly interpret and monitor the performance of the espresso brewing process.
Additionally, the GUI facilitates various interactive features including an instructional button for espresso preparation, an informational button, and historical data storage functionalities. This interactivity, provided through touch-sensitive components, ensures the system is accessible even for users without technical expertise, thereby improving operational efficiency and user satisfaction.

4.8 Framework and Tools **SC** | *AK*

This section examines the technical foundations and implementation details of the graphical user interface. It begins with criteria for framework selection and the rationale for choosing Qt and its C++ binding, including a comparison of standard C++ object-oriented principles versus Qt's signal-and-slot model. The discussion then turns to the development environment.
An overview of Qt Creator and its key tools and features before presenting the GUI's architecture and implementation. This includes a high-level system architecture diagram, descriptions of the principal classes and components, an overview of important Qt packages, and a detailed explanation of how the code operates. Finally, the section concludes by illustrating how the GUI integrates with the broader system, tying together data acquisition, processing, and presentation.

4.8.1 Framework Selection **SC** | *AK*

Selection of an appropriate GUI framework requires careful consideration of four principal criteria. Functionality and user tasks, usability and accessibility, technical constraints, and overall user-experience (UX) impact.
Initially, the stakeholders proposed two options. A **JavaScript-based solution using React with Bootstrap UI and a Node.js server on the Raspberry Pi**, or a **Python implementation utilizing the Streamlit library**. However, because the firmware is written in C/C++ and the development team lacked formal training in JavaScript, both alternatives were deemed unsuitable. The project's requirements—real-time visualization of two key coffee parameters, display of historical data, and a fully touch-enabled interface including instructional, settings, and information buttons further narrowed the choice. Following an evaluation of cross-platform capabilities and compatibility with embedded C code, the Qt framework was selected. Its native support for C++ in the Qt Creator IDE facilitated rapid prototyping, seamless deployment on the Raspberry Pi as long as the developing environment and the RPI-5 share compatible compilers, and integration with existing code, leading to the decision to implement the GUI in C++ using Qt Creator. This approach offers several advantages, including robust event-handling through signals and slots, rich widget libraries optimized for touch interfaces, and straightforward cross-compilation for embedded targets.

4.8.2 Why Qt **SC** | *AK*

Qt is an open-source available framework designed to develop graphical user interfaces that operate across multiple platforms.[123]. The Qt framework supports the creation of user interfaces for desktop, mobile, and embedded devices using a single codebase, which streamlines development and maintenance.
It ensures backward compatibility between Qt 5 and Qt 6, enabling existing applications to be upgraded with minimal code changes. By leveraging native C++ execution, Qt delivers high-performance rendering and reliable control even on hardware with limited resources [100].
The comprehensive widget library, combined with the signal-and-slot mechanism, simplifies the implementation of event-driven interactions for both touch and pointer input. Moreover, Qt provides an integrated ecosystem that obviates the need for multiple external UI libraries, reducing dependency complexity and accelerating the learning curve [106].

4.8.3 Standard C++ and Qt's OOP Model **SC** | *AK*

Qt's C++ bindings are particularly well suited for both general GUI development and the requirements of this project. First, Qt enables true cross-platform deployment from a single codebase, simplifying the build and maintenance process for desktop, embedded, and touchscreen targets alike, an advantage demonstrated by its successful use in management systems for distributed simulation platforms running on Windows, Linux, and macOS solutions [112],[55].
Second, Qt's signal-and-slot mechanism and extensive widget library streamline event-driven programming, which was critical for handling real-time pressure and temperature updates without resorting to low-level threading or polling hacks. Third, because the ESP32 firmware and existing backend components were written in C/C++, leveraging a C++-based GUI framework ensured seamless code integration and straightforward cross-compilation, sidestepping the toolchain mismatches encountered with other languages. Finally, Qt Creator's integrated development environment including cross-compilation kits, on-board simulators, and real-time debugging tools

accelerated the iterative design and testing cycle, overcoming the cross-compilation challenges and Virtual Machine (VM) overhead otherwise associated with deploying to ARM targets[93]. Collectively, these benefits cross-platform consistency, event-driven simplicity, native performance, and cohesive tooling—make Qt for C++ a robust and fitting solution for the project's multitouch GUI.

4.9 Development Environment

4.9.1 Qt Creator Overview **SC** | *AK*

The development environment for this project is Qt Creator, a cross-platform integrated development environment (IDE) specifically designed for creating applications with the Qt framework. Qt Creator offers an advanced code editor with features such as intelligent code completion, syntax highlighting, and refactoring tools, which improve code quality and developer efficiency [70].
It supports multiple build systems and toolchains through configurable "Kits", enabling seamless cross-compilation for desktop and embedded targets such as the RPI-5 without leaving the IDE [48]. In addition, Qt Creator integrates debugging and profiling tools, including GDB and performance analyzers, alongside built-in simulators for rapid testing of GUI behavior on various form factors [48]
Context-sensitive help and bundled documentation ensure that developers can quickly access API references and usage examples during implementation[101]. Collectively, these features streamline the iterative cycle of coding, testing, and deployment, making Qt Creator an ideal environment for the development of the project's multitouch GUI.

4.9.2 GUI design **SC** | *AK*

The GUI design process began with a manual requirements analysis and brainstorming to identify the essential buttons and features mandated by the system's requirement 3.5. An initial prototype was then created using the open-source Penpot platform as seen in footnote 2 to establish a visual template and development roadmap. This prototype served as a reference during implementation, facilitating alignment between design intent and code. The first draft of the app is on B.8.
Throughout development, the GUI went through continuous changes and adjustments based on usability testing and additional feedback from Semcon 1.3, ensuring that both functional and aesthetic requirements were met.

4.9.3 Tools and Features **SC** | *AK*

Qt Creator's suite of integrated tools greatly accelerated development and debugging of the project's GUI. The advanced code editor, complete with semantic highlighting, on-the-fly syntax checking, and intelligent code completion[102], helped identify typos in custom widget implementations such as "the GraphView" class) before compilation, reducing runtime errors. In Design mode, the built-in Qt Widgets Designer allowed rapid prototyping of the main window layout (mainwindow.ui) and seamless promotion of custom widgets, ensuring that visual changes immediately reflected the project's interaction requirements. Configurable Kits enabled cross-compilation for both desktop and the RPI-5 target, eliminating manual toolchain setup, while the integrated debugger and performance analyzers were used to trace signal-and-slot connections and resolve latency issues during real-time data updates[99].Together, these features provided a consistent, efficient environment for coding, testing, and refining the multitouch GUI entirely within Qt Creator.

4.10 Architecture and Implementation **SC** | *AK*

This section details the GUI's internal structure and how its components interact to fulfill project requirements. It starts with a high-level overview of the system architecture, then examines the main classes and components that drive functionality. Next, key Qt packages and custom features are highlighted, followed by a walkthrough of critical code paths and behaviors.
The section concludes by showing how these elements integrate with external data sources and the broader application.

4.10.1 Overview of System Architecture **SC** | *AK*

The GUI is structured as a standalone Qt application running on the Raspberry Pi 5, communicating with both the ESP32 and InfluxDB through distinct channels. Live sensor readings are streamed from the ESP32 via a local WebSocket server and consumed in the GUI by the "QWebSocket" client, which parses incoming JSON messages and dispatches them through signals and slots for real-time chart updates[104].
Historical data is retrieved from InfluxDB using "**QNetworkAccessManager**" over , incorporating token-based authentication and bucket queries to populate trend graphs. Visually, the interface relies on Qt Widgets, such as "**QMainWindow**", "**QDialog**", "**QGraphicsView**", and "**QGraphicsScene**", to provide multi-touch-friendly controls and custom plotting areas [99]. This modular design cleanly separates data acquisition (network and socket modules), data modeling (controller classes), and presentation (widget hierarchies), ensuring that each component can be tested and maintained independently while leveraging Qt's native C++ performance and event-driven framework.

[2]

[2] penpot: is a design tool for design and code collaboration https://penpot.app/design

4.11 Database Development

4.11.1 InfluxDB **AK** | *SC*

The bachelor project requires us to efficiently store continuously flowing data so that it is available for real-time processing. This data will be utilized to generate dynamic graphs, which will be displayed on a screen for easy human analysis. Since we are dealing with simple, but high-quantity parameter data such as temperature and pressure caught by sensors, we can use a database that has high input and write abilities. With this we can generate and update the graphs as the data flows in, thus achieving the real-time aspect.

This is where InfluxDB comes into play. It is an open-source time series database where data can be pushed and stored in real time [43].
But what is meant by "time series"? Here are some key features the database has to offer regarding its operation and functionality:

- **Time series:** The data that is input is stamped with the point in time of when it happened, and the data is then called a "point". Collect a set of these points and now you have a "series" of data. The timestamp of the data is utilized as the primary key of the table for easy navigation, because when you want to find specific data, you can just filter the search by which point in time it was collected. Nice wordplay.
- **Retention policy:** When creating a database in InfluxDB, you have the option to set a retention policy. This means that you can choose if and when a database will automatically be deleted, but why would you want this feature? What possible value would deleting data bring to the table? We need to keep in mind that constantly collecting data over a long period of time will eventually cause storage issues. Old data that will not be beneficial to anyone is unnecessary, even counterintuitive to keep in storage. We can then benefit from this feature to efficiently manage storage space.
- **High input and write rates:** InfluxDB is designed to handle high rates of data input and write, and this makes it work well with IoT sensor readings.
- **Flexible table design:** When creating a table, there are variables that will be needed to be wary of: Tag set and Field set. They are used to label data appropriately, and contain tag key and tag value, and field key and field value, respectively. Essentially, the tag set specifies where the data is coming from(e.g. which sensor), and the field set specifies what kind of data is being inputted (e.g. temperature) [47]. Here is is an example of a pair of tag set and field set:

Timestamp	Position	T.Celsius
1747469053	Upper	27.1

Table 4.11.1: Simple example of an InfluxDB table

The table does miss default columns, such as an index column and a time column. A real example from our tests is available in A.9.7.
In a potential future scenario where we decide that one thermometer will be enough, the tag set will not be required, as we will not need which thermometer our only thermometer is.

The data that is stored, is done so by essentially taking time as reference, or "time stamping" it, and storing the data accordingly. What is cool about it is that you can more easily make graphs or charts out of those databases, something other database tools such as MySQL are less supportive of. InfluxDB is also designed for storing large amounts of data and does so at a higher rate, which is another attribute that, in addition to the support of graph setup, complements our operation.

To get started, you choose which version you would like to use (typically 2.7.x, or newer), and download and install the packages in the terminal. Then type this URL: https://localhost:8086 (local server) to open InfluxDB, and enter credentials for the following: Username, password, organization name, and bucket name. A bucket is another word for database, this one is the initial bucket that is made upon user creation and has no "retention policy", meaning it will never be deleted no matter how old.

After creating a user, a token will pop up on the screen, which is extremely important to copy to the clipboard as it will be used for a number of authorization areas.
These credentials will be used later for identification in scripts, commands and queries, thus it is best practice to save them somewhere.

4.11.2 Data Retention **AK** | *SC*

A bucket is where the data will be stored for later use, and data retention is how long data is retained in that bucket. The user can choose the duration of data retention upon creating a bucket, or updating an already existing one. When data becomes older than the set retention period, it gets deleted by the retention enforcement service , which by default runs every 30 minutes.

4.11.2.1 Shards **AK** | *SC*

InfluxDB stores data in *shards*, the fundamental unit of time-based data storage in InfluxDB. Shards partition the data within a bucket by time, making data storage and retrieval efficient. Each shard contains data within a specific time range, and stored in its own separate file. The duration is set automatically based on the duration of a shard group. Depending on the time range of a query, shards that fall out of bounds are then efficiently dismissed.

4.11.2.2 Shard Group Duration **AK** | *SC*

The shard group duration determines how much time a shard will cover. The duration, by default, is set based on the retention period of the corresponding bucket, but can be altered when creating or updating the bucket. Shard groups are what is being looked at when deleting data. As an example, of the bucket retention period is set to 7 days, and the shard group duration to 1 day, meaning the data is cut up in 1-day slices, then the data that gets deleted will be 7-8 days old, no more no less. It is calculated by the minimum retention period being the retention period itself, and the maximum is the retention period + the shard group duration. In shorts terms, the entire shard needs to be older than the retention period for it to get deleted
Shard group duration that is too large in comparison to the retention period might cause hick-ups and performance issues when data is deleted, since the bucket suddenly loses a big chunk of its data. It is recommended to have the shard group duration as default, or set around that duration.

4.11.2.3 InfluxDB Data Storage Model **AK** | *SC*

InfluxDB uses its own method to store data, which is significantly different than the relational MySQL tables we all are familiar with. In order to work with InfluxDB and the data that is stored, it is essestial to understand what data looks like, and how to effectively work with them.
A typical MySQL table consists of a group key that never holds the same value twice, column names explaining what kind of values they hold, and the rows that contain all inserted data from a single record, e.g. an environment reading. Ever:

ID	Temperature (C)	Humidity (%)
1	22	55
2	23.5	60

Table 4.11.2: A MySQL table

InfluxDB, on the other hand, stores data in time-series model and writes data in time-series points, as mentioned before. Each point must consist of a timestamp, a measurement name (similar to a table), and a field set; optionally, a tag set may be included. Every data writing operation will follow this schema - the line protocol:
measurement, tag_key=tag_value field_key=field_value timestamp

```
kaffeknekt@raspberrypi:~/kaffeknekt/database $ influx query 'from(bucket: "sensor_data")
        |> range(start: time(v: "2025-05-07T17:38:39Z"), stop: time(v: "2025-05-07T17:38:41Z"))
        |> filter(fn: (r) =>
                r._measurement == "Esp32Metrics")' --org Kaffeknekt
Result: _result
Table: keys: [_start, _stop, _field, _measurement]
                   _start:time                      _stop:time           _field:string     _measurement:string                      _time:time           _value:string
------------------------------  ------------------------------  ----------------------  ----------------------  ------------------------------  ----------------------
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z                    flag            Esp32Metrics  2025-05-07T17:38:39.000000000Z                       1
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z                    flag            Esp32Metrics  2025-05-07T17:38:39.500000000Z                       1
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z                    flag            Esp32Metrics  2025-05-07T17:38:40.000000000Z                       1
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z                    flag            Esp32Metrics  2025-05-07T17:38:40.500000000Z                       1
Table: keys: [_start, _stop, _field, _measurement]
                   _start:time                      _stop:time           _field:string     _measurement:string                      _time:time                  _value:float
------------------------------  ------------------------------  ----------------------  ----------------------  ------------------------------  ----------------------------
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z                   power            Esp32Metrics  2025-05-07T17:38:39.000000000Z                          1132
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z                   power            Esp32Metrics  2025-05-07T17:38:39.500000000Z                          1177
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z                   power            Esp32Metrics  2025-05-07T17:38:40.000000000Z                          1172
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z                   power            Esp32Metrics  2025-05-07T17:38:40.500000000Z                          1018
Table: keys: [_start, _stop, _field, _measurement]
                   _start:time                      _stop:time           _field:string     _measurement:string                      _time:time                  _value:float
------------------------------  ------------------------------  ----------------------  ----------------------  ------------------------------  ----------------------------
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z                pressure            Esp32Metrics  2025-05-07T17:38:39.000000000Z                   10.38000011
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z                pressure            Esp32Metrics  2025-05-07T17:38:39.500000000Z                   10.14000034
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z                pressure            Esp32Metrics  2025-05-07T17:38:40.000000000Z                   10.61999989
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z                pressure            Esp32Metrics  2025-05-07T17:38:40.500000000Z                   10.73999977
Table: keys: [_start, _stop, _field, _measurement]
                   _start:time                      _stop:time           _field:string     _measurement:string                      _time:time           _value:string
------------------------------  ------------------------------  ----------------------  ----------------------  ------------------------------  ----------------------
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z           readable_time            Esp32Metrics  2025-05-07T17:38:39.000000000Z     2025-05-07 21:38:39
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z           readable_time            Esp32Metrics  2025-05-07T17:38:39.500000000Z     2025-05-07 21:38:39
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z           readable_time            Esp32Metrics  2025-05-07T17:38:40.000000000Z     2025-05-07 21:38:40
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z           readable_time            Esp32Metrics  2025-05-07T17:38:40.500000000Z     2025-05-07 21:38:40
Table: keys: [_start, _stop, _field, _measurement]
                   _start:time                      _stop:time           _field:string     _measurement:string                      _time:time                  _value:float
------------------------------  ------------------------------  ----------------------  ----------------------  ------------------------------  ----------------------------
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z             temperature            Esp32Metrics  2025-05-07T17:38:39.000000000Z                   92.55999756
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z             temperature            Esp32Metrics  2025-05-07T17:38:39.500000000Z                   92.77999878
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z             temperature            Esp32Metrics  2025-05-07T17:38:40.000000000Z                   91.70999908
2025-05-07T17:38:39.000000000Z  2025-05-07T17:38:41.000000000Z             temperature            Esp32Metrics  2025-05-07T17:38:40.500000000Z                   90.26000214
```

Figure 4.11.1: Query from a flux script

```
  _field:string     _measurement:string                      _time:time                  _value:float
---------------  ----------------------  ------------------------------  ----------------------------
       pressure            Esp32Metrics  2025-05-07T17:38:39.000000000Z                   10.38000011
       pressure            Esp32Metrics  2025-05-07T17:38:39.500000000Z                   10.14000034
       pressure            Esp32Metrics  2025-05-07T17:38:40.000000000Z                   10.61999989
       pressure            Esp32Metrics  2025-05-07T17:38:40.500000000Z                   10.73999977

  _field:string     _measurement:string                      _time:time           _value:string
---------------  ----------------------  ------------------------------  ----------------------
  readable_time            Esp32Metrics  2025-05-07T17:38:39.000000000Z     2025-05-07 21:38:39
  readable_time            Esp32Metrics  2025-05-07T17:38:39.500000000Z     2025-05-07 21:38:39
  readable_time            Esp32Metrics  2025-05-07T17:38:40.000000000Z     2025-05-07 21:38:40
  readable_time            Esp32Metrics  2025-05-07T17:38:40.500000000Z     2025-05-07 21:38:40

  _field:string     _measurement:string                      _time:time                  _value:float
---------------  ----------------------  ------------------------------  ----------------------------
    temperature            Esp32Metrics  2025-05-07T17:38:39.000000000Z                   92.55999756
    temperature            Esp32Metrics  2025-05-07T17:38:39.500000000Z                   92.77999878
    temperature            Esp32Metrics  2025-05-07T17:38:40.000000000Z                   91.70999908
    temperature            Esp32Metrics  2025-05-07T17:38:40.500000000Z                   90.26000214
```

Figure 4.11.2: Zoomed in

Each field is stored in its own table of rows, instead of the typical columnar value identification we are used to seeing as a visualization from SQL queries. Below is an example of what a query of time-series data looks like:

4.11.2.4 Data Layout **AK** | *SC*

At the core of the InfluxDB data layout is the concept of a measurement, which is similar to a table in relational databases. Each measurement contains many series, and a series is uniquely defined by a combination of tag keys and values. Tags are string-based metadata used for identifying and grouping data; for example, in a telemetry system, tags might include values like device id, location, or sensor type. These tags are indexed by InfluxDB to make queries that filter or group by them highly efficient.
In contrast, fields represent the actual data values being recorded, such as temperature, CPU usage, or error rate, and are not indexed. This is an important detail in the data structure design, because indexing fields would reduce write performance. Since field values are often high frequency and constantly changing, InfluxDB instead focuses on efficiently storing and retrieving them in bulk.

Storage Engine
On disk, InfluxDB uses its own custom storage engine based on "Time-Structured Merge Tree"

files. These are columnar storage files made for compression and quick retrieval of time-series data. When data is first written to the database, it is temporarily stored in a "Write-Ahead Log", which acts as a buffer and helps ensure data durability. Over time, these writes are compacted and moved into immutable TSM files as part of InfluxDB's internal compaction process. [45]

4.11.3 Introducing Flux **AK** | *SC*

Flux is a powerful, functional, and dynamic query language developed by InfluxData for working with time-series data in InfluxDB 1.8+ and 2.x. It offers a flexible and consistent method to querying, transforming, and analyzing time-series data, making it optimal for use cases such as monitoring, alerting, and reporting.

4.11.3.1 Why Flux? **AK** | *SC*

Unlike traditional SQL-based languages, which are largely associated with fixed relational operations, Flux utilizes a functional programming model, allowing queries to be constructed as pipelines of several sequential operations. This design makes it particularly powerful for use cases that require dynamic filtering and data transformations.

The first lines of a Flux script
A Flux query begins by specifying a data source, but we will refer to a bucket as it is widely accepted as the typical data type. From there, the query defines a time range using the range() function, which is a required component in most Flux scripts. After that, we can start applying operations in sequence using the pipe-forward operator (—¿), which passes the result of one function into the next. This makes queries highly readable and modular, making them easier to edit in the future. For example, a simple query might filter for a specific measurement and tag, aggregate values over five-minute intervals, and compute a mean, all expressed as a chain of transformations in an understandable format.

4.11.3.2 Developing with Flux **AK** | *SC*

From a scripting perspective, Flux supports variable declarations, function definitions, and control points such as conditionals, which gives user the ability to create reusable and dynamic queries. This makes Flux not just a query language, but a lightweight scripting environment for building data flows. For example, users can define alerting logic that checks incoming data for thresholds and triggers events, or create tasks that select and transform data within custom time ranges at every given interval.

Flux Tasks
Flux also supports tasks, which are scheduled queries made to run automatically at defined intervals. Tasks are a core feature in InfluxDB 2.x, offering the user to be able to perform automated data processing, alerting, or downsampling without the need for external schedulers that call the scripts. A task is essentially a Flux script equipped with a scheduling plan, typically defined using a fixed time interval. For example, a task might run every 10 minutes to calculate averages, detect anomalies that fall out of expected value zones, or write downsampled data into another bucket for long-term storage. This allows for data flow automation, reduced storage costs, and maintaining performance by offloading compute-heavy operations outside of query time.

Chapter 5

System Integration

Once the individual parts of the monitoring system were developed and tested, the next step was to connect them into a complete and functioning whole. This chapter focuses on how the different parts from sensor hardware, embedded software to the graphical user interface – were connected and configured to work as a single unit.
System integration involved solving both practical and technical challenges. Sensors had to be installed inside the espresso machine such that it preserved safety and accuracy, data needed to flow smoothly between devices, and of course, the interface had to reflect real-time system behavior in a clear and reliable way.
In this chapter, the steps taken to combine hardware and software are described, how the system was tested during and after integration, and the adjustments made along the way. The result is a working monitoring system that provides insight into the espresso brewing process as it happens.

5.1 Hardware Integration **MMS** | *IBM*

This section focuses on putting the physical system together, connecting the sensors, powering them and making sure everything fit and functioned inside the espresso machine.

Both electrical and mechanical work had to be carried out, tasks included figuring out how to integrate new components into a modified system. Safe power distributiion, proper wiring, and reliable communication had to be ensured.

5.2 PCB Integration **IBM** | *MMS*

Integrating a PCB into the system had its own little hurdles. The idea of a PCB was set as a low priority in our requirements in the beginning. After time, we realized that we needed to make a system that was scalable, and the PCB was a good contender for this. The design principles of the PCB were simple, create a design that would fit into the system, communicate with the other electronics, and would be designed well enough so that it does not need any major redesigns when it comes to upgrading the system. With this reasoning, the priority of the PCB got increased, and we started the design process. The entirety of the PCB design process can be seen in Appx. B.3. This followed by the hurdles that came along with it.

5.2.1 Threaded Pressure Sensor

MMS | *IBM*

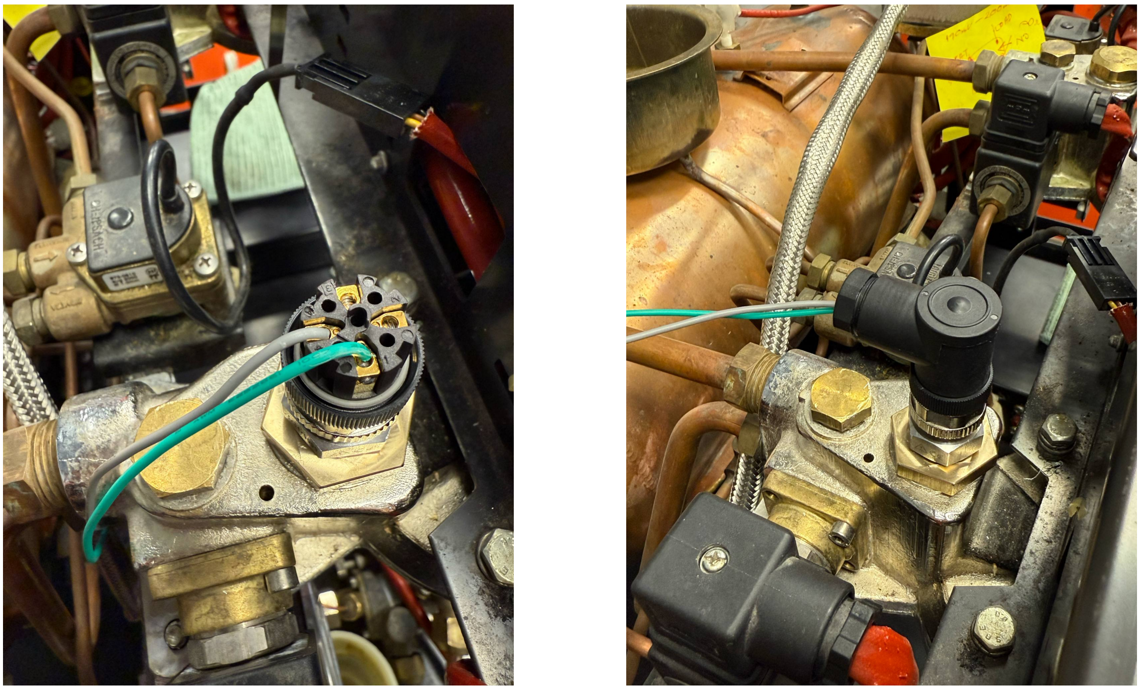
Once the correct threaded reduction couplings and adapters were acquired, the sensor could be installed inside the heat exchanger output. The first water pressure sensor actually never was tested under the espresso machine's brewing operation, since it was rated for a much lower temperature range than the temperature conditions occurring in the heat exchanger which are also influenced by the temperature inside the boiler, which rises up above 120°C.

(a) First pressure sensor installed early in the first component procurement phase.

(b) First pressure sensor, discarded from the system because of its non-functional requirements not being satisfied.

Figure 5.2.1: New I2C pressure sensor installed with a threaded adapter and sealed with a teflon tape.

5.2.2 Threaded Temperature Sensor **MMS** | *IBM*



(a) Terminals 1 and 4 which correspond to the 2-wire configuration of the RTD, are only being used. At room temperature, approximately outputting 110Ω.

(b) Fully assembled RTD connector.

Figure 5.2.2: PT100 RTD threaded temperature sensor installed inside the grouphead.

The PT100 RTD sensor was screwed inside a thread tapped and drilled plug, tightly wrapped with teflon tape to seal it and ensure no leakage occurring. The sensor has a probe length of roughly 30mm, which ensures measurements as close as possible to the brewing temperature.

More information concerning the installation of these threaded adapters and couplings is evaluated in more depth in Appx. B.4.

[1]

5.3 Software Implementation **DAB** | *SC*

When the initial components and software architecture fell in place, we could then start to develop software that would be able to satisfy our customers' requirements while working under the restriction our choices gave us, but still keeping in mind an agile work method so we can step back and reassess if the need arises.

[1]Special thanks to Joel from Semcon, who has been absolutely instrumental in providing mechanical expertise, especially related to threaded sensors, in addition to helping with providing the necessary threaded reduction couplings, both through suppliers and by own design.

5.3.0.1 Data Retriaval from the ESP32 **SC** | *AK*

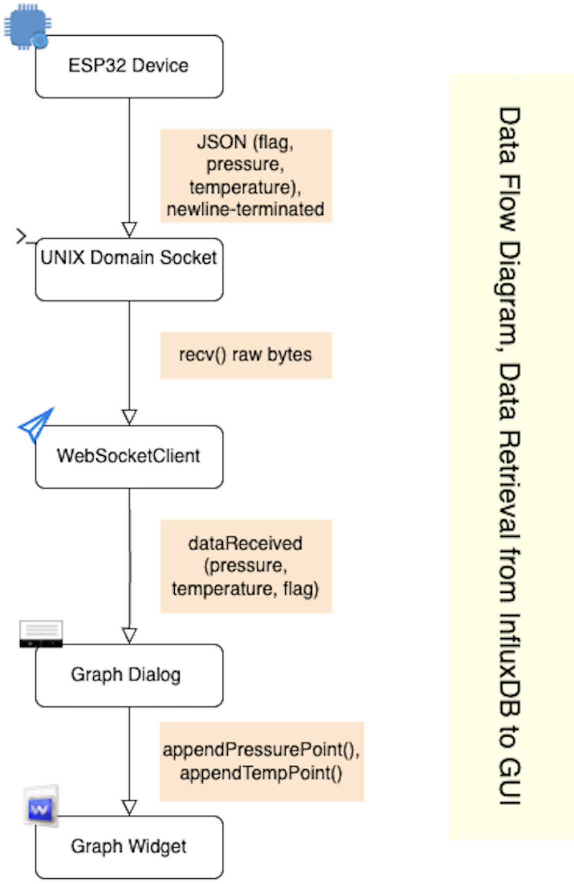


Figure 5.3.1: Data Flow Diagram of Data Retrieval from ESP32 to GUI. Designed in Drawio.

Real-time pressure and temperature readings are streamed from the ESP32 to the GUI through a Unix-domain socket acting as a lightweight WebSocket channel. On application startup, the **WebSocketClient** class (running in its own **QThread**) connects to /tmp/socket and continuously calls **recv()** to gather newline-terminated JSON objects containing the fields "**flag**", "**pressure**", and "**temperature**". Whenever the signals the start of a brewing cycle (**flag == 1**), each parsed JavaScript Object Notation (JSON) message triggers the **dataReceived(double, double, QString)** signal, which is connected to the **onDataReceived** slot in **graphDialog**. Inside this slot, elapsed time is computed relative to the dialog's launch, and the new sensor values are appended to the GraphWidget via **appendPressurePoint()** and appendTempPoint().
A subsequent call to refresh() repaints the plotting surface, producing an up-to-date line graph of both parameters with minimal latency, fully leveraging Qt's event-driven architecture for smooth, responsive visualization.

5.3.0.2 Data Retriaval from the InfluxDB to the GUI SC | *AK*

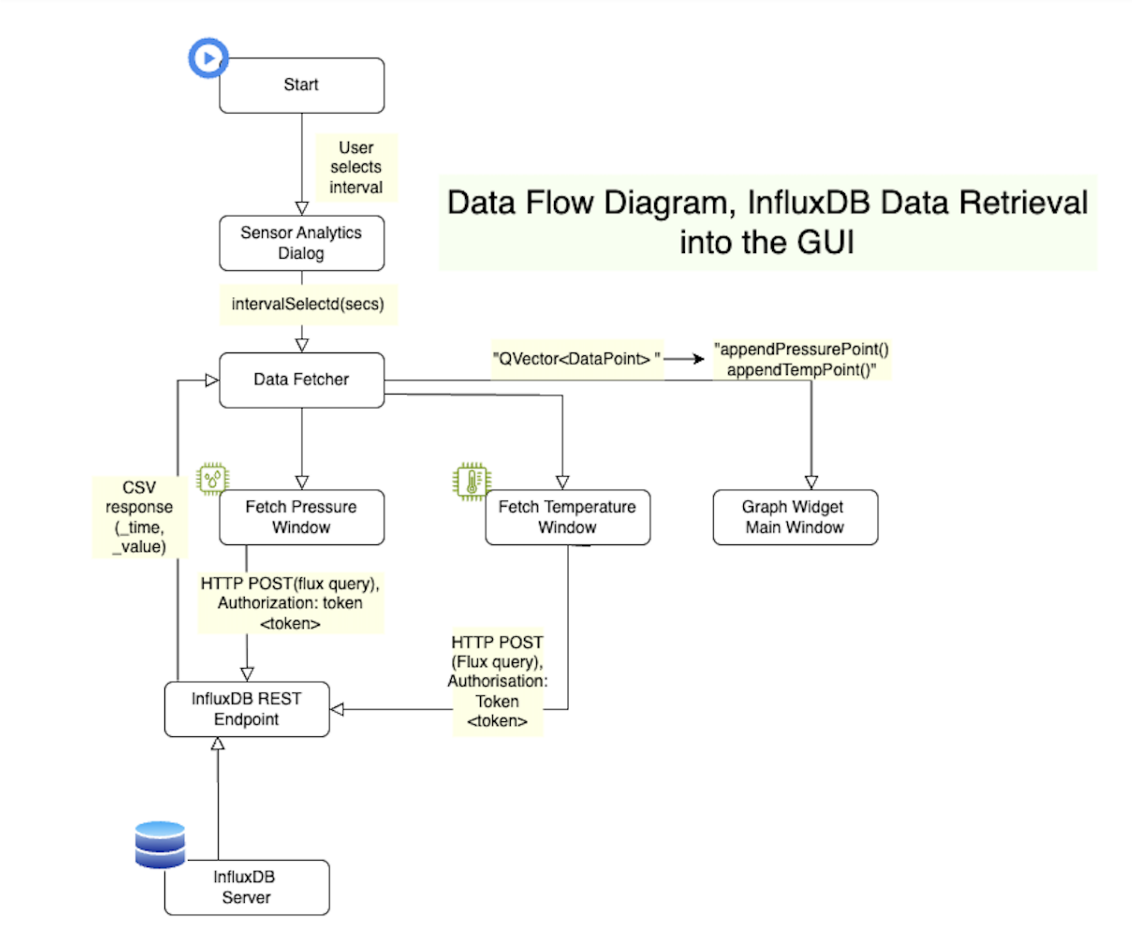


Figure 5.3.2: Data Flow Diagram of Data Retrieval from Influx database to the GUI. Designed in Drawio.

This figure 5.3.2 illustrates the flow of historical sensor data from the InfluxDB server into the graphical user interface. When the user selects a time interval in the **SensorAnalyticsDialog**, the dialog emits an intervalSelected(secs) signal to the DataFetcher component. DataFetcher then invokes two routines, **fetchPressureWindow()** and **fetchTempWindow()**, each of which issues an HTTP POST request containing a Flux query and an authorization token to the InfluxDB REST endpoint. The server responds with CSV-formatted data rows, each comprising "_time" and "_value" fields. DataFetcher parses these rows into a **QVector¡DataPoint¿** for pressure and temperature, respectively, and passes the vectors to the GraphWidget in the main window via its **appendPressurePoint()** and **appendTempPoint()** methods. This design cleanly separates user actions, data retrieval logic, and presentation, ensuring that the GUI can display historical trends efficientlly and responsively.

5.3.1 Overall Integration Of the GUI SC | *AK*

In this subsection, there will be provided a visual explanation of how the app works, each functionality and feature based on the 3.5.

5.3.1.1 Sensor Analytics SC | *MT*

To access historical sensor data within the application, the user first taps the hamburger menu icon in the upper left corner of the main window, revealing the side navigation panel.

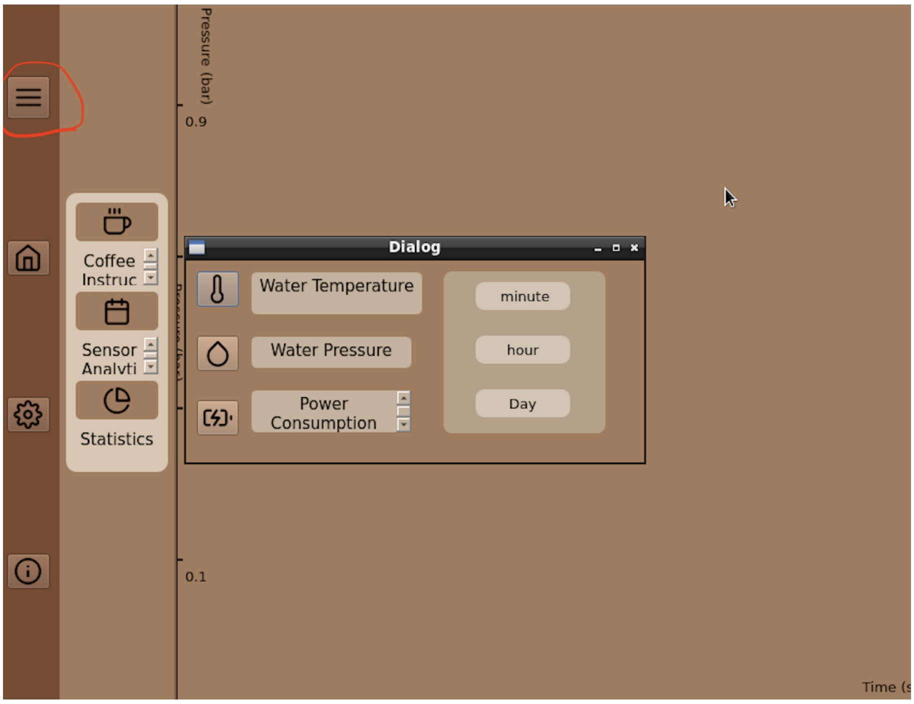


Figure 5.3.3: Visualization of steps on how to reach the sensor analytics stored values

From this panel, selecting "Sensor Analytics" opens a modal dialog that lists available parameters.

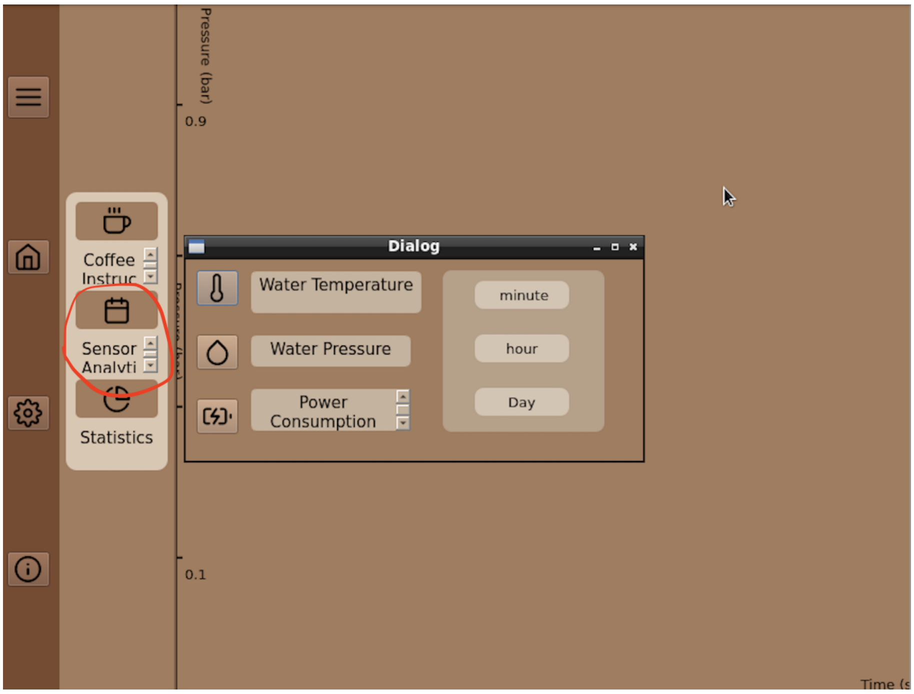


Figure 5.3.4: Visualization of steps on how to reach the sensor analytics stored values

Water Temperature, Water Pressure, and Power Consumption, each represented by a clearly labeled button.

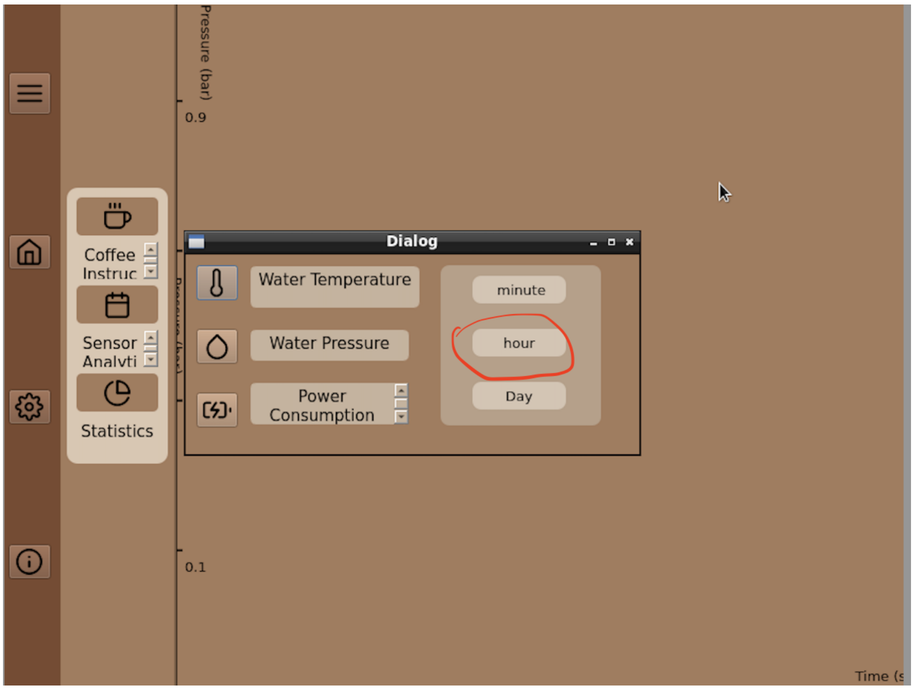


Figure 5.3.5: Visualization of steps on how to reach the sensor analytics stored values

After choosing a parameter, the user then selects one of the pre-defined time intervals (one minute, one hour, or one day) displayed on the dialog's right side. Once both parameter and interval have been specified, a dedicated QFrame overlays the main window, presenting the retrieved data as a time-series graph. This workflow ensures that users can intuitively navigate to and visualize past performance metrics without leaving the primary interface.

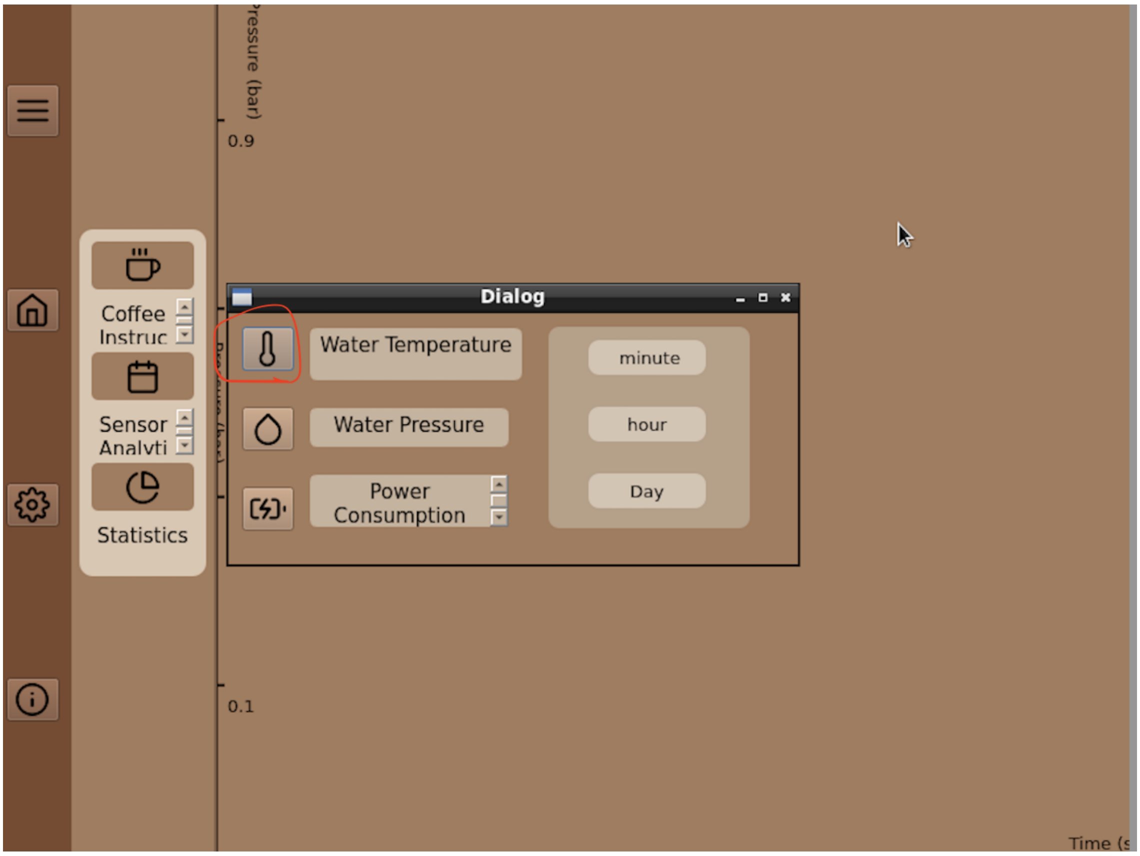


Figure 5.3.6: Visualization of steps on how to reach the sensor analytics stored values

5.3.2 Coffee Instructions **SC** | *AK*

Selecting the "Coffee Instructions" button opens a modal dialog presenting step-by-step guidance for beginner users on operating the espresso machine. This dialog contains simple text and illustrative icons to guide the user through the brewing procedure, such as coffee grinding and making sure the coffee is evenly distributed. Thereby ensuring that the users can quickly familiarize themselves with the brewing workflow without external references.

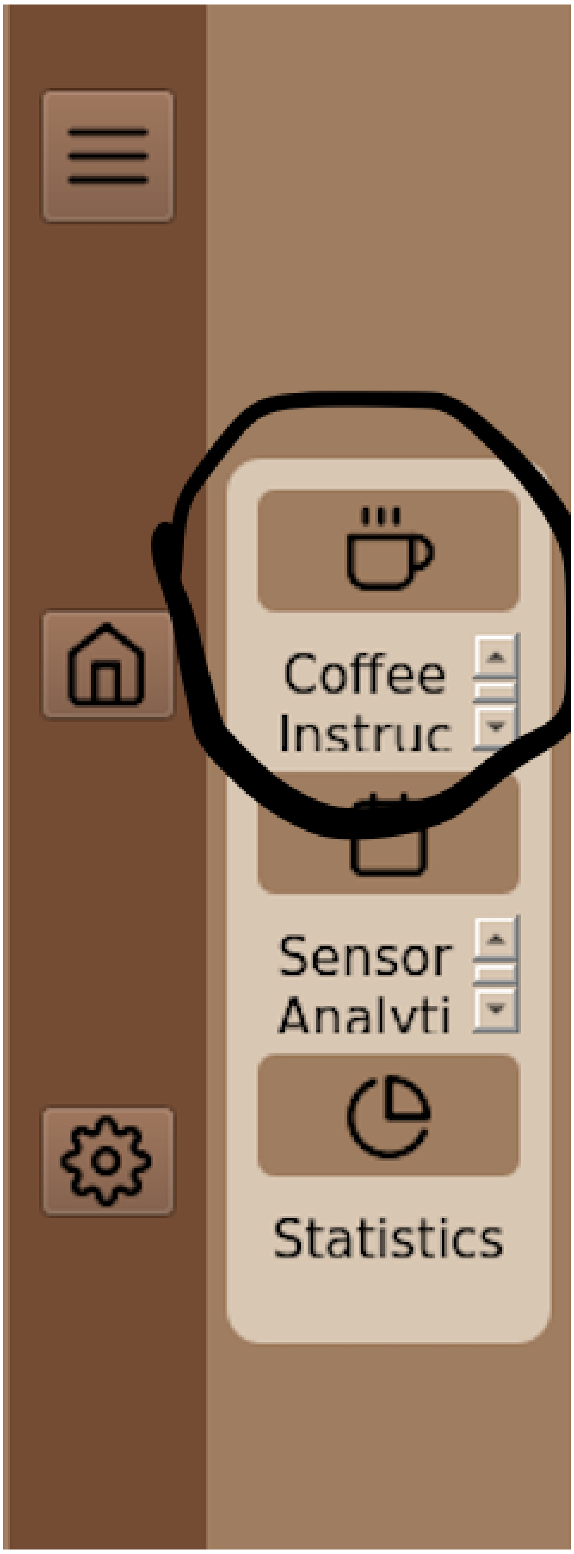


Figure 5.3.7: Visualization of steps on how to reach the coffee instructions

5.3.3 Information Button "i" SC | *AK*

Activating the "Info" button reveals an overlay QFrame containing an introductory overview of the the project and the system's functionality.

Figure 5.3.8: Visualization of steps on how to reach the coffee instructions

At the bottom of this frame, a "Here" button transitions the user to a more detailed dialog, which includes explanatory text alongside electronic schematics, sequence and activity diagrams.

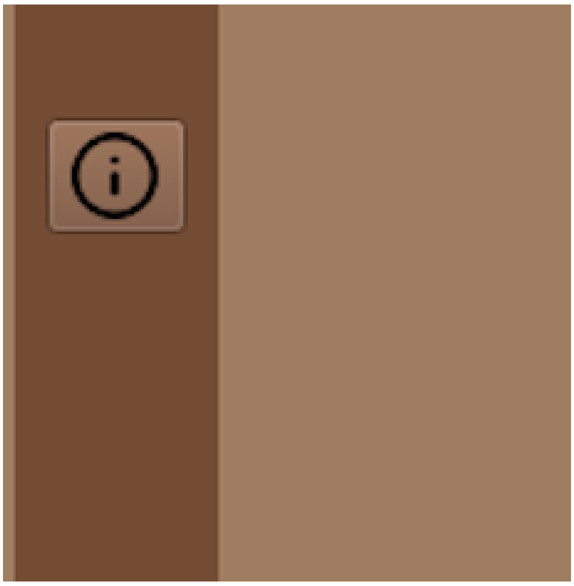


Figure 5.3.9: Visualization of steps on how to reach the coffee instructions

Navigation within this dialog is facilitated by a "Back" button on the right and a "Home" button on the left, both of which return the user either to the previous screen or directly to the main window, preserving a coherent and reversible navigation structure.

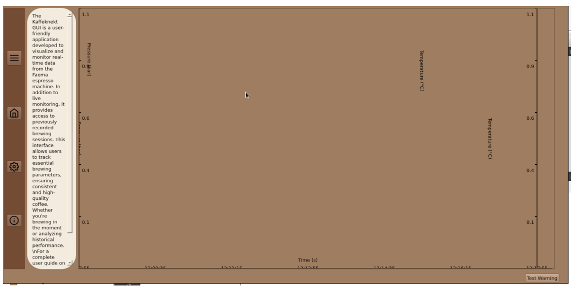


Figure 5.3.10: Visualization of steps on how to reach the coffee instructions

5.3.4 Real-Time Data Graph **SC** | *AK*

After starting the Python data-acquisition script, the user must manually launch the Qt application, which then presents a QDialog plotting live ESP32 sensor readings. When the flag value equals 1, indicating an active brewing cycle, the dialog displays water temperature in blue and pressure in red on a real-time graph. This dynamic visualization persists until the flag resets to 0 or the Python script is stopped, offering continuous feedback on the extraction process and enabling immediate parameter monitoring.

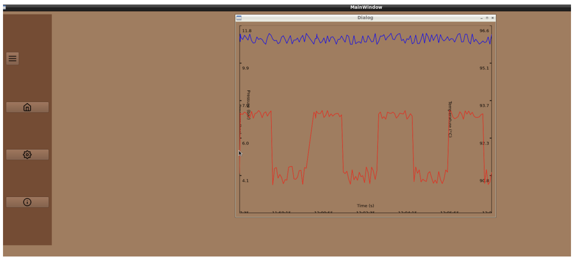


Figure 5.3.11: Visualization of steps on how to reach the coffee instructions

The initial real-time data retrieval is illustrated in figure B.8.1

5.3.4.1 Integration overview of the GUI **SC** | *AK*

The final application was assembled through a stepwise, iterative process in which each interface element was introduced and validated before moving on to the next. Initially, the MainWindow layout was created manually in the Qt Designer .ui file, establishing the overall structure and visual style. Most of the Qt code was then developed on a VM as a solution to the cross-compilation challenges6.2.1.1, allowing for consistent builds across architectures. Thereafter, individual components such as buttons, frames, and dialogs were added one at a time. After each addition, the interface was compiled and tested to verify correct placement, appearance, and behavior. Once the core functionality was stable, the remaining development and testing were conducted directly on the Raspberry Pi 5 to ensure proper interaction with both the ESP32 device and the InfluxDB server. Any issues discovered during testing were immediately addressed, and the component was refined before proceeding. This disciplined, component-by-component approach—combined with targeted VM use for cross-compilation and on-device validation, ensuring that every element worked as intended and fit into the overall design, giving a robust and well-integrated graphical application.

5.3.5 Data Aggregation and Visualization Hub

5.3.5.1 Real Time Socket Solution **DAB** | *MT*

As the project developed, we realized that our current idea which was for the Qt Desktop to continuously query the database to check if our "trigger" was active would lead to undesirable delays in the system. It would be better if the Qt Desktop could get the information without querying the database, but trying to take the data directly from the ESP32 could lead to multiple processes trying to access the same serial port.
Our solution was to expand the script that handles the data from ESP32 to the database by having it work as a socket server that sends data over an UNIX domain socket on which the Qt Desktop "listens" in. It can then use the data directly when it sees that the "trigger" has been activated.

5.3.5.2 Time Configuration **DAB** | *MT*

As the ESP32 needs to know the time and the system should also be operational without internet it was only natural that our system should be able to set the time from the touch screen. As we already had an interface that communicates with the ESP32 (i.e. a JSON format) it was natural that we would use the same format.

5.4 Minimum Viable Product

MMS | *IBM*

Along the journey, the scope of the project and the user stories which we identified early, revealed a necessity to define a realistic and achievable MVP. Its intention to reflect the core functionality of the envisioned monitoring system, with key functionalities such as data acquisition and visualization of brewing-related parameters.

During the hardware implementation phase, it was brought to our attention that the espresso machine – while central to the project concept – was not in a fully operable state. As a result, several user stories connected to real brewing scenarios and machine interaction could not be realized at this particular stage.

As a mitigation strategy, a test-bench environment was set up to simulate sensor behavior and to validate the components themselves independently of the espresso machine.

Consequently, the MVP's definition was restricted to the following functionalities as of this moment:

- "The measurement of both temperature and pressure data using selected sensors interfaced with either an MCU or RPI-5."

- "The storing of collected sensor data in some local database for analysis."

- "The visualization of data in real-time through some dashboard or screen interface for user insight."

Despite the espresso machine being inoperable under the development of an MVP, we've developed one that validates the proof of concept for a viable system architecture. The group responded in a proactive manner to the limitations met in relation to the espresso machine by mobilizing efforts around the design, testing and validation of the monitoring system through a test-bench environment.

This approach ensured that the group didn't neglect the progress and mitigated a meaningful approach to realize the project's primary objective, which is to develop insight through a monitoring system.

Chapter 6

Results and Discussion

This chapter presents the results of the project and reflects on how well the system met its intended goals. It examines whether the monitoring solution was able to deliver accurate and timely data, how the graphical interface supported user insight, and how well the system functioned as a whole during testing.

The results are discussed in relation to the initial requirements and user stories, evaluating to what extent the monitoring system delivers meaningful insight into the espresso brewing process. Challenges encountered during implementation and integration are also revisited here, with focus on how they were resolved or what limitations remained.

Finally, the chapter reflects on broader lessons learned by the team throughout the project, such as, the collaboration across disciplines, the trade-offs involved in balancing precision, usability and complexity of the system and design considerations.

6.1 Test Procedures **MMS** | *IBM*

This section outlines a presentation of our test planning, methodologies, and strategies involved with validation to verify the requirements so far realized of our monitoring system. It needs to be emphasized that this is a project of a vast interdisciplinary nature, both embedded software, mechanical and electrical subsystems intertwine in this project, this required a tailored testing approach which aligned with the specific requirements earlier defined.

The testing procedure follows a scheme where structure had to be established to ensure that each individual subsystem was performing within its required or expected boundaries before it finally would be integrated. The systems engineering principles follow us even to this stage as we have to assess at all times whether the requirements and implementation correspond to user needs, these are validated through the testing results that we strive for.

6.1.1 Electrical and Mechanical Testing **MMS** | *IBM*

Most of the electronics that we have to acquaint ourselves with are sensors, some of them are very delicate and operate on incredibly small currents. Our responsibility is to set up the environment on which the components can communicate. Eventually, a PCB will be designed with capability of expanding on further sensors. Now, to prepare ourselves for the design and layout of a PCB, it is vital to comprehend the capabilities of the sensors that we will be implementing into the system. These capabilities have been tested and validated for further integration into a hydraulic system under pressure.

6.1.1.1 Pressure **MMS** | *IBM*

Our first pressure sensor became obsolete for our system in that it could not correspond to our non-functional requirements. It was deemed that a better solution could be found and that was for an NPI-19 I2C Digital Pressure Sensor as seen in Sec. 4.4.1.
Since the communication protocol is both I2C and SPI, naturally we chose to utilize the I2C protocol since it works on the basis of finding a built-in addressing scheme which is elaborated in more depth in Sec. 4.6.0.1.

On the first test, the I2C address was not detected on the RPI-5, initially the thought was that it needed some sort of additional amplification circuitry, although that was quickly disregarded since the datasheet for the sensor is clear on that the sensor is capable of transmitting its I2C address without any difficulty whatsoever.
After a while, the sensor started smoking, the sensor was quickly disconnected from the circuit. Next, we've had to determine why it started to smoke despite that certainty that it was connected correctly.
After some troubleshooting, we've measured the voltages directly between the 3.3 V and GND GPIO pins on the RPI-5. The multimeter did not read 3.3 V, in fact it read 5 V. This was contradicting in relation to the expansion board connected to the RPI-5, it was connected the other way around so the GPIO markings on the board were reversed the other way around.
An important thing to note is the sensor's specifications, it should operate on a supply current at 3 mA. Since connecting the sensor directly to the 3.3 V would power it with slightly more than that, we've decided on connecting a resistor to current-sink the 3.3 V pin. We've measured approximately 2.7 mA when connecting a 1.2 $k\Omega$ in series with it.
The SDA and SCL pins do not require any resistors to be joined in series with them as they are already equipped with 1.8 $k\Omega$ pull-up resistors inside the RPI-5 circuit board, these sink the current adequately enough that the sensor would not be in danger of significant currents.
Finally, the sensor worked and by testing the I2C detection script on the RPI-5, we obtained the I2C address corresponding to the sensor as 28. The test was linked to establishing contact with the pressure sensor, which translates to T-1.3 from the Test Table A.4.

6.1.1.2 Solenoid Valve Coil Continuity — **MMS** | *IBM*

Late in the project, it was discovered that solenoid valves are indeed vital components in terms of brewing, but also relieving pressures, and that their behavior can be mapped to a test associated with brew-event detection, which is linked to T-1.7 from the Test Table A.4.

To actually investigate this further and verify it, firstly, a coil continuity had to be determined, this is nothing more than just finding the resistance of the coil between two terminals, in this case 1 and 2.

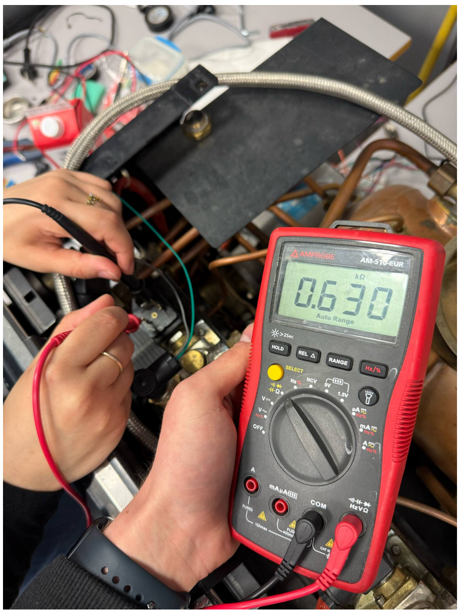


Figure 6.1.1: Measurement of the coil continuity which outputs a resistance of 630Ω, which means that the coil is intact.

[1]

[1]More information about how the solenoid valve was manipulated as a brew-event detector is explained in more depth in Appx. B.5.

6.1.1.3 ESP32's Inbuilt ADC **MMS** | *IBM*

Under one of the tests with the AC current sensor shown in Fig. 4.4.5, it was discovered that by utilizing the inbuilt ADC pins from the ESP32 to monitor the sensor, the results were not desirable, in fact, they weren't corresponding to anything of sense.

Instead, an ADS1115 ADC breakout module was used for the same test, the measurements instantly were much more realistic. ESP32's reputation behind the ADC capabilities are unfortunately poor, since an external ADC was supplied to us, it wasn't encouraged to work on a solution by making the inbuilt ADC of the ESP32 work. There are a variety of factors contributing to this inevitable limitation here are some of them:

WiFi enabled: Since the ADC2 pins can't be used when the Wi-Fi is enabled, only ADC1 instead is the only candidate for sensor use.

ADC Input range: The ESP32's ADC measures voltages which range from 0 to 3.3V and not beyond that value, like 5V.

ADC Accuracy: ADCs ideally have a linear behavior, although this is not the case for the ESP32, these are non-linear. This means that the ESP32 is incapable of distinguishing between say 3.2V and 3.3V, and the measured value will still be the 4095, or it won't distinguish between 0 and 0.13V signals as both correspond to a measured value of 0 [53].
[2]

[2] More information about the ESP32's ADC inaccuracy issues are discussed on a github issue: https://github.com/espressif/arduino-esp32/issues/92

6.1.2 Software testing and Development

6.1.2.1 GUI testing **SC** | *AK*

This section outlines the steps and components that were implemented to conduct an initial test of the graphical user interfaces.

6.1.2.2 RPI-5 **SC** | *AK*

To facilitate agile development and parallel task execution, the group utilized two separate RPI-5 devices. Following the setup of the RPI-5, which included downloading the needed packages and programs on the RPI-5. The RPI-5 was connected to a personal Windows-based laptop via Secure Shell (SSH) for remote access and control. The development of the Qt application was carried out on the Windows laptop, allowing for efficient coding and testing. Concurrently, a Python script was deployed on the RPI-5 to establish communication with the water pressure sensor, with plans to later extend this to the water temperature sensor as well. Successful sensor detection is indicated when the address "28" appears within the expected range of "20" to "40" on the screen. However, during testing, no response was detected, which was attributed to an electrical connection issue. Consequently, the group proceeded with an alternative solution, which is detailed in the next subsection.

6.1.2.3 GUI **SC** | *AK*

After all the required installations on both the MacBook, Windows laptop and the RPI-5, the tests and development could start.
The development of the GUI proceeded in several iterative stages. Initially, it started by creating a classic "Hello World" QDialog, by clicking a QPushButton triggered a QDialog displaying the text "Hello World." Thereafter, design and implementation progressed in parallel. On the design side, the visual layout and widget hierarchy were implemented in Qt Designer (.ui files), while concurrently a prototype plotting module was coded in C++. Pressing the "D" key caused a test graph to appear, displaying random values, and repressing the same key swapped to an alternate dataset, as shown in figure **??**. The subsequent phase integrated real historical data by querying the InfluxDB server via HTTP, with token and bucket parameters, to retrieve the sensor_data measurement. Initially, the returned values were displayed as raw numbers in a QDialog, and later they were displayed as a graph. Finally, real-time data acquisition from the ESP32 was added, the WebSocketClient component explained in 5.3.0.1, establishes a UNIX-domain socket connection, parses incoming JSON packets, and emits signals to append live pressure and temperature points to the on-screen graph whenever the machine enters a brewing cycle, (when the Flag == 1).

6.1.2.4 Temperature Reading and Storing tests **AK** | *SC*

The requirements state that the system shall measure the water temperature, and to satisfy them, we need an IoT temperature sensor that can seamlessly communicate with the Raspberry Pi (RPI). For this test, we have selected a sensor, DS18B20 from DFRobot, capable of providing continuous temperature readings, enabling smooth monitoring of water temperature.

6.1.2.5 The goal and plans regarding the tests **AK** | *SC*

To validate the system's functionality, we do a couple of tests to verify the accuracy and persistence of the captured temperature values. The data collected will be stored in an InfluxDB database, enabling real-time visualization through graphs and charts. In a future test, we will test the precision of the temperature sensor against a reference thermometer, which is presumed to be much more effective at its task. We then plan to calibrate the sensor to match its more equipped counterpart, thus achieving a more accurate reading.

6.1.2.6 Test environment setup **AK** | *SC*

The sensor will be connected to the RPI using 1-Wire communication protocol, which will then collect, process, and store the temperature data for further data processing. To establish the 1-Wire communication, we will first open the terminal in RPI and type this command: Sudo raspi-config. This will open the RPI's configuration menu, and from here we select: Interface → 1-wire → enable → finish.

For the connection between the sensor and the RPI, we simply connect the 3 wires from an adapter the sensor is already connected to upon arrival, avoiding extra steps with connecting to a resistor. The 3 wires are: one Data wire, Voltage(3.3V) wire, and one Ground wire, hence the name "1-Wire communication". The data wire will be plugged into the RPI's GIPO4-pin/pin 7, which is the default configured 1-Wire pin.

6.1.2.7 Test procedures **AK** | *SC*

The procedure for the tests, ranging from terminal commands to other actions, as well as python scripts are available in the list of Appendices "Temperature and Database Tests: A.9.1

6.2 Challenges **MMS** | *IBM*

Over the course of the project so far, the group encountered several challenges which slowed down the pace of development, to the extent of where some user stories couldn't be realized. The challenges that emerged in both technical, software and project management aspects, played a significant role in shaping the current state of the system.

6.2.1 Software Challenges

6.2.1.1 Cross-Compiling problem **SC** | *AK*

Cross-compilation is the process of building software on one machine, which is the host, for execution on another machine, that is considered the target, often necessary when the target has limited resources or a different CPU architecture [95]. In this project, attempts to compile the Qt-based GUI on macOS (ARM64) and Windows (x86_64) hosts for the ARM-based Raspberry Pi 5 resulted in numerous toolchain and binary-format errors, reflecting the challenges documented by developers cross-compiling for RPi on M1 Macs[79]. To overcome these issues, an Ubuntu virtual machine was deployed via VMware Fusion on macOS, providing a Linux environment compatible with the Pi's native toolchain. Within this VM, Qt Creator was configured with AArch64 GNU Compiler Collection (GCC) compilers and dedicated "Kits" for both the VM and the the RPI-5, enabling successful cross-compilation of test code such as a placeholder graph-plotting module, before deploying the same binaries directly to the RPI-5. [105]. This VM-based approach gave a consistent and reproducible build environment that resolved architectural mismatches and eliminated cross-compilation errors, thereby streamlining development of the embedded GUI.

6.2.1.2 Cmake to qmake **SC** | *AK*

The migration of the ESP32 WebSocket client from a CMake-based build into a QMake-driven Qt GUI project revealed fundamental incompatibilities between the two meta-build systems. CMake is a cross-platform tool that interprets CMakeLists.txt files to generate native build files (e.g., Makefiles, Visual Studio solutions), offering powerful commands such as find_package() and target_link_libraries() to manage dependencies and link libraries [11]. On the contrary, QMake is a Qt-specific meta-build utility that processes **.pro** and **.pri** files to produce Makefiles tailored for Qt's **moc**, **rcc**, and platform requirements [121].
Because QMake cannot parse CMake's domain-specific language, CMakeLists.txt cannot be automatically included or executed in a QMake project. Their syntaxes, variables, and build-time workflows differ fundamentally. [69].
Hence, the WebSocket client's build instructions—originally declared via CMake's add_executable() and target_link_libraries(), had to be manually re-expressed in a **.pri** file listinng **SOURCES**, **HEADERS**, **QT += core thread**, **unix: LIBS += -lpthread**, and **INCLUDEPATH** adjustments. This conversion underscores how CMake's separate configuration phase and hierarchical **CMakeLists.txt** include mechanism diverge from QMake's direct project-file processing model,

and why unifying on a single build system is often preferable in Qt-centric applications to avoid such laborious interoperability workarounds.

6.2.1.3 Minor Challenges **SC** | *AK*

Throughout the development of the GUI, ensuring a truly responsive MainWindow grid layout showed challenging due to conflicting hard-coded sizing directives. The dozens of *setFixedSize* calls and manual *addWidget* invocations prevented the left menu, center graph frame, and right panel from resizing in unison. The solution entailed refactoring the layout to a single **QGridLayout** employing *setColumnStretch* and **setRowStretch**, in combination with appropriate **QSizePolicy** settings that is expanding for the central widget and fixed for the side panels, thus allowing the center section to grow dynamically while preserving the intended widths of the flanking panels. Positioning and sizing **QLabel** static widgets (such as the logo, QR code, and "Semcon" image) repeatedly resulted in disappearing or distorted images every time the layout was adjusted. This was addressed by applying *setFixedSize*(...) to lock each label's dimensions, disabling *scaledContents* where necessary, and entrusting the layout manager, rather than manual widget repositioning, to handle placement. This approach eliminated erratic stretching and ensured consistent presentation across window resizes.

Another minor challeng was to transfer files from the VM to the Macbook. In order to secure file and folder transfer between the macOS host and Linux VM was initially impeded by the absence of an SSH server on the VM. After installing and enabling OpenSSH (sudo apt install openssh-server sudo systemctl enable –now ssh), files such as images could be pushed/sent from macOS using SCP in a single step, for example,

```
scp ~/Downloads/kaffeknekt_logo.webp sokcher@172.16.80.133:/home/sokcher/
```

This command opens an encrypted SSH session, transfers the file, and closes the connection[6], [94].

6.2.2 Technical Challenges **MMS** | *IBM*

The most significant setback in the project was the in-operability of the espresso machine during critical stages of development. A lot of our tailored user stories were directly tied to the requirement which emphasized real-time brewing conditions and monitoring. This limited the team's ability to implement, test and validate these stories as initially planned. Within our agile approach, this had a cascading effect on verification of our system, and progress tracking.

A vast portion of time was devoted to researching and comprehending the espresso machine's internal systems, these being the likes of Heat exchangers, Groupheads, and other key components. The research was necessary to accurately satisfy the requirements when the integration was being implemented into the system, for the sake of accuracy and environmental conditions. Despite this, the research extended beyond the expected scope of the group, and it delayed the technical progress. In particular, the electrical engineers couldn't establish the ideal sensor placements, as these decisions couldn't be made in isolation from physical constraints or unknown parameters in the machine.

The lack of early clarity did pose difficulties for the group to finalize choices on hardware and validation of integration strategies, although the group was required to explore domains outside of engineering disciplines, it contributed to making informed design decisions. The interdisciplinary aspect of the project made it all the more exciting to face as a challenge, as it added valuable insight and broadened the horizon for grasping the problem domain.

The group adapted to these obstacles by shifting their focus away from the espresso machine and rather focusing on the monitoring system, in the image of a test-bench.

6.2.2.1 Leakage **MMS** | *IBM*

Initially, the espresso machine provided had a leakage from the boiler water level indicator, which can be seen in disassembled in Fig. 6.2.1. The testing of the machine could not begin until a reservoir was set up, so in parallel, it was decided that the leakage should also be repaired, while deciding on a solution for a reservoir for pumping water.

Figure 6.2.1: Disassembled boiler water level indicator.

The machine was brought to our customer's workshop, where it was examined thoroughly, and after the examination, it was decided that a gasket needed to be swapped, as the former one corroded and couldn't further keep the isolation. The gasket assembly can bee seen in Fig. 6.2.2, and what was found in our machine shown in Fig. 6.2.3b

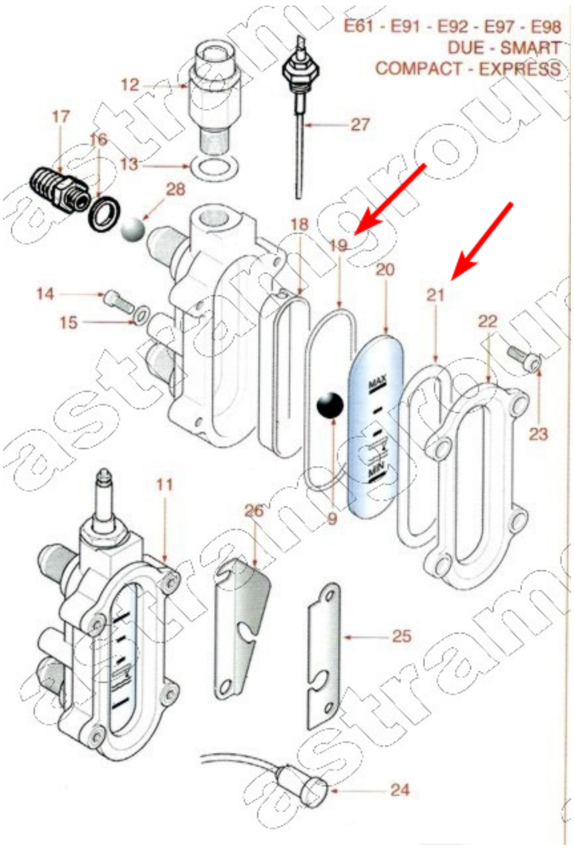


Figure 6.2.2: Parts numbered 19 and 21 (highlighted by a red arrow) are the parts that had to be swapped to new ones.

(a) Level gasket part, taken from [7].

(b) Corroded and broken level gasket.

Figure 6.2.3: Level gaskets for the boiler water level indicator.

After reassembly, the indicator was once again holding the water isolated.

6.2.2.2 Boiler overfill **MMS** | *IBM*

Under one of the tests, the boiler managed to overfill past its preset 60% threshold of the water volume inside the boiler, the boiler overfilled up to almost visible 90% on the level gauge, the reason why this is not desirable is because boilers need to have water at some preset threshold level, to make room for the layer of steam for the remaining empty space, for this espresso machine specifically, it is preset at a 60%, after that, a solenoid which is involved with providing the water to the boiler, locks itself and stops further flow. In terms of safety measures, a high water level inside the boiler would contribute to dangerous pressure building up inside the boiler under use.

Since this was the first time that this situation occurred, another examination of the internal connections was made, attempts were made to drain the boiler of water by loosening a pipe connected between the input to the boiler and the pump. This was draining too slowly, luckily, there is a drain valve underneath the boiler, we disconnected the plug and placed a funnel right underneath it inside a drain hose. The draining was too abrupt and it overfilled the funnel almost instantly, spreading the water all around our workstation, and encroaching towards the electronics.

Luckily, we reacted fast, disconnected all sensors from the machine, and moved the machine towards the floor drain in the workshop for the rest of the water to release. The aftermath of the incedent can be seen in Fig. 6.2.4.

Figure 6.2.4: The aftermath of the incident involved with draining the boiler.

The reason why the boiler overfilled must have been because of earlier tests done without the plug screwed inside the Grouphead, air being introduced inside the system, this could have fooled the electrical components, solenoids among these, and led to the water filling solenoid forgetting to lock down.

After draining the boiler, another attempt was made to fill the boiler and the solenoid stopped at approximately 60% boiler water volume, which ensured us that it wasn't damaged.

6.3 Further Work **MMS** | *IBM*

For further development, several key areas have been identified in preparation for the upcoming project milestone. The current stage has primarily been concentrated onto validating the core functionality of our MVP through a test-bench, the next stage aims to extend the system's capabilities in the actual brewing conditions.

These advancements will further serve as a framework for validating additional user stories and potentially exploring higher-level functionalities such as the assessment of the system's reliability and the possibility of expanding it with additional sensors.

6.3.1 GUI Software **SC** | *AK*

Future work will concentrate on refining the 's responsiveness and improving its overall robustness and aesthetic quality. One priority is to resolve the warning dialog issue. Currently, once twenty brewing cycles complete, the "Washing Required" Qdialog may appear repeatedly until the user dismisses each instance by clickong on "ok" button, which can disrupt workflow. Implementing a state check to ensure the warning is shown only once, this will address this bug. Additionally, optimizing layout and styling of various QFrames and dialogs, such as improving spacing, typography, and color consistency etc. This will enhance visual appeal and user experience. Continued user testing and feedback collection will guide these enhancements, ensuring that the interface not only functions reliably but also presents information in a clear and engaging manner.

6.3.2 PCB Design **IBM** | *MMS*

Some further work is needed when it comes to the PCB. This being that the PCB has been manufactured, and has arrived to us. But we do not have the components that are needed to complete the PCB. This can be read read more in Appx. B.3 Sec. B.3.5

6.3.3 Embedded improvements **MT** | *DAB*

A full restructuring of the code using classes to enable abstraction and data protection would increase the robustness of the code. At the moment any task could reach any of the data in the shared structure.
The current version of the code has a flaw. Both power sensors are incorporated in the same task (TaskPower). While this is not critical for the functionality of the system, an improvement to the system would be separating this task out into two tasks. This would be beneficial since one of the power sensors acts as the trigger that indicates when brewing starts and stops. Giving this task a higher priority would make the system detect the brewing status faster. At the moment there is a slight delay (about 0.5 second). The algorithm itself could also be optimized by taking fewer measurements. Another improvement that can be done to the current system is analyzing adjusting the timing of task.

6.3.4 Automatic Startup **DAB** | *MT*

For a more pleasant user experience, the system should run by itself at startup. To enable this a simple shell script was made that started the necessary operations (i.e. the virtual environment, the data handling script and the Qt Desktop application), but because of time restraints it could not be tested with the whole system.

6.3.5 Additional Features **MMS** | *IBM*

As the focus shifted towards milestones which were in our grasp, it was inevitable that more ambitious additions to our system would be too time-consuming and of course, less prioritized in relation to the requirements set for our project. Some of the features will be presented and argumented for their potential use in future expansions of the system.

6.3.5.1 Drip Tray Scale **MMS** | *IBM*

The addition of a drip tray scale was motivated mainly by the necessity of having a component linked with brew-event detection capabilities. A scale deemed itself a suitable candidate, although the amount of work it required to fully implement into the system was too overwhelming, given our other prioritized requirements. A drip tray scale requires a Load cell, it consists of an incredibly sensitive resistance array, in other words called a Strain gauge, the inspiration was taken from a project showcased in [38].

6.3.5.2 InfluxDB Tasks **AK** | *MMS*

As a system that is mainly to monitor, transform and visualize real-time data taken from an operated machine, an addition of several more of sophisticated Flux tasks, where every one of them carry out their unique operations. This aims for more custom data analytics availability, ensuring that the user has a better grasp of e.g. machine operation and its cost. An example of a future flux task can be one that compares current aggregated data with last week's aggregated data, which in and of itself should offer realistic suggestions to the user.

Chapter 7

Conclusion

In summary, this project started with a simple, although, challenging idea: how can a user get insight into the brewing process of an espresso machine, without taking away the craft that makes it so special? Along the way, we've dealt with technical limitations, time pressure, unexpected problems, and the interesting journey through the eyes of a barista.

The result isn't a finished product, although a working prototype that demonstrates the capabilities of real-time monitoring with cutting-edge technology from Industry 4.0 principles. The prototype is capable of collecting real-time data from key parameters, storing it and visualizing it in a way that is useful and accessible for the user. It's purpose is to be a tool meant to support the user, and not replace them, and that's been an important principle throughout the project.

More importantly, the process of building a system of this scale has taught us a lot, not only about sensors, code or even circuits, however how key is collaboration, adaptation, and learning from each other's strengths. Our achievement is the outcome of shared effort, our curiosity, and a contribution in the technologies of Industry 4.0 principles.

References

[1] Adafruit. *Adafruit MAX31865 RTD PT100 or PT1000 Amplifier*. Accessed: 2025-05-17. 2016. URL: https://learn.adafruit.com/adafruit-max31865-rtd-pt100-amplifier/.

[2] Susana Andueza et al. "Influence of water pressure on the final quality of arabica espresso coffee. Application of multivariate analysis". In: *Journal of agricultural and food chemistry* 50.25 (2002), pp. 7426–7431.

[3] Daniele Apiletti and Eliana Pastor. "Correlating espresso quality with coffee-machine parameters by means of association rule mining". In: *Electronics* 9.1 (2020), p. 100.

[4] ArduinoJson. *JsonVariant::as¡T¿() — ArduinoJson v6 API Reference*. Accessed: 2025-05-17. n.d. URL: https://arduinojson.org/v6/api/jsonvariant/as/.

[5] Arrow Electronics. *How Does a Thermosiphon Work*. https://www.arrow.com/en/research-and-events/articles/how-does-a-thermosiphon-work. Accessed: 2025-03-28.

[6] Ask Ubuntu Community. *How to scp a file from Mac (Ubuntu VirtualBox)*. Accessed: 2025-05-18. 2011. URL: https://askubuntu.com/questions/48436/how-to-scp-a-file-from-mac-ubuntu-virtualbox.

[7] Astram Group. *FAEMA FA14*. Accessed: 2025-05-18. 2025. URL: http://www.astramgroup.com/EspressoMachines/Faema/FAEMA_FA14.htm.

[8] Alan Moran (auth.) *Agile Risk Management*. 1st ed. SpringerBriefs in Computer Science. Springer International Publishing, 2014. ISBN: 978-3-319-05007-2,978-3-319-05008-9.

[9] Ronald M Baecker. *Readings in human-computer interaction : toward the year 2000*. Morgan Kaufmann Publishers, , Elsevier Inc, 1995. ISBN: 9780080515748; 0080515746; 1558602461; 9781558602465. URL: libgen.li/file.php?md5=808878175d7ea703b826d8b887dbace8.

[10] Barista Hustle. *EM 3.03 The First Thermosyphons*. https://www.baristahustle.com/lesson/em-3-03-the-first-thermosyphons/. Accessed: 2025-03-28.

[11] Jörg Bornemann. *Qt and CMake: The Past, the Present and the Future*. Accessed: 2025-05-18. 2021. URL: https://www.qt.io/blog/qt-and-cmake-the-past-the-present-and-the-future.

[12] Peter Burggräf et al. "Sensor retrofit for a coffee machine as condition monitoring and predictive maintenance use case". In: (2019).

[13] cafelifestyle. *Feama E91 A Diplomat*. Accessed: 2025-05-12. 2012. URL: https://cafelifestyle.wordpress.com/2012/01/23/feama-e91-a-diplomat/.

[14] Caffè Italia. *Single Boiler - Multi Boiler versus Heat Exchanger Coffee Machines*. Accessed: 2025-05-18. 2025. URL: https://www.coffeeitalia.co.uk/single-boiler-multi-boiler-versus-heat-exchanger-coffee-machines/.

[15] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2014.

[16] Cliff Pebble. *The Role of Temperature Stability in Espresso: How the E61 Group Head Keeps Your Shots Consistent*. Accessed: 2025-03-28. 2023. URL: https://cliffandpebble.com/blogs/our-blog/the-role-of-temperature-stability-in-espresso-how-the-e61-group-head-keeps-your-shots-consistent.

[17] Clive Coffee. *Espresso Machine Types, by Boiler*. Accessed: 2025-05-12. 2017. URL: https://clivecoffee.com/blogs/learn/espresso-machine-types-by-boiler.

[18] *Debugging in Qt*. URL: https://doc.qt.io/qtcreator/creator-debugging.html (visited on 03/26/2025).

[19] DFRobot. *Fermion: DS3232 Precise RTC (SKU: DFR0821)*. Accessed: 2025-05-17. n.d. URL: https://wiki.dfrobot.com/Fermion_DS3232_Precise_RTC_SKU_DFR0821.

[20] DFRobot. *FireBeetle 2 ESP32-E IoT Microcontroller*. https://www.dfrobot.com/product-2195.html. Accessed: 2025-03-28.

[21] DFRobot. *Gravity: Analog AC Current Sensor (SKU: SEN0211)*. Accessed: 2025-05-17. n.d. URL: https://wiki.dfrobot.com/Gravity_Analog_AC_Current_Sensor__SKU_SEN0211_.

[22] DFRobot. *Gravity: I2C ADS1115 16-Bit ADC Module (Arduino & Raspberry Pi Compatible) (SKU: DFR0553)*. Accessed: 2025-05-17. n.d. URL: https://wiki.dfrobot.com/Gravity__I2C_ADS1115_16-Bit_ADC_Module_Arduino_%26_Raspberry_Pi_Compatible__SKU__DFR0553.

[23] DFRobot. *Gravity: I2C Digital Wattmeter (SKU: SEN0291)*. Accessed: 2025-05-17. n.d. URL: https://wiki.dfrobot.com/Gravity:%20I2C%20Digital%20Wattmeter%20SKU:%20SEN0291.

[24] DFRobot. *Terminal Sensor Adapter V2 - DFR0055*. Accessed: 2025-05-17. n.d. URL: https://wiki.dfrobot.com/Terminal_sensor_adapter_V2_SKU_DFR0055.

[25] Digi-Key Electronics. *Amphenol Advanced Sensors NPI-19J-200G2 Pressure Sensor*. https://www.digikey.no/en/products/detail/amphenol-advanced-sensors/NPI-19J-200G2/15789382. Accessed: 2025-03-28.

[26] Discord. *Group chat that's all fun games*. Available: https://discord.com/. 2025.

[27] *drawio*. URL: https://app.diagrams.net/ (visited on 03/26/2025).

[28] Earthstoriez. *History and evolution of the Italian espresso machine*. https://earthstoriez.com/history-and-evolution-of-the-italian-espresso-machine. Accessed: 2025-03-28.

[29] Murray Esler. "Coffee drinking then and now: research continues to better understand this ubiquitous beverage". In: *Clinical Autonomic Research* 33 (2023). Accessed: 2025-05-19, pp. 613–615. DOI: 10.1007/s10286-023-00981-7. URL: https://link.springer.com/article/10.1007/s10286-023-00981-7.

[30] *ESP32 Wi-Fi & Bluetooth SoC | Espressif Systems*. Espressif Systems. URL: https://www.espressif.com/en/products/socs/esp32 (visited on 03/24/2025).

[31] *Espressif Announces the Launch of ESP32 Cloud on Chip and Funding by Fosun Group*. Accessed: 2025-03-24. Espressif Systems. 2016. URL: https://www.espressif.com/en/media_overview/news/20160907-esp32briefing.

[32] Espresso Parts. *Solenoid Valves for Espresso Machine: What to Know*. Accessed: 2025-05-19. 2025. URL: https://www.espressoparts.com/blogs/barista-basics-tutorials/solenoids.

[33] EspressoExperts. *Faema Due D92 Espresso Machine*. https://www.ebay.com/itm/186630072362. Seller refurbished, listed on eBay. Accessed: 2025-03-28.

[34] Maria Amélia G Ferrão et al. "Genomic-assisted breeding for climate-smart coffee". In: *The Plant Genome* 17.1 (2024), e20321.

[35] Britta Folmer. *The craft and science of coffee*. Academic Press, 2016.

[36] GeeksforGeeks. *Kernel in Operating System*. Accessed: 2025-03-28. 2025. URL: https://www.geeksforgeeks.org/kernel-in-operating-system/.

[37] Google. *Online, collaborative spreadsheets*. Available: https://workspace.google.com/products/sheets/. 2025.

[38] Jan Goolsbey. *Clue Coffee Scale*. Accessed: 2025-05-19. 2024. URL: https://www.digikey.no/en/maker/projects/clue-coffee-scale/2fa95462a8cc4fd19fbe014377b6597b.

[39] Atsushi Hashimoto et al. "Influences of pH and temperature on infrared spectroscopic features of brewed coffee". In: *Procedia Food Science* 1 (2011), pp. 1132–1138.

[40] Home-Barista.com. *Espresso Machines 101*. Accessed: 2025-05-18. 2005. URL: https://www.home-barista.com/espresso-machines/espresso-machines-101-t368.html.

[41] how.dev. *What is C++?* Accessed: 2025-03-28. URL: https://how.dev/answers/what-is-cpp.

[42] Barista Hustle. *EM 3.04 How the E61 Thermosyphon Works*. Accessed: 2025-03-28. 2025. URL: https://www.baristahustle.com/lesson/em-3-04-how-the-e61-thermosyphon-works/.

[43] InfluxData. *InfluxDB OSS v1 Documentation*. Accessed: 2025-03-28. URL: https://docs.influxdata.com/influxdb/v1/.

[44] InfluxDB. *InfluxDB bucket retention*. 2025. URL: https://docs.influxdata.com/influxdb/v2/reference/cli/influx/bucket/update//#update-the-shard-group-duration-of-a-bucket.

[45] InfluxDB. *InfluxDB storage engine*. 2025. URL: https://docs.influxdata.com/influxdb/v2/reference/internals/storage-engine/.

[46] *Introduction to Qt*. URL: https://doc.qt.io/qt-6/qt-intro.html (visited on 03/26/2025).

[47] Ippon Technologies. *A Beginner's Guide to InfluxDB: A Time-Series Database*. Accessed: 2025-03-28. URL: https://blog.ippon.tech/a-beginners-guide-to-influxdb-a-time-series-database.

[48] Seth Kenlon. *Getting started with Qt Creator*. Accessed: 2025-05-18. 2021. URL: https://opensource.com/article/21/6/qtcreator.

[49] Kjell$_C$*ompany*. *Raspberry Pi 5 Ettkortsdatamaskin Model B 4 GB*. https://www.kjell.com/no/produkter/data/raspberry-pi/raspberry-pi-5-ettkortsdatamaskin-model-b-4-gb-p88382. Accessed: 2025-03-28.

[50] Linux Mint Community. *Timeshift - System Restore Utility*. Accessed: 2025-05-16. 2024. URL: https://community.linuxmint.com/software/view/timeshift.

[51] David C. Luckham et al. "Specification and analysis of system architecture using Rapide". In: *IEEE transactions on software engineering* 21.4 (1995), pp. 336–354.

[52] Lux Cafe Club. *The Definitive Guide to Optimal Coffee Brewing Time*. Accessed: 2025-01-26. 2024. URL: https://www.luxcafeclub.com/blogs/news/the-definitive-guide-to-optimal-coffee-brewing-time.

[53] Khaled Magdy. *ESP32 ADC – Read Analog Input in Arduino IDE*. Accessed: 2025-05-20. 2023. URL: https://deepbluembedded.com/esp32-adc-tutorial-read-analog-voltage-arduino/.

[54] Wendy L Martinez. "Graphical user interfaces". In: *Wiley Interdisciplinary Reviews: Computational Statistics* 3.2 (2011), pp. 119–133.

[55] Ivan Mezei. "Cross-platform GUI for educational microcomputer designed in Qt". In: *2017 IEEE East-West Design Test Symposium (EWDTS)*. 2017, pp. 1–4. DOI: 10.1109/EWDTS.2017.8110109.

[56] Microsoft. *Get ready for the future of work with Microsoft Teams*. Available: https://www.microsoft.com/en-us/microsoft-teams/group-chat-software. 2025.

[57] Microsoft Corporation. *What is Azure DevOps?* Accessed: 2025-05-19. 2024. URL: https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops.

[58] *NovaSensor NPI-19 Digital I2C Pressure Sensor*. Tech. rep. AAS-920-699F. Accessed: 2025-05-17. Amphenol Advanced Sensors, 2022. URL: https://www.amphenol-sensors.com/hubfs/AAS-920-699F-NovaSensor-NPI-19-I2C-061322-web.pdf.

[59] Nuova Ricambi. *Faema 02*. Accessed: 2025-05-14. 2025. URL: https://nuovaricambi.net/en/draw/85.

[60] Vedat Ozan Oner. *Developing IoT Projects with ESP32: Automate Your Home or Business with Inexpensive Wi-Fi Devices*. Packt Publishing, 2021. ISBN: 978-1-83864-116-0 978-1-83864-280-8.

[61] OpenAI. *Introducing ChatGPT*. Available: https://openai.com/index/chatgpt/. 2025.

[62] Overleaf. *Why latex*. Available: https://www.overleaf.com/about/why-latex. 2025.

[63] Perfect Daily Grind. *What's the future for espresso machine technology?* 2024. URL: https://perfectdailygrind.com/2024/11/future-for-espresso-machine-technology/.

[64] Zachariah Peterson. *What Is a Solder Bridge Jumper? Best Practices in PCB Design*. URL: https://resources.altium.com/p/solder-bridge-jumper-best-practices-pcb-design.

[65] PlatformIO. *What is PlatformIO*. Accessed: 2025-05-16. 2024. URL: https://docs.platformio.org/en/latest/what-is-platformio.html.

[66] Deeksha Prahallad and Kamalraj R. "IoT Coffee Machine". In: *International Journal of Research Publication and Reviews* 5.3 (2024). Accessed: 2025-05-19, pp. 1295–1299. DOI: 10.55248/gengpi.5.0324.0658. URL: https://ijrpr.com/uploads/V5ISSUE3/IJRPR23451.pdf.

[67] *QApplication*$_c$*lass*. URL: https://doc.qt.io/qt-6/qapplication.html (visited on 03/26/2025).

[68] *QMaiWindow*. URL: https://doc.qt.io/qt-6/qmainwindow.html (visited on 03/26/2025).

[69] Qt Centre Community. *The simple form, What is the difference between cmake and qmake?* Accessed: 2025-05-18. 2014. URL: https://www.qtcentre.org/threads/60165-The-simple-form-What-is-the-difference-between-cmake-and-qmake.

[70] Qt Company. *Qt Development Tools*. Accessed: 2025-05-18. 2025. URL: https://www.qt.io/product/development-tools.

[71] *Qt Documentation*. URL: https://doc.qt.io/qt-6.7/ (visited on 03/27/2025).

[72] *Qt Features*. URL: https://www.qt.io/product/features (visited on 03/27/2025).

[73] *Qt Overview*. URL: https://doc.qt.io/qtcreator/creator-overview.html#:~:text=Qt%20Creator%20is%20a%20cross-platform%2C%20complete%20integrated%20development,interfaces%20once%20and%20deploy%20them%20to%20many%20platforms. (visited on 03/26/2025).

[74] *qWidget*. URL: https://doc.qt.io/qt-6/qwidget.html (visited on 03/26/2025).

[75] *qWidgetClass*. URL: https://doc.qt.io/qt-6/qwidget.html (visited on 03/26/2025).

[76] RASPBERRY PI. *256GB SSD for Raspberry Pi*. https://www.tme.eu/no/en/details/sc1439/raspberry-pi-acessories/raspberry-pi/256gb-ssd/. Accessed: 2025-03-28.

[77] RaspberryPi.dk. *15,6 HDMI QLED Touchscreen Display til Raspberry Pi*. https://raspberrypi.dk/no/produkt/156-hdmi-qled-touchscreen-display-til-raspberry-pi/?currency=NOK. Accessed: 2025-03-28.

[78] Red Hat. *What is Linux?* Accessed: 2025-03-28. URL: https://www.redhat.com/en/topics/linux/what-is-linux.

[79] Reddit Community. *Cross-compiling for Raspberry Pi on M1 Macs*. Accessed: 2025-05-18. 2021. URL: https://www.reddit.com/r/osdev/comments/or8kch/crosscompiling_for_raspberry_pi_on_m1_macs/.

[80] Cadance Resources. *What Is A Solder Bridge*. URL: https://resources.pcb.cadence.com/blog/2022-what-is-a-solder-bridge.

[81] Barry Richard. *Mastering the FreeRTOS Real Time Kernel: A Hands-On Tutorial Guide*. Pre-release 161204 Edition. 2016. URL: https://github.com/FreeRTOS/FreeRTOS-Kernel-Book/releases/download/V1.1.0/Mastering-the-FreeRTOS-Real-Time-Kernel.v1.1.0.pdf.

[82] *rpi5*. URL: https://www.raspberrypi.com/news/introducing-raspberry-pi-5/ (visited on 03/26/2025).

[83] RS. *What is a DIP switch*. URL: https://no.rs-online.com/web/generalDisplay.html?id=ideas-and-advice/dip-switches-guide.

[84] *RTOS Fundamentals*. Accessed: 2025-03-27. FreeRTOS. URL: https://www.freertos.org/Documentation/01-FreeRTOS-quick-start/01-Beginners-guide/01-RTOS-fundamentals.

[85] *SCA Standard 350-2021: Semi-automatic and Automatic Espresso Machines – Specifications and Test Methods*. Tech. rep. Accessed: 2025-05-17. Specialty Coffee Association, 2022. URL: https://static1.squarespace.com/static/584f6bbef5e23149e5522201/t/627a7d67ebc1285ef2453497/1652194663312/SCA_Standard_350-2021_Dev03.pdf.

[86] Ken Schwaber and Jeff Sutherland. *The Scrum Guide*. Nov. 2020. URL: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf.

[87] Scythe Studio. *What is Qt and how to create GUIs using it?* Accessed: 2025-03-28. URL: https://scythe-studio.com/en/blog/what-is-qt-framework-and-how-to-create-gui-with-it.

[88] semcon. *hoved nettsiden semcon*. Available: https://www.semcon.com/about-us. 2024.

[89] Semcon. *Semcon*. Available: https://www.semcon.com/projects/cases?utm_source. 2025.

[90] semcon. *semcon's merge with knightec*. Available: https://www.semcon.com/news/semcon-and-knightec-merge-to-form-northern-europes-leading-consulting-firm-in-product-and-digital-service-development. 2024.

[91] amphenol sensors. *NPI-19 I2C Digital Pressure Sensor Application Guide*. Accessed: 2025-05-20. URL: https://www.amphenol-sensors.com/hubfs/I2C%20NPI-19%20product%20application%20Note.pdf.

[92] *Signals and Slots in Qt*. URL: https://doc.qt.io/qt-6/signalsandslots.html (visited on 03/26/2025).

[93] Barry R. Smith. "Cross-platform GUI development". In: *IUCr Computing Commission Newsletter* 1 (2003). Accessed: 2025-05-18. URL: https://www.iucr.org/resources/commissions/computing/newsletters/1/cross-platform-gui-development.

[94] Stack Overflow Community. *Copying a local file from Mac to an SSH session in Terminal.* Accessed: 2025-05-18. 2016. URL: https://stackoverflow.com/questions/39457759/copying-a-local-file-from-mac-to-an-ssh-session-in-terminal.

[95] Stack Overflow Community. *What is cross compilation?* Accessed: 2025-05-18. 2009. URL: https://stackoverflow.com/questions/897289/what-is-cross-compilation.

[96] Jack C. Stanley and Erik D. Gross. *Project Management Handbook Simplified Agile, Scrum and DevOps for Beginners.* The Tech Academy, 2020.

[97] Andrew Stellman and Jennifer Greene. *Learning Agile: understanding Scrum, XP, Lean, and Kanban.* en. First edition. Sebastopol, CA: O'Reilly Media, 2015. ISBN: 978-1-4493-3192-4 978-1-4493-6385-7.

[98] TDK-Lambda. *TDK-Lambda Switching Power Supply, LS50-5, 5V dc, 10A, 50W, 1 Output, 125–373 V dc, 88–264 V ac Input.* Accessed: 2025-05-20. 2025. URL: https://no.rs-online.com/web/p/switching-power-supplies/6802748.

[99] The Qt Company. *Getting Started with Qt 5.* Accessed: 2025-05-18. 2025. URL: https://doc.qt.io/qt-5/gettingstarted.html.

[100] The Qt Company. *Introduction to Qt.* Accessed: 2025-05-18. 2025. URL: https://doc.qt.io/qt-6/qt-intro.html.

[101] The Qt Company. *Qt Creator Manual.* Accessed: 2025-05-18. 2025. URL: https://doc.qt.io/qtcreator/qtcreator-toc.html.

[102] The Qt Company. *Qt Creator Overview.* Accessed: 2025-05-18. 2025. URL: https://doc.qt.io/qtcreator/creator-overview.html.

[103] The Qt Company. *Qt History.* Accessed: 2025-05-18. 2025. URL: https://wiki.qt.io/Qt_History.

[104] The Qt Company. *QWebSocket Class Reference.* Accessed: 2025-05-18. 2025. URL: https://doc.qt.io/qt-6/qwebsocket.html.

[105] The Qt Company. *Raspberry Pi Quick Start Guide.* Accessed: 2025-05-18. 2025. URL: https://doc.qt.io/Boot2Qt/b2qt-qsg-raspberry.html.

[106] The Qt Company. *The Perfect Framework for Industrial Applications.* Tech. rep. Accessed: 2025-05-18. The Qt Company, 2018. URL: https://www.qt.io/hubfs/Resource%20Center%20migration/white-paper-the-perfect-framework-for-industrial-applications.pdf.

[107] TIFATECH Co., Ltd. *HTCCA Control Board for CIME, FAEMA, CASADIO Coffee Machines.* Accessed: 2025-05-15. 2024. URL: https://tifatech.vn/en/htcca-control-board.

[108] Tooplate. *Tooplate Free HTML Templates.* https://www.tooplate.com/. Accessed 2025-05-19. 2025.

[109] *touchscreen.* URL: https://raspberrypi.dk/no/produkt/156-hdmi-qled-touchscreen-display-til-raspberry-pi/ (visited on 03/26/2025).

[110] *TSP-1PAG10305MZ Resistance Thermometer.* Tech. rep. Accessed: 2025-05-19. SICK AG, 2023. URL: https://www.farnell.com/datasheets/4248426.pdf.

[111] *Understanding Split Core Current Transformer Design, Operation, Applications.* Tech. rep. Accessed: 2025-05-19. Yokogawa Test Measurement, 2025. URL: https://tmi.yokogawa.com/no/library/resources/application-notes/split-core-current-transformers/.

[112] Xibao Wang, Ge Li, and Peng Wang. "Qt-Based Cross-platform Design of Management System for Distributed Real-time Simulation Platform". In: *Proceedings of the 5th International Conference on Computer Sciences and Automation Engineering (ICCSAE 2015).* Accessed: 2025-05-18. 2016, pp. 856–860. DOI: 10.2991/iccsae-15.2016.159. URL: https://www.researchgate.net/publication/314641157_Qt-Based_Cross-platform_Design_of_Management_System_for_Distributed_Real-time_Simulation_Platform.

[113] Wikipedia contributors. *BibTeX — Wikipedia, The Free Encyclopedia.* [Online; accessed 28-March-2025]. 2025. URL: https://en.wikipedia.org/w/index.php?title=BibTeX&oldid=1274908574.

[114] Wikipedia contributors. *Bluetooth — Wikipedia, The Free Encyclopedia*. [Online; accessed 28-March-2025]. 2025. URL: https://en.wikipedia.org/w/index.php?title=Bluetooth&oldid=1282286790.

[115] Wikipedia contributors. *CERN — Wikipedia, The Free Encyclopedia*. [Online; accessed 28-March-2025]. 2025. URL: https://en.wikipedia.org/w/index.php?title=CERN&oldid=1280754673.

[116] Wikipedia contributors. *Convection — Wikipedia, The Free Encyclopedia*. [Online; accessed 28-March-2025]. 2025. URL: https://en.wikipedia.org/w/index.php?title=Convection&oldid=1269012706.

[117] Wikipedia contributors. *Espresso machine — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Espresso_machine&oldid=1278627644. [Online; accessed 1-May-2025]. 2025.

[118] Wikipedia contributors. *Google Scholar — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-March-2025]. 2025. URL: https://en.wikipedia.org/w/index.php?title=Google_Scholar&oldid=1280805544.

[119] Wikipedia contributors. *KiCad — Wikipedia, The Free Encyclopedia*. [Online; accessed 26-March-2025]. 2024. URL: https://en.wikipedia.org/w/index.php?title=KiCad&oldid=1266273887.

[120] Wikipedia contributors. *Load cell — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Load_cell&oldid=1288643076. [Online; accessed 14-May-2025]. 2025.

[121] Wikipedia contributors. *Qmake – Wikipedia, The Free Encyclopedia*. Accessed: 2025-05-18. 2025. URL: https://en.wikipedia.org/wiki/Qmake.

[122] Wikipedia contributors. *Transducer — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Transducer&oldid=1253199242. [Online; accessed 12-May-2025]. 2024.

[123] Jakub Wincenciak. *What is Qt Framework and How to Create GUI with It*. Accessed: 2025-05-18. 2025. URL: https://scythe-studio.com/en/blog/what-is-qt-framework-and-how-to-create-gui-with-it.

[124] *window and dialog widgets*. URL: https://doc.qt.io/qt-6/application-windows.html (visited on 03/26/2025).

[125] Joseph Wu. "A basic guide to I2C". In: *Application Note* (2022).

[126] MOSS Zack. "Open Source Espresso Machine for Makers". PhD thesis. University of Sheffield, 2019.

Appendix A

Project Management Documentation

Risk Analysis

Risk Analysis and Mitigation Stratagy for Project								
#	Failure:	Risk ID:	Cause Of Failure:	Effect Of Failiure:	P	C	R	Mitigation Strategy:
1	Group Member Gets Sick	RU-1.0	Health-Related Issues	Group Member Has To Catch Up On Undone Work, Or Another Member Has To Take Over Their Task	4	2	8	The Group Should Actively Clean Their Hands And Try Not To Cross-Contaminate When They Are Starting To Feel Sick Or Unwell.
2	Group Member Gets Sick Over A Long Period Of Time	RU-2.0	Health-Related Issues	The Enitre Group Has To Take Responsibilty And Take Over Their Task	3	3	9	The Group Should Actively Clean Their Hands And Try Not To Cross-Contaminate When They Are Starting To Feel Sick Or Unwell.
3	Multiple Group Members Get Sick	RU-3.0	Health-Related Issues	Group Memebers Need To Catch Up On Undone Work, While Other Team Members Take Over Their Task	3	3	9	Ensure No Single Member Is A Bottleneck By Distributing Knowledge Across The Team (Cross-Discipline Exposure) So Tasks Can Be Covered By Others. The Group Leader Should Adjust The Sprint Backlog, Adjust Deadlines If Necessary.
4	Multiple Group Members Get Sick Over A Long Period Of Time	RU-4.0	Health-Related Issues	The Rest Of The Group Need To Reevaluate The Project Timeline, And Plan Accordingly	2	4	8	Ensure At Least Two People Are Familiar With Every Critical Task, Consider Getting Temporary Help From External Advisors Or Supervisors. The Backlog Should Be Revised To Make Sure The Scrum-Method Is Implemented Correctly.
5	A Member Leaves The Group	RU-5.0	Personal And/Or Academic Reasons	Causes The Team To Reevalute The Project Timeline, And How To Split Up The Work.	1	4	4	Identify Tasks Which Were Assigned To The Departed Member, Determine Critical Skills Lost And Impact On Deadlines. Emergency Backlog Session To Reprioritize Work, If Possible, Split The Departed Member's Workload Across Multiple Members.
6	Semcon Goes Bankrupt	RU-6.0	Financial Instability	Semcon Is A Wealthy Company, Although In The Unfortunate Event, The Project Would Halt.	1	5	5	Force Majeure, Uncontrollable External Event, Negligible.
7	The Loss Of All Of Our Files	RU-7.0	Data Corruption Or Accidental Deletion	Causes The Group To Re-Do Most, If Not All Work. If Backups Are Not Made. Which In Turn, Makes It So That The Group Has The Reevalute The Timeline,	2	5	10	Utilizing Git With Cloud Repositories, Push Updates To A Remote Repository. Store CAD Files, Schematics, Reports And Documentation In Cloud Storage Such As Google Drive. Overleaf For Shared Document Collaboration. Set Up Backups For Important Files.
8	Group Members Expirencing "Burnout"	RU-8.0	High Workload And Stress Or Personal Reaons	Reduced Efficiency And Lower Productivity.	4	3	12	Ensure That No Single Team Member Is Overloaded, If One Person Is Overwhelmed, Redistribute Tasks Among Available Team Members. Use Retrospectives To Voice Concerns About Workloads And Stress Levels. The Scrum Master Should Monitor Signs Of Burnout.
9	The Espresso Machine Breaks During The Process	RU-9.0	Overuse Or Mechanical/Technical Failure	Semcon Should Be Notified, And Further Steps Will Happen From There. Most Likely Giving The Group A Maintencance Periode.	3	2	6	Advise With Semcon/Knightech When The Espresso Machine Starts To Expirence Failures. Both Before And After Said Failures.
10	A Death In The Group	RU-10.0	Health-Related / Tragic Accident	Significant Emotional And Project Disruption.	1	5	5	If A Death Is Is To Occur During This Project, Then The Group Has To Reconsider The Project Time Line, And Plan Accordingly.
11	Disagreements In The Group	RU-11.0	Differences In Opinions	Delays And Inefficiencies In Decision-Making.	4	2	8	Disagreement Concerning Technical Implementation: - Use Experiments To Test Solutions Instead Of Endless Debates - If Neither Side Can Agree, Consider A Compromise Or A Review By A Third-Party. Disagreement Concerning Workload Or Collaboration: - Retrospective To Reassign Tasks In A Fair Manner. - Balance Workload And Define Them Clearly. Disagreement Concerning Personal Differences: - Encourage Private Discussions First. - If Necessary, The Project Leader Steps In As A Neutral Mediator.

Table A.1.1: Risk Analysis Project 1

12	Group Members Not Respecting The Contract	RU-12.0	Lack Of Commitment, Communication And Interpersonal Conflicts	Causes Disruptions/Friction In The Group Dynamic And Might Cause Cascading Problems Down The Line	3	3	9	Involve The Scrum Master Or Project Leader To Check On Struggling Members Before It Escalates. Offer A Chance To Correct Their Actions, If The Behavior Persists, Discuss It In A Team Meeting. Provide The Person A Chance To Agree To A Clear Action Plan To Correct Their Behavior. As A Last Resort, Escalate To A Supervisor, Along With Evidence Of Missing Tasks Or Lacking Contribution.
13	Somebody Close To A Group Member Dies	RU-13.0	Health-Related / Tragic Accident	Delays And Inefficiencies In Decision-Making.	3	3	9	The Team As A Whole Should Look At The Project Timeline And Adjust Accordingly, And The Team Should Be There For One Another.
14	Communication Issues Within The Group	RU-14.0	Poor Responsiveness, Members Not Voicing Concerns Leading To Hidden Resentment.	Missed Deadlines, Design Flaws From Misalignment Between Hardware And Software, Increased Stress And Frustration.	3	2	6	-Communication Is A Key Factor Here. -Be Open To Feedback And Criticism. -Group Memebers Should Open Up, If They Feel That There Is Miscommunication.
15	Internet Problems	RU-15.0	Network Outage	Cloud-Dependent Features Could Get Lost/Become Unavailable	3	3	9	Backup All Data Locally.
16	We Start A Fire	RU-16.0	Electrical Short Circuit Or Overheating	Delays For Our Group And Other Groups Around Us, And The Potential Of The Room Burning Down, Severe Safety Hazard, Potential Loss Of Equipment.	2	4	8	-Follow Safety Measures -Try To Kill The Fire If That Fails: -Evacuate The Room -Sound The Fire Alarm
17	We Are Not Able To fix The Machine	RU-17.0	Lack Of Expertise Or Spare Parts	Delays In Testing And Development.	2	3	6	Prioritize The MVP (Minimal Viable Product), Determine The Core Functionalities That Must Work For The Project To Be Successful. Implement A Workaround. Reduce Scope And Adjust Sprint Goals. As A Last Resort, Escalate And Seek External Help. Document Failures.
18	Ventilation Stops Working	RU-18.0	Technical Reasons	Bad Odor, Procrastination Due To Discomfort, And Temporary Unwellness (Such As Headaches Or Nausea)	4	2	8	The Facility Manager Can Keep The Ventilation On For Longer Periods And Especially Weekends, Although This Needs To Be Communicated In Advance.
19	Hand-In USB With Virus	RU-19.0	Malware Infection	Potential Data Corruption Or Security Breach.	1	5	5	Run An Antivirus Program Such As AVG To Scan The USB Memory Stick For Hazardous Malware.
20	Tasks Are Not Finished Before Deadline	RU-20.0	Poor Time Management, Unclear Task Distribution, Unexpected Technical Issues, Or Team Member Delays	Project Delays, Lower Quality Work Due To Last-Minute Rushing, Missed Milestones, And Potential Failure To Meet Project Objectives	2	4	8	Make Sure Tasks Meet An "Approved" State Before Being Added To A Sprint. Ensure That The Scrum Master Prioritizes Tasks Efficiently In The Product Backlog. Daily Stand-Ups For Early Detection, Encourage Team Members To Raise Concerns Immediately Rather Than Waiting Until The End Of The Sprint.
21	If A Group Member Gets Divorced	RU-21.0	Personal Life Event Causing Emotional Distress And Distraction	Reduced Focus And Productivity, Potential Absence From Meetings, And Delays In Assigned Tasks	1	4	4	Contact Group Leader, And Possibly Distribute The Task From Divorced Group Memeber To Other Members To Help Out.
22	The Printers At The University Dont Work At A Critical Time	RU-22.0	No Paper/Ink Or General Use Leads To Failure	Delay For Repport Delivery, And Will Be Unprofessional On Our Part	2	4	8	Make Sure The Report Is Ready Before The Deadline, And Printed Prior To The Delivery Date.
23	Something Going Wrong When Changing The System From A Three-Phase Electrical System Over To A Singel-Phase Electrical System	RU-23.0	Electrical Failure/Short Circuit	Delays In Testing and The Possibility Of Frying Electronics And Components. Might Cause Electrical Shock.	2	3	6	-Consult With Technical Supervisor, On How To To It Safely And Correctly -Proceed With Caution -Follow Electrical Safety Guidelines -Follow General Safety Guidelines

Table A.1.2: Risk Analysis Project 2

24	Team Memeber Causes An Accident	RU-24.0	Inexperience With Tools/ Reckless Behavior	Causes Possible Delays, Depending On The Severity Of The Accident	3	2	6	Follow Safety Regulations
25	Norway Gets Involved In A War	RU-25.0	Geopolitical, political shifts, International tensions affecting national security	The Country Is At War, State Of Emergency.	2	4	8	-Uncontrollable External Event. -The Project Will Most Likely End If This Is The Case
26	Parts Dont Arrive On Time	RU-26.0	Failure On Our Part For Ordering Too Late. Distributor Has Problems. Delivery Problems	Delays For The Development Of The Project	3	4	12	Order Items On Time To Neglect This Occurance Plan Accordingly
27	Our Linux Machine Crashes	RU-27.0	Fail Instalation and Configuration	Delays For The Development Of The Project	3	4	12	Backup The Linux Server, So That A Backup Can Be Called Upon If It Crashes Have A Proper "Computer" To Run Linux For The Best Results
28	We Get Locked Out Of Our Group Room	RU-28.0	Most If Not All Members Forget Their Access Card In The Room Before Leaving The Room	Minor Delays To The Day	3	1	3	Group Members Should Keep Their Student Card On Them

Table A.1.3: Risk Analysis Project 3

Risk Analysis and Mitigation Stratagy for Product								
#	Failure	Risk ID:	Cause Of Failure	Effect Of Failiure	P	C	R	Mitigation Strategy
1	Sensor Failure	RP-1.0	Sensor Issue	Machine Provides Incorrect Data, Affecting Automation And Consistency.	2	3	6	Test The Sensors, Trace The Origin Of The Problem, If The Problem Is The Sensors Themselves Then Buy/Get New Sensors. If The Problem Is Something Else, Focus On Solving The Issue.
2	Sensor Sends Wrong Information	RP-2.0	Sensors Get Uncalibrated	Incorrect Readings Lead To Poor Espresso Quality.	2	4	8	- Establish A Routine For Sensor Calibration To Ensure Accuracy And Prevent Drift Over Time. - Use Multiple Sensors For Critical Measurements And Compare Readings To Detect Anomalies. - Implement Software-Based Validation To Identify And Filter Out Inconsistent Or Outlier Sensor Data. - Design The System To Trigger Alerts Or Switch To Default Safe Settings When Abnormal Sensor Data Is Detected.
3	Display Not Working	RP-3.0	Lost Power	Users Cannot Adjust Settings Or Monitor Machine Status.	1	2	2	- Allow Control Via A Mobile App Or External Interface. - Use Basic LED Signals To Convey Essential System Statuses. - Ensure The Display Can Be Easily Replaced Without Affecting Core Functions. - Check Power Connections And Software Updates To Prevent Failures.
4	Machine Gets Dirty	RP-4.0	Coffee Machine Does Not Get Cleaned Regualary	Affects Both Hygiene And Espresso Taste.	4	3	12	Implement Cleaning Routines.
5	Espresso Tastes Bad	RP-5.0	Incorrect Grind Size, Water Temperature Too Low Or Too High, Incorrect Brewing Time Or Pressure Issues	Poor User Experience, Affects Customer Satisfaction.	3	4	12	Display Warnings When Affected Variables Are Not Within Correct Threshold.
6	Bad Maintenence	RP-6.0	Electrical Issues, Faulty Heating Element, Clogged Water System.	Under-Extraction, Leading To Sour Espresso, Unsatisfied Customers	4	4	16	Use The Test Espresso Machine For Parts To Fix The Heating Element, Otherwise Consult Semcon/Knightech If Replacement Part And/Or Technician Are Within Budget.
7	Machine Overheats	RP-7.0	Electrical Issues, Faulty Heating Element, Clogged Water System.	May Cause Damage To Internal Components, Increasing Maintenance Costs.	3	4	12	Use The Test Espresso Machine For Parts To Fix The Problem Otherwise Consult Semcon/Knightech If Replacement Part And/Or Technician Are Within Budget.
8	Lost Connection To The Server	RP-8.0	Internet / Bluetooth Problems	Remote Monitoring And Control Functions Fail.	3	4	12	The System Will Save The Data Locally While Disconnected From The Server.
9	Loss Of A Porta filter	RP-9.0	Misplacement Or Accidental Damage	Machine Cannot Brew Espresso Properly.	3	2	6	Make Routines For Placement Of The Porta Filter And Give Intructions For Correct Use Of The Porta Filter
10	Fried Electronics	RP-10.0	Power Surge And/Or Overheating	Machine Becomes Inoperable, Requiring Costly Repairs.	2	4	8	Utilization Of Fuses Or Circuit Breakers, Regulated Power Supply, Current-Limiting Resistors.
11	Pressure Drop	RP-11.0	The Pressure In The Tank Dosent Get Regulated	Under-Extraction, Leading To A Weak Espresso.	2	4	8	Use The Test Espresso Machine For Parts To Fix The Heating Element Otherwise Consult Semcon/Knightech If Replacement Part And/Or Technician Are Within Budget
12	Information Gets Lost	RP-12.0	Software Or Server Issue	Important Data For Calibration And Monitoring Gets Lost.	2	3	6	Routinelly Backup The Data Sent To The Server.

Table A.1.4: Risk Analysis Product 1

13	Machine Breaks During Operation	RP-13.0	Mechanica/Technicall Failure	Interrupts Workflow, Leading To Downtime.	2	4	8	The Machine Should Be Checked On Regularly. The Senosrs Should Be Checked Whether Or Not If They Are Damaged Or Not. If The Sensors Are Broken Or Damaged. They Should Be Replaced. If The Espresso Machine Is Broken. Then It Has To be Sent To Maintenance.
14	User Operates the Machine Incorrectlly	RP-14.0	Human Error/Inexperience	Higher Likelyhood Of Damages And Chance For Incorrect Data To Be Collected	4	2	8	Give Instructions For Correct Use Of The Espresso Machine And Have Algorithems To Detect Statistical Outliers Within Datasets

Table A.1.5: Risk Analysis Product 2

Consequences For Product		
Level	Description	Definition
1	Minimal	Product Works As Normal
2	Minor	Product Stops Working As Intended
3	Moderate	Product Is Hindered From Functioning Properly
4	Severe	Product functions with limitations.
5	Catastrophic	Product Stops Working

Table A.1.6: Consequences For Product

Probability Level Matrix Product			
Level	Description	Definition	Probability Frequence
1	Very Low	Plausible But Rare	1 < 1000
2	Low	Happens Infrequently	1 < 500
3	Medium	Happens Sometimes	1 < 200
4	High	Happens Often	1 < 100
5	Very High	Happens Frequently	1 < 50

Table A.1.7: Probability Level Matrix For Product

Consequences For Project		
Level	**Description**	**Definition**
1	Minimal	Project Continues As Normal
2	Minor	Project Becomes Delayed
3	Moderate	Project experiences significant delays.
4	Severe	Project comes to a standstill/stops
5	Catastrophic	Projects Gets Cancelled

Table A.1.8: Consequences For Project

Probability Level Matrix Project			
Level	**Description**	**Definition**	**Probability Frequence**
1	Very Low	Plausible But Rare	1 < 200
2	Low	Happens Infrequently	1 < 100
3	Medium	Happens Sometimes	1 < 50
4	High	Happens Often	1 < 20
5	Very High	Happens Frequently	1 < 10

Table A.1.9: Probability Level Matrix For Project

Risk Matrix						
		Consequence Level				
		1 Minimal	2 Minor	3 Moderate	4 Severe	5 Catastrophic
Probability Level	5 Frequent	5 Medium	10 High	15 High	20 Very High	25 Very High
	4 Often	4 Low	8 Medium	12 High	16 High	20 Very High
	3 Sometimes	3 Low	6 Medium	9 Medium	12 High	15 High
	2 Rarely	2 Very Low	4 Low	6 Medium	8 Medium	10 High
	1 Unlikely	1 Very Low	2 Very Low	3 Low	4 Low	5 Medium

Table A.1.10: Risk Matrix

VL	Measures To Reduce Risk Is Not Required
L	Measures To Reduce Risk Can Be Required
M	Measures To Reduce Risk May Be Considered
H	Measures To Reduce Risk Should Be Considered
VH	Measure To Reduce Risk Have To Be Implemented

Table A.1.11: Risk Matrix Legend

User Stories

ID	S-1.1
Title	Instructions
User Story	**As a** user **I want** instructions on how to operate an esspresso machine. **So that** I can make a delicious and satisfying cup of espresso.
Acceptance criteria	**Given** that the user can press the "Guide" button. **Then** the user gets access the instructions screen.

ID	S-1.2
Title	Machine Usage
User Story	**As a** user **I want to** be able to receive discrete and detailed data regarding machine usage. **So that** I can monitor and understand how the machine is used over time
Acceptance criteria	**Given** that the system detects machine usage **And** updates relevant data **When** the user navigates to the statistics dashboard. **Then** the user shall be able to see statistics reflecting the machine usage over time.

ID	S-1.3
Title	Parameters
User Story	**As** a user **I want** to in real-time monitor key parameters during brewing time. **So that** I can assess and ensure the quality of the espresso.
Acceptance criteria	**Given** that the system collects real-time sensor data, **And** presents the data on the dashboard **Then** they will see: - Numerical values updated in real-time.

ID	S-1.3.1
Title	Temperature Parameter
User Story	**As** a user **I want** to in real-time monitor the water temperature during brewing time. **So that** I can assess and ensure the quality of the espresso.
Acceptance criteria	**Given** that the system collects real-time sensor data from the temperature sensor, **And** presents the data on the dashboard **Then** they will see: - Numerical values or graphs updated in real-time.

ID	S-1.3.2
Title	Pressure Parameter
User Story	**As** a user **I want** to in real-time monitor the water pressure during the espresso-making process **So that** I can assess and ensure the quality of the espresso.
Acceptance criteria	**Given** that the system collects real-time sensor data from the pressure sensor, **And** presents the data on the dashboard **Then** they will see: - Numerical values updated in real-time.

ID	S-1.3.3
Title	Timestamp Parameter
User Story	**As** a user **I want** to know the brewing time of each espresso made. **So that** I can assess and ensure the quality of the espresso.
Acceptance criteria	**Given** that the system knows when the espresso starts to brew **And** knows when the espresso is finished brewing **And** presents the data on the dashboard **Then** they will see: - Numerical values updated in real-time.

ID	S-1.4
Title	Cleaning Notifications
User Story	**As a** user **I want** to be informed of when the machine needs cleaning **So that** I can maintain the machine over time
Acceptance criteria	**Given** that the machine usage threshold has been reached since it was last cleaned **Or** approprate time has passed since last cleaning **Then** notification outputs will be showcased on the dashboard

ID	S-1.5
Title	Parameter Deviations
User Story	**As a** user **I want** to be notified when key parameters deviate from optimal conditions. **So that** I can take appropriate action to resolve the issue
Acceptance criteria	**Given** that the detected parameters fall out of bounds **Then** relevant notifications will be displayed to the dashboard

ID	S-1.6
Title	Mobility
User Story	**As a** user **I want** to be able to move the espresso machine to different locations without requiring an internet connection **So that** I can continue using the machine without losing functionality.
Acceptance criteria	**Given** that all data is stored locally. **When** we power off the machine **And** power it back on again. **Then** all saved data should still be available locally in the machine.

ID	S-1.7
Title	Coffee Report
User Story	**As a** user **I want** a report detailing the quality of espresso over a period of time **So that** I can take action to improve the overall espresso quality if needed
Acceptance criteria	**Given** that relevant data is collected, **When** the user navigates to the report screen **Then** the user will be able to view reports detailing the espresso quality over a period of time

ID	S-1.8
Title	Main Screen
User Story	**As a** user **I want** to interact with the system through an interactive-display, **So that** I can select and view relevant data.
Acceptance criteria	**Given** that the system architecture supports displaying different data types and formats, **When** the user navigates to the information selection screen, **Then** the user shall be able to select their preferred data to be shown on the main screen.

ID	S-1.9
Title	Power Usage
User Story	**As** a user **I want** to in real-time monitor the power usage of the espresso machine **So that** I can know the power consumption of the espresso machine.
Acceptance criteria	**Given** that the system collects real-time sensor data from the power sensors, **And** presents the data on the dashboard **Then** they will see: - Numerical values updated in real-time.

ID	S-1.10
Title	Graph Visualization
User Story	**As** a user **I want** sensor data visualized as graphs. **So that** I can more efficiently understand the information delivered to me.
Acceptance criteria	**Given** that the system collects sensor data, **When** real-time data is presented on the dashboard **Then** they will see: - Live graphs of temperature, pressure and/or power usage.

ID	S-1.11
Title	Scalability
User Story	**As** a user **I want** the system to support new sensors and visual elements with minimal effort **So that** it can support future needs without needing a major redesign
Acceptance criteria	**Given** that the design of the system is scalable **When** a new sensor or visual element is implemented **Then** the system's architecture shall allow its implementation with minimal change

ID	S-1.12
Title	Time Setting
User Story	**As** a user **I want** to be able to ajust the time of the system **So that** the system operates accurately according to my local time or schedule, even when it is not connected to the internet
Acceptance criteria	**Given** that the system operates without the need of an internett connection **And** the system has an "Adjust Time" functionality available on the dashboard **When** the user manually sets the system time **Then** the system updates its internal clock accordingly and continues to operate using the set time

Requirements

Requirement	Sub-requirement	Priority
R-1.1 *Use sensors to collect relevant data.*	***R-1.1.1*** *Measure the water temperature of the coffee machine.*	A
	R-1.1.2 *Measure the water pressure of the coffee machine.*	
	R-1.1.3 *Detect when a cup of coffee starts brewing and when it stops.*	
	R-1.1.4 *Measure the power used by the coffee machine.*	
	R-1.1.5 *Count the amount of coffe cups made, and record the point in time that they are made.*	
R-1.2 *Display relevant information to the user.*	***R-1.2.1*** *Display warning when water temperature is out of range (90-96°C)*	A
	R-1.2.2 *Display warning when water pressure is out of range (8-10 bar)*	
	R-1.2.3 *Display a warning when the coffee machine requires cleaning if either a week has passed or 120 cups have been brewed since the user last indicated that cleaning was performed.*	
	R-1.2.4 *Display real-time info reguarding temperature and pressure*	
	R-1.2.5 *Display amounts of coffee cups made.*	
	R-1.2.6 *Display the power usage of the coffee machine*	
R-1.3 *Save all collected data to a server.*	***R-1.3.1*** *To a local server*	A
R-1.4 *Have a user interface.*	***R-1.4.1*** *The user interacts with the system through a display.*	A
R-1.5 *Analyze collected data and make a report.*	***R-1.5.1*** *Using theoretical coffee principles to determine the coffee's quality.*	B

R-1.6 *Give instructions*	***R-1.6.1*** *On the coffee making process using premade instructions.*	A
	R-1.6.2 *On the coffee making process using users preference.*	C
	R-1.6.3 *On the development of the system.*	B
R-1.7 *Be scaleable.*	***R-1.7.1*** *The architecture of the system shall allow additional sensors to be integrated into the system without compromising real-time performance.*	B
	R-1.7.2 *The architecture of the system shall allow additional visual elements to be integrated into the system.*	
R-1.8 *Have a timesetting functionality*	***R-1.8.1*** *The system shall allow the user to manually set the system time through a user interface*	B

A.3.1 Non-invasive Requirements **MMS** | *IBM*

Early in the project, it was emphasized that one of the system-level requirements for the monitoring system was that it should not be designed such that it would be invasive to the espresso machine's internal components. In other words, no modifications would be made to the internal structure or any other pathways already existing in the espresso machine.

The constraint introduced by our customer was in the concern of preserving the integrity of the existing hardware, and to not introduce risks of damaging commercial components.

As the project progressed, it became evident that strict adherence to this requirement would significantly limit the technical feasibility of accurate sensor placements, scalable features for future work, and of course the ambitions and potential for innovation of the team.

To truly capture and evaluate accurate readings for monitoring of the parameters presented for our requirements, they had to allow for some form of invasive modifications to the system. This brings us back to the very fundamental essence of what needs to be evaluated, that is, the quality of espresso, and this requirement demands a strict measuring precision.

The non-invasive constraint associated with the requirements, hindered the team's component procurement early in the project, although the constraint was lifted by our customer as soon as it became clear at one of the meetings when we requested permission to install controlled modifications into the espresso machine, this change allowed for a broader range of technical solutions for us to explore in the development of our monitoring system.

A.3.2 Non-functional requirements **MMS** | *IBM*

Although no explicit requirements were provided by the customer concerned with the accurate monitoring of key parameters, we recognized the need to define such constraints to guide ourselves through quality, usability and of course, expectations concerned with performance of the monitoring.

Given that the customer's goal is to showcase the system as a demonstration of Industry 4.0 technologies, we prioritized precision of data and visual responsiveness of the monitoring system as crucial attributes.

The selection of components, specifically with regard to precise sensors and other components, was influenced by our self-imposed constraints. Although more cost-effective sensors were certainly considered, the decision to implement and invest in higher quality and precision components was influenced by our desire to minimize error margins, while at the same time creating a more trustworthy result of the final product.

The rationale for the selection of specific components, their specifications and interfaces are presented in Appx. B.1

Test Table

Test Case ID:	Test Type:	Test Description:	Precondition:	Test Steps:	Result:	Status:	Assigned To:	Comments:
T-FUNC-1.0	FUNC	Functionality of the ESP32	1. ESP32 properly connected to each sensor 2. All sensors individually tested 3. Connection to the Raspberry Pi 4. Program capable of reading the interface	See ESP32 Dev log	Fullfills wanted functionality.	Complete	Martin	All unit tests described in ESP32 dev log
T-FUNC-1.1	FUNC	Functionality of the Database	1. Database correctly installed on Raspberry Pi 2.An InfluxDB user set up 3. A bucket defined and configured 4. Python library packages installed in virtual environment 5. virtual environment activated	1. Activate virtual environment 2. Run "temperatur_influx.py", a script that reads data from a One-Wire temperature sensor and inputs to InfluxDB 3. Run command 'influx v1 shell', type 'use "sensor_data", 'select * from "Temperature" ' 4. Observe the screen and verify that data has been saved in a measurement	Data is being stored in a bucket	Complete	Kadir	
T-FUNC-1.2	FUNC	Functionality of the Graphical User Interface	1. Application available on the Raspberry Pi 2.			TBD		
T-FUNC-1.3	FUNC	Functionality of the Data Handling Script	1. Data Handling Scrip available on the Raspberry Pi 2. An input from a data source 3. Virtual environment activated to use library packages 4. Socket client script ready to be run	1. Successfully run, then terminate instance of a data-generating Esp32 script with platformio 2. Activate virtual environment 3. Run 'Data Handling Script' 4. Run the client script	Data shown in Data Handling Script saved in bucket, simultaneously transferred to the client script	Complete	Kadir	
T-FUNC-1.4	FUNC	Functionality of the Printed Circuit Board				TBD		
T-INT-1.0	INT	Data from the ESP32 gets stored in the database				TBD		
T-INT-1.1	INT	Data from the ESP32 triggers a graph shown on a dispaly that uses real-time data				TBD		
T-INT-1.2	INT	Data from the Database is used to show graphs shown on a dispaly				TBD		
T-INT-1.3	INT	User sets time from dispay and it sets the RTC				TBD		
T-INT-1.4	INT	Full system test				TBD		
T-ESP32-1.0	UNIT	Temperature sensor testing	See ESP32 Dev log	See ESP32 Dev log	Sensor works.	Complete	Martin	Temperature changes when we touch the probe. Cannot conclude if accurate.
T-ESP32-1.1	UNIT	DC current sensor testing	See ESP32 Dev log	See ESP32 Dev log	Sensor works.	Complete	Martin	Values are changing. Cannot conclude if accurate. Datasheet mentions calibration.
T-ESP32-1.2	UNIT	DC current sensor calibration	See ESP32 Dev log	See ESP32 Dev log	Sensor is now calibrated.	Complete	Martin	Cannot conclude if accurate. Datasheet mentions calibration.
T-ESP32-1.3	UNIT	AC current sensor testing	See ESP32 Dev log	See ESP32 Dev log	Not expected readouts.	Failed	Martin	
T-ESP32-1.4	UNIT	USB communication testing	See ESP32 Dev log	See ESP32 Dev log	Successful communication	Complete	Martin	The idea is to structure the data being sent over usb by using JSON format since it is easy to read and understand.
T-ESP32-1.5	UNIT	Combining two sensors	See ESP32 Dev log	See ESP32 Dev log	Successfully combined two tasks	Complete	Martin	Need to start planning how to structure the code. Using ai to combine the tasks is allready starting to get a bit messy. Also: The system is not connected to the internet - In this current iteration the esp32 gets the time sent from the Raspberry Pi. This is a bad idea since it can cause the time to drift and have to be revised.

T-ESP32-1.6	UNIT	Pressure sensor testing	See ESP32 Dev log	See ESP32 Dev log	Sensow works. Readouts makes sense.	Complete	Martin	I notice a sporadic i2c error once in a while.
T-ESP32-1.7	UNIT	AC current sensor 2nd test	See ESP32 Dev log	See ESP32 Dev log	Readouts doesnt makes sense.	Failed	Martin	Looks like 0.04 A means the sensor is disconnected. It is not disconnected so something is going on.
T-ESP32-1.8	UNIT	RTC-module testing	See ESP32 Dev log	See ESP32 Dev log	Setting and recieving time works.	Complete	Martin	
T-ESP32-1.9	UNIT	AC current sensor 3rd test	See ESP32 Dev log	See ESP32 Dev log	Sensor works.	Complete	Martin	These readouts makes sense, but we want to confirm this with using ampere meter later.
T-ESP32-2.0	UNIT	Calibrating AC current sensor	See ESP32 Dev log	See ESP32 Dev log	Current sensor gives accurate readouts.	Complete	Martin	
T-ESP32-2.1	UNIT	MQTT communication testing	See ESP32 Dev log	See ESP32 Dev log	Takes too much time.	Failed	Martin	Not optimal. See dev log.
T-ESP32-2.2	UNIT	Pressure sensor 2nd test	See ESP32 Dev log	See ESP32 Dev log	Sensor works.	Complete	Martin	
T-ESP32-2.3	UNIT	Bluetooth communication test	See ESP32 Dev log	See ESP32 Dev log	Established connection.	Complete	Martin	Putting this on hold until USB communication version is complete
T-ESP32-2.4	UNIT	RTC interface test	See ESP32 Dev log	See ESP32 Dev log	Date got sent and registered. ESP32 keeps sending data.	Complete	Martin	
T-ESP32-2.5	UNIT	RTC leading zeros test	See ESP32 Dev log	See ESP32 Dev log	Leading zero works now	Complete	Martin	No need to consider any more validation o fthe string since the values will be set from the gui.
T-ESP32-2.6	UNIT	Combining sensors test	See ESP32 Dev log	See ESP32 Dev log	Bug fixed.	Complete	Martin	
T-ESP32-2.7	UNIT	New temperature sensor test	See ESP32 Dev log	See ESP32 Dev log	Sensor now works.	Complete	Martin	See ESP32 Dev log, too much info
T-ESP32-2.8	UNIT	Combined test with static data	See ESP32 Dev log	See ESP32 Dev log	Successfully added new field in readouts.	Complete	Martin	
T-ESP32-2.9	UNIT	Implementation of AC current sensor	See ESP32 Dev log	See ESP32 Dev log	Unsure how to proceed.	Failed	Martin	Getting mutex errors. See ESP32 dev log
T-ESP32-3.0	UNIT	Implementation of temperature sensor as static data.	See ESP32 Dev log	See ESP32 Dev log	Static data shows up in JSON structure.	Complete	Martin	
T-ESP32-3.1	UNIT	Implementation of temperature sensor as task	See ESP32 Dev log	See ESP32 Dev log	Getting expected readouts in JSON structure	Complete	Martin	
T-ESP32-3.2	UNIT	Implementation of static pressure data	See ESP32 Dev log	See ESP32 Dev log	Static data shows up in JSON structure.	Complete	Martin	
T-ESP32-3.3	UNIT	Implementation of pressure sensor	See ESP32 Dev log	See ESP32 Dev log	Getting readouts but crashes suddenly.	Failed	Martin	
T-ESP32-3.4	UNIT	Implementation of pressure sensor take 2	See ESP32 Dev log	See ESP32 Dev log	Getting expected readouts in JSON structure	Complete	Martin	Still noticing timing issue. Multiple readings within same second, and sometimes it skips a beat. Must look into this.
T-ESP32-3.5	UNIT	Testing of brewing trigger	See ESP32 Dev log	See ESP32 Dev log	Flag = 1 when brewing, Flag = 0 when not	Complete	Martin	Look into the timing skipping a beat next!
T-ESP32-3.6	UNIT	Testing new timer functionality	See ESP32 Dev log	See ESP32 Dev log	Milliseconds works now.	Complete	Martin	
T-PCB-1.0	UNIT	Visual Inspection	1. PCB	Inspect the PCB and refer to drawings of the design	PCB looks good, and no signs of solder-bridges	Complete	Ivan	
T-PCB-1.1	UNIT	Connection between 3.3V pads	1. PCB 2. Multimeter	Connect multimeter probes, to certain pads on the PCB and double check with the schematic.	Pads that should have 3.3V are connected, and other pads that should not, are not connected	Complete	Ivan	

T-PCB-1.2	UNIT	Connection between 5v pads	1. PCB 2. Multimeter	Connect multimeter probes, to certain pads on the PCB and double check with the schematic.	Pads that should have 5V are connected, and other pads that should not, are not connected	Complete	Ivan	
T-PCB-1.3	UNIT	Connection between Battery pads	1. PCB 2. Multimeter	Connect multimeter probes, to certain pads on the PCB and double check with the schematic.	Pads related to the battery work.	Complete	Ivan	
T-PCB-1.4	UNIT	Connection between GND	1. PCB 2. Multimeter	Connect multimeter probes, to certain pads on the PCB and double check with the schematic.	Ground is Grounded	Complete	Ivan	
T-INT-1.0	UNIT	clock_time: Time is set and displayed in terminal test 1	1. Raspberry Pi	1. Start clock_time 2. Verify that the set time widget is properly shown and functions 3. Change time 4. Verify that the changed time is displayed on terminal	Changed time is displayed in the terminal and in expected format	Complete	Didrik	Was done for only HH:mm:ss, needed yyyy:MM:dd:HH:mm:ss
T-INT-1.1	UNIT	clock_time: Time is set and displayed in terminal test 2	1. Raspberry Pi	1. Start clock_time 2. Verify that the set time and date widget is properly shown and functions 3. Change time 4. Verify that the changed time is displayed on terminal	Changed time is displayed in the terminal and in expected format	Complete	Didrik	Tested for yyyy:MM:dd:HH:mm:ss
T-INT-1.2	UNIT	SocketData: Connects to Data Handling Script and reads values from random script	1. Raspberry Pi 2. ESP32	1. Activate virtual environment 2. Start Data Handling Script 3. Start SocketData 4. See that values mimic values from random script	Got errors on single lines on both sides.	Failed	Didrik & Kadir	Changed code for both sides, will retry test
T-INT-1.2	UNIT	SocketData: Connects to Data Handling Script and reads values from random script	1. Raspberry Pi 2. ESP32	1. Activate virtual environment 2. Start Data Handling Script 3. Start SocketData 4. See that values mimic values from random script	No errrors, displays correct values	Complete	Didrik & Kadir	
T-MAC-1.0	UNIT	Examine the inner circuit	1. Solenoid 2. Multimeter	Find out how the solenoid works, determine where the coil continuity is between terminals	Measurement done. 630 ohm between terminals 1 and 2.	Complete	Mikolaj	
T-MAC-1.1	UNIT	Test the solenoid	1. Solenoid 2. Multimeter	Determine if the solenoid is NC or NO. After, determine when the solenoid is turned on (230V)	The solenoid is NO (Normally open). 230V when the brewing begins.	Complete	Mikolaj	
T-MAC-1.2	UNIT	Examine the control board inside the espresso machine.	1. Multimeter	Determine where the 230V(AC) phases are connected inside the control board.	230V between every phase, verified that it is an IT-network 3-phase 230V system.	Complete	Mikolaj	
T-MAC-1.3	UNIT	Connect the 5V power supply to the espresso machine.	1. Multimeter 2. Wiring	Investigate if the power supply will output the 5V when choosing 2 phases and connecting them to L and N inputs to the Power supply.	Stable 5V output signal measured.	Complete	Mikolaj	
T-MAC-1.4	UNIT	Connect the 12V power supply to the espresso machine.	1. Multimeter 2. Wiring	Investigate if the power supply will output the 12V when choosing 2 phases and connecting them to their corresponding inputs.	Almost stable 12V (11.81V)	Complete	Mikolaj	
T-MAC-1.5	UNIT	Connect the threaded pressure sensor inside the hydraulic system of the espresso machine.	1. Threaded reduction couplers 2. Threaded adapters	Investigate where the threaded pressure sensor can be connected to measure the brewing pressure.	Threaded pressure sensor connected to the output of the left heat exchanger, connected to	Complete	Mikolaj	
T-MAC-1.6	UNIT	Connect the threaded temperature sensor to the grouphead.	1. Thread drill 2. Thread tap	Investigate where the threaded temperature sensor (RTD) can be installed inside the grouphead for most precise temperature measurements.	M20 screw unplugged and a new one designed, thread tapped and drilled by Semcon for threading compatibility with the PT100 RTD sensor.	Complete	Mikolaj	
T-MAC-1.7	UNIT	Define the control board connections, confirm wiring from the control board to the solenoids, and terminal block connections.	1. Multimeter	Determine if the wires connected in the control board are connected according to a schematic found on the website.	Terminal block connections mapped, Solenoid (Group 1), Solenoid (Group 2), Solenoid (boiler fill), identified. Button panel + flow meter, identified.	Complete	Mikolaj	
T-MAC-1.8	UNIT	Comprehend the design and operation of the espresso machine.	1. Research 2. Manual 3. Parts manual	Determine what kind of espresso machine it is, single boiler, double boiler, or heat exchanger. Get acquainted with the inner key components of the machine.	Two heat exchangers, one boiler, identified.	Complete	Mikolaj	
T-MAC-1.9	UNIT	Connect the AC current clamp sensor at the correct conductor to relate it to a "trigger" from the solenoid.	1. Multimeter 2. Examination	Determine which wire is connected to the solenoid control board.	Wire identified, clamped around.	Complete	Mikolaj	

T-GUI-1	UNIT	Qt "Hello World" QDialog build & run	Qt Creator installed; project template in place	1. Pull/checkout GUI repo. 2. Open "HelloWorld" dialog project in Qt Creator. 3. Build & Run → no compile errors. 4. Click button → "Hello World" message appears	Hello World Dialog pops up successfully	Complete	Sokaina	
T-GUI-2	UNIT	InfluxDB read/write via standalone Python	InfluxDB running on RPi2; test bucket & user created	1. Run Python script to write a sample point. 2. Query InfluxDB CLI for that point. 3. Run Python script to read it back. 4. Verify values match.	Got data from esp32 to the database	Complete	Sokaina	
T-GUI-3	UNIT	Graph module pops up on "D" key with fake data	Graph code in project; no ESP32 required	1. Load graph module in isolation. 2. Feed it a fake data array. 3. Simulate "D" key press. 4. Verify graph window opens and plots data.	Graph pop up when "D" key is pressed and changes into 3 different graphs when the same key is pressed.	Complete	Sokaina	
T-GUI-4	FUNC	MainWindow skeleton (buttons, widgets, labels)	GUI built and launched on RPi/VM	1. Launch the full GUI. 2. Verify MainWindow opens with all expected buttons, widgets & labels visible and correctly laid out.	Mainwindow opens up when code is built and run, the entire app woks as wanted.	Complete	Sokaina	
T-GUI-5	FUNC	Button & dialog end-to-end responsiveness	MainWindow open	1. Click each top-level button. 2. Verify the correct dialog/widget appears and can be closed without error.	Correct dialog oppens when buttons are clicked	Complete	Sokaina	
T-GUI-6	FUNC	"Hamburger" menu QFrame	MainWindow open	1. Click hamburger icon. 2. Verify a QFrame slides out containing three buttons: C	QFrame shows up with correct button when hamburger button is clicked	Complete	Sokaina	
T-GUI-7	FUNC	Coffee Instructions dialog	Hamburger menu open	1. Click "Coffee Instructions". 2. Verify its dialog appears with expected instructions and can be closed.	Correct dialog oppens when buttons are clicked	Complete	Sokaina	
T-GUI-8	FUNC	Sensor Analytics nested QFrame	Hamburger menu open	1. Click "Sensor Analytics". 2. Verify a nested QFrame appears with three buttons: Water Temperature, Water Pressure, Power Consumption.	Correct frame oppens when buttons are clicked	Complete	Sokaina	
T-GUI-9	FUNC	Water Temperature dialog	Sensor-Analytics QFrame open	1. Click "Water Temperature". 2. Verify temperature-readout widget/dialog opens and displays a placeholder or live value.	Correct dialog oppens when buttons are clicked	Complete	Sokaina	
T-GUI-10	FUNC	Water Pressure dialog	Sensor-Analytics QFrame open	1. Click "Water Pressure". 2. Verify pressure-readout widget/dialog opens and displ	Correct dialog oppens when buttons are clicked	Complete	Sokaina	
T-GUI-11	FUNC	Power Consumption dialog	Sensor-Analytics QFrame open	1. Click "Power Consumption". 2. Verify power-readout widget/dialog opens and display	Correct dialog oppens when buttons are clicked	Complete	Sokaina	
T-GUI-12	FUNC	Statistics (history) graph dialog	Hamburger menu open	1. Click "Statistics". 2. Verify history-graph dialog opens with date-range selectors.	Correct dialog oppens when buttons are clicked	Complete	Sokaina	
T-GUI-13	FUNC	Settings dialog	MainWindow open	1. Click "Settings". 2. Verify settings dialog appears with expected controls (DB credentials, ESP32 port, etc.).	Correct dialog oppens when buttons are clicked	Complete	Sokaina	
T-GUI-14	FUNC	Info dialog & "Here" link	MainWindow open	1. Click "Info". 2. Verify info dialog appears. 3. Click "Here" link/button. 4. Verify help text or external page loads.	Correct dialog and frame oppens when buttons are clicked	Complete	Sokaina	
T-GUI-15	FUNC	GUI ↔ ESP32 connectivity	ESP32 attached or stub server running; GUI open	1. Click "Connect ESP32". 2. Verify status indicator shows "Connected". 3. Click "Fetch Real"; confirm no errors if stub present.	Failed	Complete	Sokaina	
T-GUI-16	FUNC	GUI ↔ InfluxDB connectivity	InfluxDB running; GUI open	1. Click "Connect DB". 2. Verify status indicator shows "DB OK". 3. Click "Write Sample"; use CLI to confirm it landed in the bucket.	Correct graphs pop up	Complete	Sokaina	

T-GUI-17	UNIT	Real-time graph (fake data)	T-GUI-3 passed; GUI connected to DB & ESP32 stub	1. Press "D" in MainWindow. 2. Verify real-time graph window opens and fake data streams in.	Fake data streams and real time graph pops up successfully	Complete	Sokaina	
T-GUI-18	FUNC	Real-time graph (live ESP32 data)	ESP32 streaming live; GUI connected	1. Press "Fetch Real" or "D". 2. Verify graph window opens and plots live sensor data.	Graph plots in the correct Dialog	Complete	Sokaina	
T-GUI-19	FUNC	Historical graph from InfluxDB	InfluxDB populated with ≥30 points; GUI open	1. Open "Statistics" dialog. 2. Select date range. 3. Click "Load". 4. Verify historical graph renders and pan/zoom works.	Correct graphs pop up when clicked on spesific button (time range) in Qframe	Complete	Sokaina	
T-GUI-20	FUNC	GUI ↔ ESP32 connectivity	ESP32 attached or stub server running; GUI open	1. Click "Connect ESP32". 2. Verify status indicator shows "Connected". 3. Click "Fetch Real"; confirm no errors if stub present.	Data is fetched succesfully	Complete	Sokaina	
T-GUI-21	FUNC	Python app launches and shows main buttons	Python app code checked out; all dependencies installe	1. Run the Python GUI (python main.py). 2. Verify the main window opens with at least "Start Brewing" and "Warning" buttons visible. 3. Close the app without errors.	App runs corectly.	Complete	Sokaina	
T-GUI-22	FUNC	"Start Brewing" button triggers brew workflow	T-GUI-21 passed; (hardware stub or simulator attached)	1. Launch the app. 2. Click "Start Brewing". 3. Verify a brewing progress indicator/dialog appears. 4. Simulate (or wait for) brew completion; confirm a "Done" or similar success message is shown.	App runs corectly and displays water temp and pressure values in "real-time" with fake values	Complete	Sokaina	
T-GUI-23	FUNC	"Warning" button displays warning dialog	T-GUI-21 passed	1. Launch the app. 2. Click the "Warning" button. 3. Verify a warning dialog or pop-up appears with the correct warning text. 4. Close the warning dialog and ensure the main window remains responsive.	Warning dialog pops up as iintended with an "ok" and "ignore" button	Complete	Sokaina	
T-DTB-01	UNIT	Flux query test	1. Data available in selected bucket 2. Flux query script available	1. Save query script in a file 2. Follow syntax for executing an influx query file command in terminal 3. Observe command results	Correct data shown, compared with queried data from 'influx v1 shell'	Complete	Kadir	Flux queries drop rows that include null values
T-DTB-02	UNIT	Event logging task	1. Data within set range of task script available 2. Script written with correct syntax and logic	1. Log into InfluxDB UI 2. Save event logging script in 'Tasks* section 3. Activate the task 4. Observe sequential run statuses 5. Verify by querying data from new bucket	Given conditions in the task script, correct data has been moved to the new bucket	Complete	Kadir	
T-DTB-03	UNIT	Readable time logic	1. 'Datetime' library package included in the data handlir	See Database Dev Log	Logic that converts incoming UNIX timestamps into readable time with date and clock	Complete	Kadir	Cannot overwrite default _time column in InfluxDB
T-DTB-04	UNIT	Data handling scalability	1. An expanding JSON line 2. Virtual Environment activated 3. Available bucket	1. Run the data handling script 2. Feed JSON lines into the script 3. Add a new key-value to subsequent JSON lines 4. Verify that the script does not return any error 5. Confirm smooth data line size transition using print() 6. Query data from bucket to verify that new data has been written without any issues	Script and database can handle change in JSON line size	Complete	Kadir	Table rows before addition of new item are expectedly registered as <null>

Traceability Matrix

REQUIREMENTS TRACEABILITY MATRIX

Req # Unique ID	SubReq # Unique ID	Scope Deliverable or Feature	US or WP ID	User Story or Work Package	Assigned To	Test Case	Current Status
R-1.1	R-1.1.1	Real-Time Monitoring	S-1.3.1	User Story: Temperature Parameter	Martin	T-ESP32-1.0	Complete
R-1.1	R-1.1.4	Real-Time Monitoring	S-1.10	User Story: Power Usage	Martin	T-ESP32-1.1	Complete
R-1.1	R-1.1.4	Real-Time Monitoring	S-1.10	User Story: Power Usage	Martin	T-ESP32-1.2	Complete
R-1.1	R-1.1.4	Real-Time Monitoring	S-1.10	User Story: Power Usage	Martin	T-ESP32-1.3	Failed
R-1.3	R.1.3.1	Real-Time Monitoring	S-1.3	User Story: Parameters	Martin	T-ESP32-1.4	Complete
R-1.1	N/A	Real-Time Monitoring	S-1.3	User Story: Parameters	Martin	T-ESP32-1.5	Complete
R-1.1	R-1.1.2	Real-Time Monitoring	S-1.3.2	User Story: Pressure Parameter	Martin	T-ESP32-1.6	Complete
R-1.1	R-1.1.4	Real-Time Monitoring	S-1.10	User Story: Power Usage	Martin	T-ESP32-1.7	Failed
R-1.8	R-1.8.1	Real-Time Monitoring	S-1.12	User Story: Time Setting	Martin	T-ESP32-1.8	Complete
R-1.1	R-1.1.4	Real-Time Monitoring	S-1.10	User Story: Power Usage	Martin	T-ESP32-1.9	Complete
R-1.1	R-1.1.4	Real-Time Monitoring	S-1.10	User Story: Power Usage	Martin	T-ESP32-2.0	Complete
R-1.8	R-1.8.1	Real-Time Monitoring	S-1.12	User Story: Time Setting	Martin	T-ESP32-2.1	Failed
R-1.1	R-1.1.2	Real-Time Monitoring	S-1.3.2	User Story: Pressure Parameter	Martin	T-ESP32-2.2	Complete
R-1.8	R-1.8.1	Real-Time Monitoring	S-1.12	User Story: Time Setting	Martin	T-ESP32-2.3	Complete
R-1.8	R-1.8.1	Real-Time Monitoring	S-1.12	User Story: Time Setting	Martin	T-ESP32-2.4	Complete
R-1.8	R-1.8.1	Real-Time Monitoring	S-1.12	User Story: Time Setting	Martin	T-ESP32-2.5	Complete
R-1.1	N/A	Real-Time Monitoring	S-1.3	User Story: Parameters	Martin	T-ESP32-2.6	Complete
R-1.1	R-1.1.1	Real-Time Monitoring	S-1.3.1	User Story: Temperature Parameter	Martin	T-ESP32-2.7	Complete
R-1.1	N/A	Real-Time Monitoring	S-1.3	User Story: Parameters	Martin	T-ESP32-2.8	Complete
R-1.1	R-1.1.4	Real-Time Monitoring	S-1.10	User Story: Power Usage	Martin	T-ESP32-2.9	Failed
R-1.1	R-1.1.1	Real-Time Monitoring	S-1.3.1	User Story: Temperature Parameter	Martin	T-ESP32-3.0	Complete
R-1.1	R-1.1.1	Real-Time Monitoring	S-1.3.1	User Story: Temperature Parameter	Martin	T-ESP32-3.1	Complete

R-1.1	R-1.1.1	Real-Time Monitoring	S-1.3.1	User Story: Temperature Parameter	Martin	T-ESP32-3.2	Complete
R-1.1	R-1.1.1	Real-Time Monitoring	S-1.3.1	User Story: Temperature Parameter	Martin	T-ESP32-3.3	Failed
R-1.1	R-1.1.1	Real-Time Monitoring	S-1.3.1	User Story: Temperature Parameter	Martin	T-ESP32-3.4	Complete
R-1.1	R-1.1.5	Real-Time Monitoring	S-1.2	User Story: Machine Usage	Martin	T-ESP32-3.5	Complete
R-1.8	R-1.8.1	Real-Time Monitoring	S-1.12	User Story: Time Setting	Martin	T-ESP32-3.6	Complete
R-1.7	R-1.7.1	Real-Time Monitoring	S-1.11	User Story: Scalability	Ivan	T-PCB-1.0	Complete
R-1.7	R-1.7.1	Real-Time Monitoring	S-1.11	User Story: Scalability	Ivan	T-PCB-1.1	Complete
R-1.7	R-1.7.1	Real-Time Monitoring	S-1.11	User Story: Scalability	Ivan	T-PCB-1.2	Complete
R-1.7	R-1.7.1	Real-Time Monitoring	S-1.11	User Story: Scalability	Ivan	T-PCB-1.3	Complete
R-1.7	R-1.7.1	Real-Time Monitoring	S-1.11	User Story: Scalability	Ivan	T-PCB-1.4	Complete
							Not started
R-1.2	R-1.2.1	UI Development	S-1.5	User Story: Parameter Deviations	Sokaina	T-GUI-5	Complete
R-1.2	R-1.2.2	UI Development	S-1.5	User Story: Parameter Deviations	Sokaina	T-GUI-10	Complete
R-1.2	R-1.2.3	UI Development	S-1.4	User Story: Cleaning Notifications	Sokaina	T-GUI-5	Complete
R-1.2	R-1.2.4	UI Development	S-1.3; S-1.3.1; D-1.3.2; S-1.10	User Story: Parameters User Story: Temperature Parameter User Story: Graph Visualization	Sokaina	T-GUI-17, T-GUI-18, T-GUI-11, T-GUI-10, T-GUI-9	Complete
R-1.2	R-1.2.5	UI Development	S-1.2	User Story: Machine Usage	Sokaina	T-GUI-12	Complete
R-1.2	R-1.2.6	UI Development	S-1.9	User Story: Power Usage	Sokaina	T-GUI-11	Complete
R-1.4	R-1.4.1	UI Development	S-1.8	User Story: Main Screen	Sokaina	T-GUI-4	Complete
R-1.5	R-1.5.1	UI Development	S-1.7	User Story: Coffee Report	Sokaina	T-GUI-7	Complete
R-1.6	R-1.6.1	UI Development	S-1.1	User Story: Instructions	Sokaina	T-GUI-14	Complete
R-1.6	R-1.6.2	UI Development	S-1.1	User Story: Instructions	Sokaina	T-GUI-14	Failed
R-1.6	R-1.6.3	UI Development	S-1.1	User Story: Instructions	Sokaina	T-GUI-14	Complete
R-1.1	R-1.1.3	Data Handling Developement	S-1.3.3	User Story: Timestamp Parameter	Abdulkadir	T-DTB-02	Complete
R-1.1	R-1.1.3	Data Handling Developement	S-1.3.3	User Story: Timestamp Parameter	Abdulkadir	T-DTB-03	Complete
R-1.7	R-1.7.1	Database Developement	S-1.11	User Story: Scalability	Abdulkadir	T-DTB-04	Complete

R-1.3	R-1.3.1	Database Developement	S-1.6	User Story: Mobility	Abdulkadir	T-FUNC-1.1	Complete
R-1.8	R-1.8.1	UI Development	S-1.12	User Story: Time Setting	Didrik	T-INT-1.0	Complete
R-1.8	R-1.8.1	UI Development	S-1.12	User Story: Time Setting	Didrik	T-INT-1.1	Complete
R-1.2	R-1.2.4	UI Development	S-1.3	User Story: Parameters	Didrik	T-INT-1.2	Failed
R-1.2	R-1.2.4	UI Development	S-1.3	User Story: Parameters	Didrik	T-INT-1.3	Complete
R-1.1	R-1.1.1	Real-Time Monitoring	S-1.3.1	User Story: Temperature Parameter	Mikolaj	T-MAC-1.6	Complete
R-1.1	R-1.1.2	Real-Time Monitoring	S-1.3.2	User Story: Pressure Parameter	Mikolaj	T-MAC-1.5	Complete
R-1.1	R-1.1.3	Real-Time Monitoring	S-1.3.3	User Story: Timestamp Parameter	Mikolaj	T-MAC-1.0, T-MAC-1.1, T-MAC-1.7	Complete
R-1.1	R-1.1.4	Real-Time Monitoring	S-1.9	User Story: Power Usage	Mikolaj	T-MAC-1.9	Complete
R-1.1	R-1.1.5	Real-Time Monitoring	S-1.2	User Story: Machine Usage	Mikolaj	T-MAC-1.3, T-MAC-1.4	Complete
R-1.7	R-1.7.1	Real-Time Monitoring	S-1.11	User Story: Scalability	Mikolaj	T-MAC-1.8	Complete

Code Documentation Of The Water Pressure Graph

The goal of this document: explanation of the code (I did not use doxygen this time).

This Qt-based application is designed to visually display predefined espresso machine pressure data in the form of a line graph.It is a well-structured example of a Qt GUI program designed for data visualization.

The primary goal is to simulate how water pressure readings can be presented in a real-time dashboard environment. The user interface is built using the Qt 6 framework, employing object-oriented programming in C++. Core components include the **QApplication** and **QMainWindow** classes to define the application structure, and a custom **GraphView** class derived from **QGraphicsView**, which handles the visual display of the graph. The graph updates when the user presses the "D" key.

1. *The GraphView class:*

This class is defined in **graphview.h** and implemented in **graphview.cpp**, is a subclass of **QGraphicsView**. It serves as a custom widget that manages and displays pressure graphs using a **QGraphicsScene**. In the constructor, a new graphics scene is created and attached to the view using setScene(). The window and drawing area are fixed to a size of 600x500 pixels to maintain layout consistency.

The class overrides the keyPressEvent() method to handle keyboard input. When the user presses the **'D'** key, the method triggers generatePressureGraph(), which is responsible for rendering the graph. This function clears any existing graphics from the scene and selects a dataset from three predefined pressure data sets. These datasets represent different pressure scenarios, such as irregular readings caused by blockages or consistent values indicating optimal machine performance. The data is mapped to screen coordinates using **QPointF**, and lines are drawn between each data point using **scene->addLine()**. Additionally, the graph includes labeled X and Y axes, where the X-axis shows time labels from 07:00 to 17:00(which represents when the employees mostly consume/uses the espresso machine, this is just an assumption), and the Y-axis displays pressure values ranging from 3 to 12 bar(where a good espresso coffee ranges between 8 and 10 bars)

2. *MainWindow Class:*

The **MainWindow** class is defined in **mainwindow.h** and implemented in **mainwindow.cpp**. It inherits from **QMainWindow**, a Qt class specifically designed to represent the main window in a GUI application. Within the constructor, the UI elements generated by **mainwindow.ui** are initialized using **ui->setupUi(this)**. The central widget of the main window is then set to an instance of the **GraphView** class using setCentralWidget(). This embeds the custom graph view directly into the application's main window interface, making it the focal point of the user interface. The destructor of the class ensures that resources are properly released by deleting the ui pointer. This helps prevent memory leaks and maintains clean application behavior.

3. *Application Entry Point:*

The application’s entry point is defined in **main.cpp**, which follows the standard structure of a Qt GUI program. The function begins by creating a **QApplication** object, passing in the command-line arguments to initialize the application environment. This object is responsible for managing the GUI application's lifecycle and event loop. A **MainWindow** instance is then created, representing the primary window of the application. Before displaying it, the window title is set to “**Trykkgraf – Espresso**” using setWindowTitle(), which gives context to the interface. The show() method is called to render the main window on the screen. Finally, the app.exec() call starts Qt’s main event loop, allowing the application to remain responsive to user input such as keyboard events and window interactions. This setup ensures that the GUI remains active and functional until the user closes the application.

4. *Project Configuration with CMake:*

The build configuration is handled using CMakeLists.txt, which specifies the project requirements and build instructions. The file sets a minimum required CMake version of **3.19** and specifies that the project uses C++ with **Qt 6.5**. Required Qt components include the Core and Widgets modules. The executable, named TestGraf, includes source and header files such as main.cpp, mainwindow.cpp, graphview.cpp, and their corresponding headers and UI files.

The CMake script also links the application to necessary Qt libraries e.g. (**Qt::Core** and **Qt::Widgets**) and sets up installation paths using standard variables. Deployment scripts are generated with **qt_generate_deploy_app_script()**, which helps automate distribution, particularly on platforms like Windows or Linux(for RPI).

Group Contribution

Project Management	
Responsible: Martin Taraldstad	
Task	**Performed By**
Research On The Agile Method	Martin
Foundational Support For Scrum	Martin
Processing And Documentation Of The Projectmodel	Martin
Management of Azure DevOps	Martin

Requirements Formulation	
Responsible: Didrik Aas Bergan	
Task	**Performed By**
First Draft Of User Stories	Kadir
Current User Stories	Didrik, Martin, Sokaina

Requirements	Didrik, Martin, Sokaina
Tests	Didrik, Martin, Sokaina
Documentation Of Requirements	Didrik

Risk Management	
Responsible: Ivan Bergmann Maronsson	
Task	**Performed By**
Research, Matrices, And Templates	Ivan
First Draft Of The Risk Analysis	Ivan
Risk Analyses For Product And Project	Ivan w/Group
Mitigation Strategies	Group
Documentation	Ivan, Mikolaj
Revaluation Of Risk	Ivan

Updating Risks	Ivan

Software Architecture	
Responsible: Martin Taraldstad/Didrik Aas Bergan	
Task	**Performed By**
System Diagrams	Martin, Sokaina
GUI Diagrams	Sokaina
Brainstorm & Planning	Didrik, Sokaina
Overall Context Diagrams	Didrik, Kadir, Sokaina, Martin

System Architecture	
Responsible: Sokaina Cherkane	
Task	**Performed By**

Brainstorming, Collaboration And Planning	**Group**
Design Of System Architecture	**Sokaina, Martin**

Tables	
Responsible: Group	
Task	**Performed By**
Budget	**Ivan, Mikolaj**
Components	**Mikolaj, Ivan**
Requirements	**Didrik w/Group**
Test	**Didrik, Martin**
Traceability	**Martin, Didrik**
User Stories	**Didrik, Kadir**
Risk Management	**Ivan**

Administrative Documents	
Responsible: Sokaina Cherkane	
Task	**Performed By**
Group Contract	**Group**
Templates	**Group**
Time Lists	**Martin w/ Group**
Date Planner	**Sokaina**
Meeting Referral	**Didrik, Sokaina w/Group**
Meeting Agenda	**Sokaina**

Components And Technical Reasearch	
Responsible: Mikolaj Szczelewski	

Task	Performed By
Espresso Machine	Mikolaj, Ivan
Raspberry Pi 5	Martin
Microcontroller/ESP32	Martin, Mikolaj
Sensors	Mikolaj, Martin, Ivan

Data Base	
Responsible: Abdulkadir Kabuk	
Task	**Performed By**
Enviroment Setup	Kadir, Sokaina
Sensor Testing without ESP32	Kadir, Didrik
Sensor Testing with ESP32	Kadir, Martin
Database Architecture	Kadir

Sequence Diagram	Kadir
Data Logging	Kadir
Socketing	Didrik, Kadir

Espresso Machine	
Responsible: Mikolaj Szczelewski	
Task	**Performed By**
Disassembly	Mikolaj, Ivan
Documentation	Mikolaj
Part Schematics	Mikolaj
Inspection	Mikolaj
Observational Testing	Martin, Sokaina, Didrik, Mikolaj

PCB	
Responsible: Ivan Bergmann Maronsson	
Task	**Performed By**
Design	Ivan
Footprints & Symbols	Mikolaj, Ivan
Schematic	Ivan, Mikolaj
Layout	Ivan
Datasheets & Reaserch	Mikolaj, Ivan
DRC	Mikolaj, Ivan
Ordering	Ivan
Testing without compoents	Ivan

Social Media

Responsible: Martin Taraldstad	
Task	**Performed By**
Instagram	**Martin**
Bilder	**Martin W/group**

Marketing	
Responsible: Sokaina Cherkane	
Task	**Performed By**
Website	**Sokaina**
Merch	**Mikolaj, Sokaina**
Promotional Material	**Mikolaj, Sokaina**
Posters	**Sokaina**

Software	
Responsible: Martin Taraldstad	
Task	**Performed By**
GIT Repository Setup	Martin
Raspberry Pi 5 Setup	Martin
InfluxDB NR 1 Setup	Kadir
InfluxDB NR 2 Setup	Sokaina
QT Creator Program	Sokaina
QT Creator Kit Setup On Two Raspberry PI5's	Sokaina
ESP32/FreeRTOS setup	Martin

Testing
Responsible: Martin Taraldstad

Task	Performed By
Sensor Testing	Martin, Mikolaj
Microcontroller Testing	Martin
Integration Testing	Martin w/Group
PCB Testing	Ivan
Interface Testing	Martin w/Group

Main Report	
Responsible: Ivan Bergmann Maronsson	
Task	**Performed By**
Basic LaTeX Setup	Mikolaj, Ivan
First Draft Report Structure	Mikolaj
Overall Report Structure	Ivan, Mikolaj, Sokaina

LaTeX Support	Ivan, Mikolaj
Error Decoding & Problem Solving	Ivan, Mikolaj
Figures, Tables, And Appendix Support	Ivan, Mikolaj
Doxgen/Sphinx Responsible	Didrik, Sokaina

Final product	
Responsible: Group	
Task	**Performed By**
Espresso Machine	Mikolaj
Sensors	Mikolaj, Martin, Ivan
DataBase	AbdulKadir
GUI	Sokaina, Didrik
PCB	Ivan, Mikolaj

ESP32	**Martin**
System Architecture	**Didrik**
Version Control	**Martin**

Budget & Components List

Component:	Model:	Description:	Supplier:	URL:	Price in NOK:	Quantity:	Cost in NOK:	Did We Get Covered	Status:
				Electronics					
Analog AC Current Sensor	SEN0211	For measuring the power consumption of the espresso machine.	Farnell	https://no.farnell.com/dfro	220.67 kr	1	220.67 kr	Semcon	Delivered
I2C Digital Wattmeter, I2C/UART 4-Pin Sensor Wire	SEN0291	For measuring the power consumption of the monitoring system.	Farnell	https://no.farnell.com/dfro	167.40 kr	2	334.80 kr	Semcon	Delivered
ANALOG WATER PRESS SENSOR, ARDUINO BRD	SEN0257	Grunnet at vi skal måle vanntrykk, blir det nødvendig med en vanntrykksensor, denne skal måle trykket i varmeveksleren.	Farnell	https://no.farnell.com/dfro	195.37 kr	1	195.37 kr	Semcon	Delivered
Precise RTC Module	DFR0821	A real-time clock module, with the capability of storing progammable time-of -day alarms, "event reminders" and (notifications / warnings)	Farnell	https://no.farnell.com/dfro	166.23 kr	1	166.23 kr	Semcon	Delivered
I2C ADS1115 16-Bit ADC Module	DFR0553	Analog-to-digital-converter, ideal for precise sensor measurements, such as thermocouples, RTDs or NTC and PTC thermistors.	Farnell	https://no.farnell.com/dfro	170.41 kr	1	170.41 kr	Semcon	Delivered
Ettkortsdatamaskin 2,4GHz 4 Kjerne 8GB RAM Broadcom BCM2712 Arm Cortex-A76	SC1112	For drifting av skjerm, gode tilkoblingsmuligheter, USB/I2C/Bluetooth/Wi-Fi/UART	Digikey	https://www.digikey.no/en/	909.60 kr	1	909.60 kr	Semcon	Delivered
K-type Temperatursensor -73°C – 482°C Ledningsterminal	240-080	Måling av temperatur, dataen blir lest fra signalomformeren (DFR0553)	Digikey	https://www.digikey.no/no/	113.59 kr	1	113.59 kr	Semcon	Delivered
M.2 HAT (tilkoblingsenhet for m.2 SSD)	2648-SC1166-ND	Expansion board for Rpi 5, SSD-compatible	Digikey	https://www.digikey.no/no/	136.44 kr	1	136.44 kr	Semcon	Delivered
256GB SSD til raspberry pi (SC1439)	2648-SC1439-ND	SSD storage	Digikey	https://www.digikey.no/no/	341.10 kr	1	341.10 kr	Semcon	Delivered
Kjølevifte for raspberry pi 5(SC1148)	2648-SC1148-ND	Cooling fan for the raspberry pi	Digikey	https://www.digikey.no/en/	56.85 kr	1	56.85 kr	Semcon	Delivered
Spesialisert Festeplater For bruk med Termoelement-spisser	5425-XE-3599-001-ND	Festeplate til temperatursensoren	Digikey	https://www.digikey.no/no/	75.16 kr	1	75.16 kr	Semcon	Delivered
Pt100 Temperatursensor -50°C – 250°C 2-ledersystem Eksponerte ledningstråder	5425-XE-5586-001-ND		Digikey	https://d.digikey.com/dc/n	386.01 kr	1	386.01 kr	KaffeKnekt	Delivered
MAX31865 Motstand-til-digital-omformer Grensesnitt Evalueringskort	1528-1804-ND	RTD-to-digital converter amplifier, includes the amplification and conversion of the signal from the RTD.	Digikey	https://www.digikey.no/no/	169.98 kr	1	169.98 kr	KaffeKnekt	Delivered
Termiske plater Hvit 1,14m x 27,00mm Rektangulær Lim - på begge sider	1168-2060-ND		Digikey	https://www.digikey.no/no/	69.02 kr	1	69.02 kr	KaffeKnekt	Delivered
Analog AC Current Sensor	SEN0211		DFRobot	https://www.dfrobot.com/pr	222.17 kr	1	222.17 kr	KaffeKnekt	Delivered
Gravity: Water Pressure Sensor	SEN0257		DFRobot	https://www.dfrobot.com/pr	177.52 kr	1	177.52 kr	KaffeKnekt	Delivered
Precise RTC	DFR0821		DFRobot	https://www.dfrobot.com/pr	133.97 kr	2	267.94 kr	KaffeKnekt	Delivered
Digital Wattmeter	SEN0291		DFRobot	https://www.dfrobot.com/pr	77.03 kr	3	231.09 kr	KaffeKnekt	Delivered
ADC Programmable Gain Amplifier	DFR0316		DFRobot	https://www.dfrobot.com/pr	133.97 kr	1	133.97 kr	KaffeKnekt	Delivered
Analog Steam Sensor	SEN0121		DFRobot	https://www.dfrobot.com/pr	22.22 kr	4	88.88 kr	KaffeKnekt	Delivered
ACS712 Analog Hall Current Sensor	SEN0214		DFRobot	https://www.dfrobot.com/pr	99.36 kr	1	99.36 kr	KaffeKnekt	Delivered
Waterproof DS18B20Temperature Sensor Kit	KIT0021		DFRobot	https://www.dfrobot.com/pr	83.73 kr	1	83.73 kr	KaffeKnekt	Delivered
Digital Capacitive Touch Sensor	DFR0030		DFRobot	https://www.dfrobot.com/pr	39.08 kr	1	39.08 kr	KaffeKnekt	Delivered
I2S MEMS Microphone(Breakout)	SEN0526		DFRobot	https://www.dfrobot.com/pr	54.71 kr	2	109.42 kr	KaffeKnekt	Delivered
FireBeetle 2 ESP32-E(N16R2)	DFR1139		DFRobot	https://www.dfrobot.com/pr	144.02 kr	2	288.04 kr	KaffeKnekt	Delivered
Electrolytic Capacitor Pack 100PCS	FIT0117		DFRobot	https://www.dfrobot.com/pr	43.54 kr	1	43.54 kr	KaffeKnekt	Delivered
M3*6Nylon Screws	FIT0065		DFRobot	https://www.dfrobot.com/pr	15.63 kr	1	15.63 kr	KaffeKnekt	Delivered
M3*20 Hex Standoff Mounting Kit	FIT0183		DFRobot	https://www.dfrobot.com/pr	32.38 kr	1	32.38 kr	KaffeKnekt	Delivered
M3*30 Nylin Standoff	FIT0194		DFRobot	https://www.dfrobot.com/pr	22.33 kr	1	22.33 kr	KaffeKnekt	Delivered
I2S Microphone Module	SEN0327		DFRobot	https://www.dfrobot.com/pr	39.08 kr	2	78.16 kr	KaffeKnekt	Delivered
PCB Engineering Ruler	DWG0014M		DFRobot	https://www.dfrobot.com/pr	39.08 kr	2	78.16 kr	KaffeKnekt	Delivered

Boks Plast, ABS Svart Deksel inkludert, lokk 7,700" L x 4,300" B (195,58mm x 109,22mm)	DKS-PB40-87		Digikey	https://www.digikey.no/no/p	216.37 kr	1	216.37 kr	Semcon	Delivered
SENSOR 200PSIG 1/8"NPT 14BIT	NPI-19J-200G2		Digikey	https://www.digikey.no/no/p	880,49kr	1	880,49kr	Semcon	Delivered
AC/DC CONVERTER 5V 50W	LS50-5		Digikey	https://www.digikey.no/no/p	271,35 kr	1	271,35 kr	Semcon	Delivered
TEST LEAD 18AWG BLACK/RED	FIT0585		Digikey	https://www.digikey.no/no/p	34,73 kr	2	69,46 kr	Semcon	Delivered
RPI 5 DISPLAY CABLE 200MM	SC1131		Digikey	https://www.digikey.no/no/p	11.13 kr	2	22.26 kr	Semcon	Delivered
27W USB-C PSU EU, BLACK	SC1408		Digikey	https://www.digikey.no/no/p	133.56kr	1	133.56kr	Semcon	Delivered
STACKING HEADER 40PIN	2223		Digikey	https://www.digikey.no/no/p	27,83 kr	3	83.49kr	Semcon	Delivered
Maestrini Rørvinkel Messing, Inv/Utv 1/4"	116843		Watski	https://www.watski.no/mae	69 kr	1	69.00 kr	KaffeKnekt	Delivered
Maestrini Reduksjonsnippel utv. 1/4" inv. 1/8" Mes	116886		Watski	https://www.watski.no/mae	39 kr	1	39.00 kr	KaffeKnekt	Delivered
PCB			JLC PCB	https://jlcpcb.com/	1,280.84 kr	1	1,280.84 kr	Ivan	Delivered
230V 3-fase skjøtekabel			Biltema	https://www.biltema.no/byg	1998kr	2	1998kr	KaffeKnekt	Delivered
RES 10K OHM 1% 1/4W 1206			Digikey	https://www.digikey.no/en/	5.02kr	27	5.02kr	Semcon	In transit
RES SMD 300 OHM 1% 3/4W 1206			Digikey	https://www.digikey.no/en/	18.77kr	12	18.77kr	Semcon	In transit
SWITCH SLIDE SPDT 100MA 6V			Digikey	https://www.digikey.no/en/	126.81kr	10	126.81kr	Semcon	In transit
IC REG LINEAR 3.3V 250MA SOT23-5			Digikey	https://www.digikey.no/en/	23.30kr	5	23.30kr	Semcon	In transit
PH2.0 CONNECTOR KIT 2.0MM 0.079"			Digikey	https://www.digikey.no/en/	92.20kr	1	92.20kr	Semcon	In transit
FIREBEETLE ESP32-E IOT MICROCONT			Digikey	https://www.digikey.no/en/	204.72kr	2	204.72kr	Semcon	In transit
CAP CER 1UF 100V X7R 1206			Digikey	https://www.digikey.no/en/	16.22kr	18	16.22kr	Semcon	In transit
CAP CER 10UF 25V X7R 1206			Digikey	https://www.digikey.no/en/	5.20kr	5	5.20kr	Semcon	In transit
CAP CER 0.1UF 50V X7R 1206			Digikey	https://www.digikey.no/en/	10.20kr	12	10.20kr	Semcon	In transit
BATTERY HOLDER COIN 12MM SMD			Digikey	https://www.digikey.no/en/	50.67kr	3	50.67kr	Semcon	In transit
ADAFRUIT NAU7802 24-BIT ADC - ST			Digikey	https://www.digikey.no/en/	123.28kr	2	123.28kr	Semcon	In transit
STRAIN GAUGE LOAD CELL - 4 WIRES			Digikey	https://www.digikey.no/en/	81.84kr	2	81.84kr	Semcon	In transit
IC ADC 16BIT SIGMA-DELTA 10VSSOP			Digikey	https://www.digikey.no/en/	181.52kr	4	181.52kr	Semcon	In transit
DIODE ZENER 3.6V 200MW SOD323F			Digikey	https://www.digikey.no/en/	4.35kr	3	4.35kr	Semcon	In transit
TERM BLK 2POS SIDE ENT 3.5MM PCB			Digikey	https://www.digikey.no/en/	24.54kr	3	24.54kr	Semcon	In transit
EVAL BOARD FOR MAX31865			Digikey	https://www.digikey.no/en/	464.64kr	3	464.64kr	Semcon	In transit
CONN SOCKET 18POS 0.1 GOLD PCB			Digikey	https://www.digikey.no/en/	68.70kr	3	68.70kr	Semcon	In transit
CONN SOCKET 14POS 0.1 GOLD PCB			Digikey	https://www.digikey.no/en/	54.69kr	3	54.69kr	Semcon	In transit
CONN SOCKET 8POS 0.1 GOLD PCB			Digikey	https://www.digikey.no/en/	26.73kr	3	26.73kr	Semcon	In transit
15,6" HDMI QLED Touchscreen Display til Raspbe	Touchscreen		RaspberryPI.dk	https://raspberrypi.dk/no/pr	2,610.00 kr	1	2,610.00 kr	Semcon	Delivered
JST PH 2mm 3-pin Plug-Plug Cable - 100mm long			Kiwi Electronics	https://www.kiwi-electronics	69,37kr	8	69,37kr	Mikolaj	Delivered
STEMMA Cable - 150mm Long 4 Pin JST-PH Cable–Female/Female			Kiwi Electronics	https://www.kiwi-electronics	69,37kr	8	69,37kr	Mikolaj	Delivered
						Sum Of Electronics:	9,355.47 kr		
			Miscellaneous						
Latex Subscription	N/A	We Use Latex to write our bachelor	Overleaf	https://www.overleaf.com/	650.00 kr	1	650.00 kr	Semcon	Delivered
Merch	Skjorter med LOGO		Kai Hansen AS	https://www.kai-hansen.no	828.00 kr	6	4,968.00 kr	Ivan	Delivered
Cleaning Supplies				Varries	300.00 kr	1	300.00 kr	KaffeKnekt	Delivered
						Sum Of Miscellaneous:	5,918.00 kr		

Temperature and Database Tests

```
kaffeknekt@raspberrypi:~ $ cd /sys/bus/w1/devices
kaffeknekt@raspberrypi:/sys/bus/w1/devices $ ls
28-0000104848fa  w1_bus_master1
kaffeknekt@raspberrypi:/sys/bus/w1/devices $ cd 28-0000104848fa
kaffeknekt@raspberrypi:/sys/bus/w1/devices/28-0000104848fa $ cat w1_slave
8e 01 4b 46 7f ff 02 10 02 : crc=02 YES
8e 01 4b 46 7f ff 02 10 02 t=24875
kaffeknekt@raspberrypi:/sys/bus/w1/devices/28-0000104848fa $
```

Table A.9.1: Initial temperature reading

```
kaffeknekt@raspberrypi:/sys/bus/w1/devices/28-0000104848fa $ cd ~
kaffeknekt@raspberrypi:~ $ cd sensorFiles/
kaffeknekt@raspberrypi:~/sensorFiles $ python3 temp.py
24.87500°C
24.87500°C
24.93700°C
24.93700°C
24.93700°C
```

Table A.9.2: Temperature reading with script, code in A.3.2

24.93700°C
25.31200°C
27.87500°C
30.31200°C
32.12500°C
33.50000°C
34.62500°C
35.43700°C
36.18700°C
36.81200°C
37.31200°C
37.81200°C
38.25000°C
38.62500°C
38.87500°C
39.18700°C
39.37500°C
39.62500°C
39.87500°C
40.12500°C
40.37500°C
40.50000°C
40.62500°C
40.68700°C
40.75000°C
40.81200°C
40.87500°C
40.93700°C

Table A.9.3: Temperature in warm water

41.06200°C
40.50000°C
37.43700°C
34.43700°C
32.06200°C
30.25000°C
28.75000°C
27.50000°C
26.50000°C
25.68700°C
25.00000°C
24.43700°C
23.93700°C
23.50000°C
23.06200°C
22.75000°C
22.43700°C
22.25000°C
22.00000°C
21.81200°C
21.68700°C
21.50000°C
21.37500°C
21.25000°C
21.12500°C
21.00000°C

Table A.9.4: Temperature from warm to cold water. We can observe the rapid temperature drop at the upper lines.

```
kaffeknekt@raspberrypi:~/sensorFiles $ source ~/influx_env/bin/activate
(influx_env) kaffeknekt@raspberrypi:~/sensorFiles $ python3 tempdb.py
21.62500°C
25.37500°C
31.37500°C
35.31200°C
37.31200°C
38.31200°C
38.25000°C
30.50000°C
26.50000°C
24.18700°C
22.87500°C
22.00000°C
21.50000°C
21.18700°C
```

Table A.9.5: Temperature data forwarded to InfluxDB database

```
(influx_env) kaffeknekt@raspberrypi:~/sensorFiles $ deactivate
kaffeknekt@raspberrypi:~/sensorFiles $ cd ..
kaffeknekt@raspberrypi:~ $ influx v1 shell
InfluxQL Shell dev
Connected to InfluxDB OSS v2.7.11
> use sensor_data
> select * from Temperature
                            "Temperature"  Measurement on "sens
```

Table A.9.6: InfluxDB table queries

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

index	time	temperature_C
1	1743069833224135168.0000000000	23.1870000000
2	1743069837224883968.0000000000	22.8750000000
3	1743069841227430656.0000000000	22.6250000000
4	1743069845230047744.0000000000	22.5000000000
5	1743069849232790016.0000000000	22.4370000000
6	1743069852235082496.0000000000	22.3120000000
7	1743069856237664512.0000000000	22.2500000000
8	1743069860240463360.0000000000	28.3750000000
9	1743069864242950656.0000000000	32.7500000000
10	1743069868244733952.0000000000	35.3120000000
11	1743069871247364864.0000000000	37.0000000000
12	1743069875250139648.0000000000	37.2500000000
13	1743069879254324480.0000000000	29.2500000000
14	1743069883258472192.0000000000	24.7500000000
15	1743069887261277184.0000000000	22.2500000000
16	1743069890263255040.0000000000	20.7500000000
17	1743069894265134592.0000000000	19.8750000000
18	1743069898268024320.0000000000	19.3120000000
19	1743069902270792960.0000000000	18.9370000000
20	1743072303935568128.0000000000	21.6250000000
21	1743072307937202688.0000000000	25.3750000000
22	1743072311940108800.0000000000	31.3750000000
23	1743072315942806528.0000000000	35.3120000000
24	1743072319945595136.0000000000	37.3120000000
25	1743072322947031040.0000000000	38.3120000000
26	1743072326949283840.0000000000	38.2500000000
27	1743072330951520256.0000000000	30.5000000000
28	1743072334953889792.0000000000	26.5000000000
29	1743072338956208128.0000000000	24.1870000000

3 Columns, 33 Rows, Page 1/2
Table 1/1, Statement 1/1

Table A.9.7: InfluxDB table page 1 (the first half of values are from a previous test)

index	time	temperature_C
30	1743072341958638592.0000000000	22.8750000000
31	1743072345961539840.0000000000	22.0000000000
32	1743072349964156672.0000000000	21.5000000000
33	1743072353968909824.0000000000	21.1870000000

3 Columns, 33 Rows, Page 2/2
Table 1/1, Statement 1/1

Table A.9.8: InfluxDB table page 2

The following script covers the step-by-step procedure in setting up a line of communication for the raspberry pi to import real-time data from the DS18B20 temperature sensor. To accomplish this, we use terminal commands, raspberri pi configuration operations, as well as creating an InfluxDB user in the browser. Comments for every "action" will be available, highlighted in italic format and an indent in their respective line for easier distinguishment from their corresponding "action."

Sudo raspi-config

-> Interface -> 1-wire -> enable -> finish

- *This enables communication between the raspberry and sensor using 1-Wire. The sensor supports communication with 1-Wire only.*

Sudo restart

Sudo modprobe w1-therm

- *Force add module for the sensor to avoid any problems where the sensor is not being detected properly*

- *To ensure proper communication setup, we will edit the config file.*

Sudo nano /boot/config.txt

- *At the very bottom, add:*

dtoverlay=w1-gpio, gpiopin=4

- *to ensure that the assigned pin matches the one the sensor is plugged into*

dtoverlay=w1-therm

Cd /sys/bus/w1/devices

- *Look for a directory "28-xx…"*

Cd 28-xx…

- *The data received from the sensor is stored in the w1_slave file.*

Cat w1_slave

- *The 'YES' at the end of the first line means that data is being successfully received. At the end of the second line the temperature is shown in millicelsius. The data being displayed is the most recent data cached at the moment the command was run – in order to repeatedly display data we need to set up a loop for it, that reads the file every given timeframe.*

This section covers the installation and configuration of InfluxDB, a time-based database service system, as well as compliance commands that ultimately allow us to send data.

curl -O https://dl.influxdata.com/influxdb/releases/influxdb2_2.7.5-1_arm64.deb

sudo apt update

sudo dpkg -i influxdb2_2.7.5-1_arm64.deb

- *Download and install InfluxDB*

sudo systemctl start influxdb

sudo systemctl enable influxdb

sudo systemctl status influxdb

- *Enable and verify its status*

sudo apt install influxdb2-cli

sudo apt update

- *this allows us to use influx commands in the terminal*

- *On the browser, hit this URL: http://localhost:8086*
- *After logging in by typing username, password, organization and bucket name, a token will pop up that we will need, so we make sure to copy paste it somewhere.*
- *Now that we have created an influxdb account, we go back to the terminal*

influx config create --config-name my-config --host-url http://localhost:8086 --org [the org name] --token [the token] --active

- *set up the token as default token for future uses*

python3 -m venv influx_env

source influx_env/bin/activate

pip install influxdb-client

deactivate

- *We must create a virtual environment and install the influxdb-client library in it to use in the python script. This is because the installations would collide with the system-managed python environment*

- *Within this environment we have developed a python script that both reads the temperature from the “w1_slave” file and sends the data to a designated InfluxDB database.*

```python
import glob
import time
from influxdb_client import InfluxDBClient, Point

#Specify identification keys
URL = 'http://localhost:8086'
ORG = 'Kaffeknekt'
TOKEN = 'ETV_6VBhkfF7HzNGfOjN6F7nTvX0ye_tblcGObcB1OVJDLYxQXUWpt8NU84PJmrn6R6IV921X2eWLLJDg1wgdQ==' #token skal limes inn her
BUCKET = 'sensor_data'

#set up client between python script and influxdb
client = InfluxDBClient(url=URL, token=TOKEN, org=ORG)
API = client.write_api()


#Find the file containing the temperature data
path = glob.glob('/sys/bus/w1/devices/28*')[0] + '/w1_slave'

#Reads the contents of the file
def file_data():
    with open(path, 'r') as data:
        return data.readlines()


def temp_data():
    temp = file_data()

    #Checks and waits for the validity of the data
    while 'YES' not in temp[0]:
        time.sleep(0.2)
        temp = file_data()

    #Finds the temperature value and converts to celsius
    t = temp[1].split('t=')
    c = float(t[1])/1000
    return c


#sends the temperature data every 3 seconds
while True:
    celsius = temp_data()

    #create datapoint to send
    p = Point('Temperature').field('temperature_C', celsius)

    API.write(bucket=BUCKET, org=ORG, record=p)
    print(f'{celsius:.5f}°C')

    time.sleep(3)
```

```python
import glob
import time

#Find the file containing the temperature data
path = glob.glob('/sys/bus/w1/devices/28*')[0] + '/w1_slave'

#Reads the contents of the file
def file_data():
    with open(path, 'r') as data:
        return data.readlines()


def temp_data():
    temp = file_data()

    #Checks and waits for the validity of the data
    while 'YES' not in temp[0]:
        time.sleep(0.2)
        temp = file_data()

    #Finds the temperature value and converts to celsius
    t = temp[1].split('t=')
    c = float(t[1])/1000
    return c


#Prints the temperature every 0.21 seconds
while True:
    celsius = temp_data()
    print(f'{celsius:.5f}°C')
    time.sleep(0.21)
```

AI Usage Documentation

AI tools were selectively used across different aspects of the project to enhance productivity, brainstorming ideas and clarification of technical concepts. AI supported development, documentation, problem-solving and research. The aim of it, to promote the transparency in how AI supplemented the work, while at the same time not replacing original thinking, additionally, to ensure that all contributions strictly are guided by the academic integrity standards which have to be respected.

A.10.1 Sokaina - AI Use

During the development, I used ChatGPT as an AI tool, spesifically in the project's early stages. It provided structured guidance, always accompanied by source references of course that were immediately verified to make sure that the info is reliable and trustworthy before proceeding.
The model also helped with brainstorming by suggesting different ideas, and was also a good help when it came to suggesting for example what open-sources I can use for designing the GUI instead of doing the long process of googling it. In the cases where I met unfamiliar errors not addressed by online resources such as the Qt-Forum or StackOverflow, ChatGPT occasionally helped with debugging, although at times its suggestions became repetitive, then I know it is useless and I had to do it on my own.
It helped for instance with "override problem", A custom widget's "resizeEvent" handler was never invoked at runtime, even though it was reimplemented it in a subclass of QWidget. No compiler errors or warnings appeared. so it suggested a solution:

cpp Copy Edit

```cpp
// Correct signature in MyWidget.h
class MyWidget : public QWidget {
    Q_OBJECT
public:
    explicit MyWidget(QWidget* parent = nullptr);

protected:
    // Exact match + override specifier
    void resizeEvent(QResizeEvent* ev) override;
};
```

Figure A.10.1: ChatGPT answer.

A.10.2 Martin - AI Use

During the development process of the code running on ESP32 I have often used AI to get a quick insight into how to set up unit tests. I have also used it when debugging errors which I have never seen. Explaining the situation by for example pasting error codes has often lead me in the right direction.

Though using AI to assist in the development had its' moments of success. When the complexity of the program increased, AI was useless. This is documented in the ESP32 development logC.

Example query:

On esp32. What is a good approach two tasks work at the same time. One tasks calculates time from a rtc. And one task get sensor data from a watt meter. Both are i2c

On the ESP32, when two FreeRTOS tasks access I2C devices (e.g., an RTC and a watt meter), the key challenge is to **safely share the I2C bus** between them. Here's a solid approach:

Recommended Approach: Use a Mutex to Protect the I2C Bus

Why?
ESP32 has only one I2C controller per port, and concurrent I2C access without synchronization can cause bus collisions or corrupted data.

Figure A.10.2: ChatGPT answer.

Steps to Implement:

1. **Create a Mutex (e.g., `i2c_mutex`)**

cpp Kopier Rediger

```cpp
SemaphoreHandle_t i2c_mutex = xSemaphoreCreateMutex();
```

2. **Wrap all I2C operations in the tasks with the mutex**

cpp Kopier Rediger

```cpp
if (xSemaphoreTake(i2c_mutex, portMAX_DELAY)) {
    // Safe to use I2C here
    rtc.getTime();  // or wattMeter.getData();
    xSemaphoreGive(i2c_mutex);
}
```

3. **Run both tasks in parallel**

- `TaskReadRTC`: Reads from the RTC periodically.
- `TaskReadWattmeter`: Polls the watt meter at the needed frequency.
- Make sure both tasks call `vTaskDelay()` or `vTaskDelayUntil()` so they yield the CPU and don't starve each other.

Figure A.10.3: ChatGPT query.

Example Task Skeletons:

cpp Kopier Rediger

```cpp
void TaskReadRTC(void* pvParameters) {
  while (1) {
    if (xSemaphoreTake(i2c_mutex, portMAX_DELAY)) {
      rtc.update();  // or whatever call reads the RTC time
      xSemaphoreGive(i2c_mutex);
    }
    vTaskDelay(pdMS_TO_TICKS(1000));  // Run every 1 second
  }
}

void TaskReadWattmeter(void* pvParameters) {
  while (1) {
    if (xSemaphoreTake(i2c_mutex, portMAX_DELAY)) {
      wattmeter.readValues();  // Your wattmeter I2C read
      xSemaphoreGive(i2c_mutex);
    }
    vTaskDelay(pdMS_TO_TICKS(500));  // Run every 500 ms
  }
}
```

Figure A.10.4: ChatGPT example solution.

I used Chat GPT to help me comment my code in Doxygen conventions. I also used Chat GPT to instruct me how to install the tools needed generate Doxygen pdf.

A.10.3 Didrik - AI Use

During this project I used AI to help me come up with fitting titles for sections as seen in A.10.5.

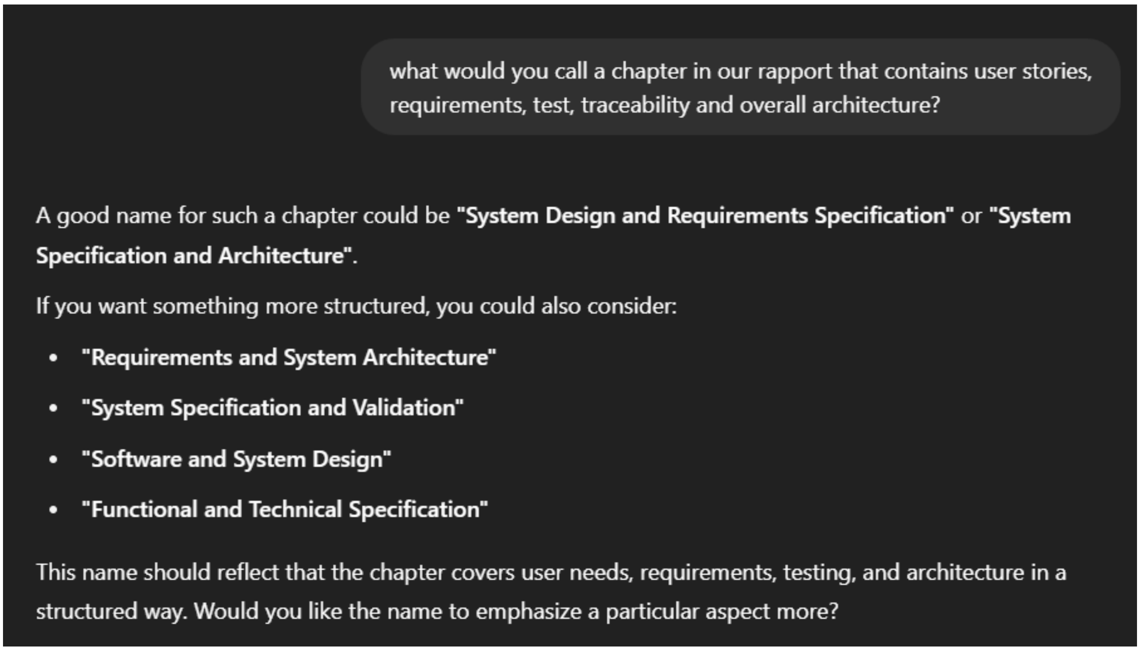


Figure A.10.5: ChatGPT Title Suggestion

I also used chatGPT to quickly help install a missing package like you see in A.10.6.

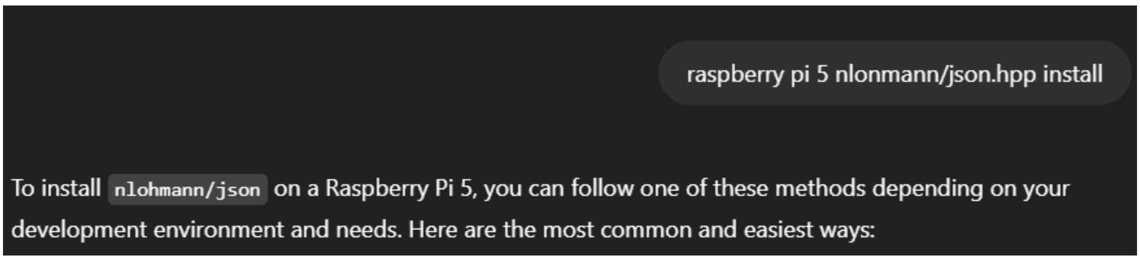


Figure A.10.6: ChatGPT Install Help

Finally if I was unfamiliar with certain libraries (like Qt) and the documentation felt lacking I would as chatGPT to give me an example so I could get a grasp on its general setup like you see in A.10.7.

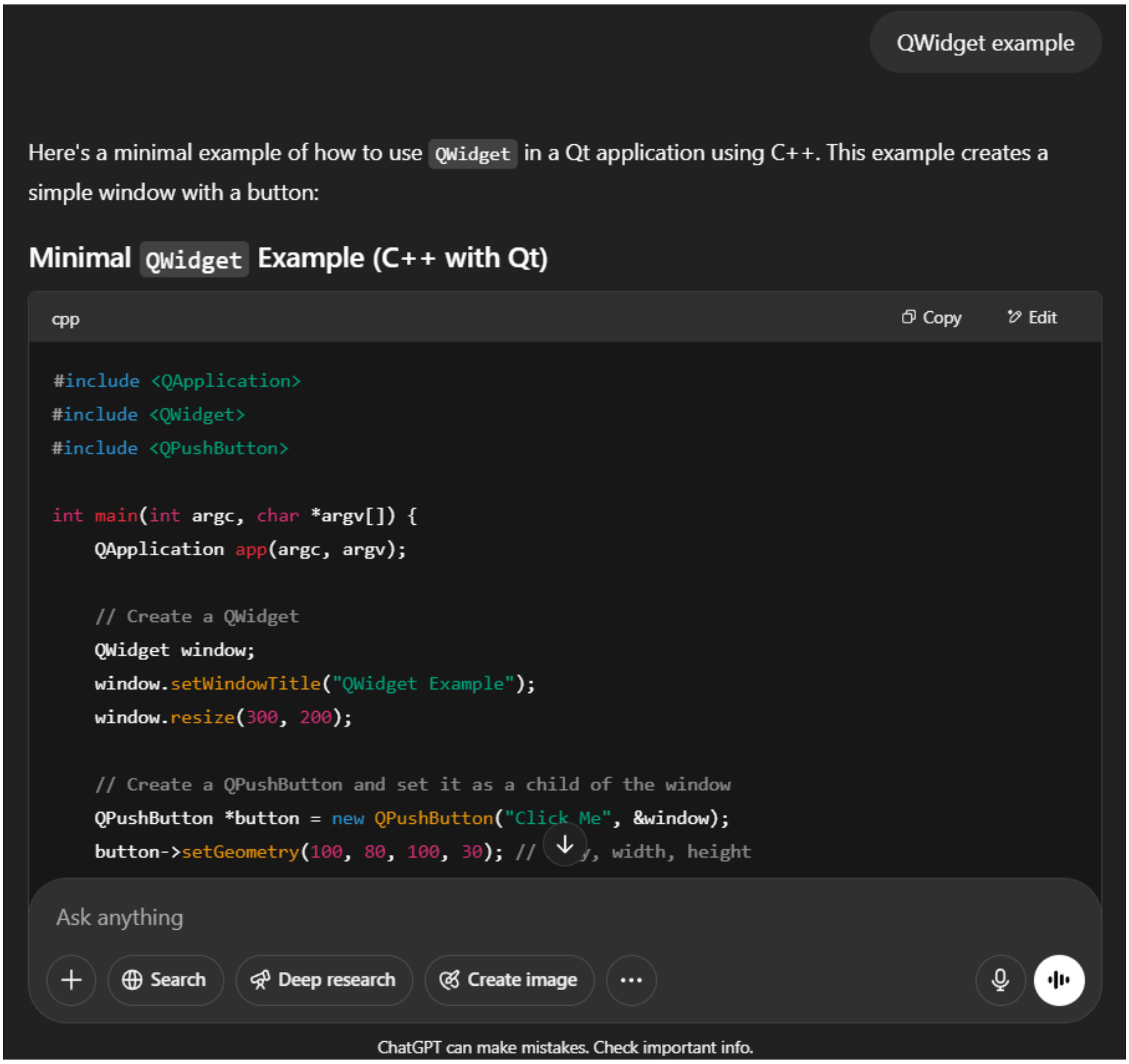


Figure A.10.7: ChatGPT Give Example

A.10.4 Kadir - AI Use

I use AI to confirm my understanding of newly acquired knowledge from trustable sources such as influxdata.com. That way I can more easily think of plans on how I will approach the task at hand and start carving a path to the solution.

It was also helpful in discovering new functions that I would otherwise never find, to solve my code problems by asking vague questions on how to do something.

Not to mention was it helpful in installing applications and dependency programs so I could set up work environments.

A.10.5 Ivan - AI Use

AI was rarely used, but was used to point in the right direction when it came to formatting the LaTeXdocument, giving better words and spelling. AI tools helped occasionally, but would usually end in a tangent where a solution was not found.

A.10.6 Mikolaj - AI Use

AI usage on my part was involved with ChatGPT, where i consulted on the following topics:

Vocabulary and grammar: Some of the texts in the document had a too informal approach of reflection and explanation of different topics. Additionally, there was always room to improve the

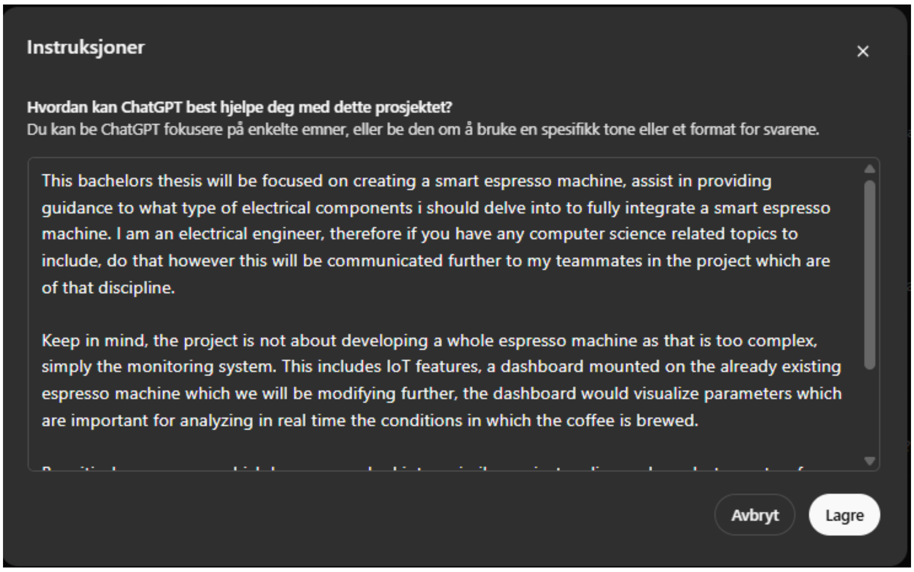


Figure A.10.8: Instruction window with a restricted topic focus for the trained ChatGPT model.

grammar, i've used ChatGPT for help in relation to this.

Trained model: A trained ChatGPT chat model was developed by me, which was focused on the bachelor's thesis and its problem domain. ChatGPT allows you to implement instructions to a chat which restrict the model to the specific topic you'd like expertise or brainstorming on.
Better understanding of user stories and requirements: I've used ChatGPT for brainstorming around real-world scenarios of user stories and requirements connected to espresso machine technology domains. To better understand, analogy was used.

Appendix B

Technical Documentation

Component Procurement Strategy

MMS | *IBM*

The choice of components was made with a strong emphasis on scalability, risk mitigation, and flexibility for integration. To reduce risk for the project and accommodate iterative development, multiple quantities of key components were ordered during the procurement phase. This approach not only compensated for potential hardware failures or damages, however, also enabled the team to explore alternative applications of the same components when unexpected challenges arose.

The strategy behind the procurement relied heavily on extensive research into the design and functioning of espresso machines, which is presented in the main body of the report. We explored brewing dynamics, thermodynamics of heat exchangers, and industry standards for espresso preparation, by delving into these factors, the theoretical and practical aspects of espresso machine functionality directly informed and set the argument for every critical hardware decision.

B.1.1 Threaded sensors

MMS | *IBM*

To establish the system's purpose as a real-time monitoring system for espresso brewing, it was essential to obtain both accurate and reliable pressure and temperature data directly from within the espresso machine's hydraulic system. The parameters are central to our system insight above all things, although explicitly linked to one of the core user stories:

> "As a user, i want to in real-time, monitor the parameter during the brewing time, so that i can assess and ensure the quality of the espresso A.2."

While it doesn't define specific accuracy constraints, the intent behind it clearly implies the necessity of measurements that truly reflect the conditions at the point of extraction. For parameters such as brew pressure, this rules out completely non-invasive implementations, as external sensors wouldn't be able to capture spontaneous dynamics of pressure, in addition to not being in direct contact with the fluid pathway.

To satisfy the requirements, we utilized threaded sensors, both for pressure and temperature, these Transducers specifically designed for direct mechanical integration into the system, in conjunction with both threaded adapters and sealing methods.

The implementation of threaded sensors contributed to following key advantages:

· Direct fluid contact established accurate and fast pressure and temperature readings.

· Stable mounting, which hinders potential measurement errors.

· Repeatability, testing results were consistent with comparable over time data.

This approach certainly necessitated some form of invasiveness into the hydraulic system, although it was justified mainly by the commitment to deliver credible insights of the brewing conditions.

Power Strategy: Isolated Supply

During the phase of system design, a possibility arose which was considered although wasn't encouraged, that was to source the power from the electronic control board in the espresso machine to the monitoring system. However, as is common with commercial espresso machines, especially those manufactured by Faema, any detailed schematics or electrical documentation of the board are almost certainly proprietary and protected, making them unavailable to the public. Without the certainty of current limitations, voltage levels, and internal circuitry, this posed unacceptable risks for integration to the system. An isolated supply approach was therefore decided upon and this aligned with our responsible engineering practice under uncertain conditions.

B.2.1 Power Source **MMS** |

The espresso machine is connected to the 3-phase IT-network at the workshop, it utilizes 240V Alternating Current (AC) voltage. To power the monitoring system, two phases had to be selected and connected to the Power Supply Unit (PSU)'s input terminal as shown in Fig. B.2.1

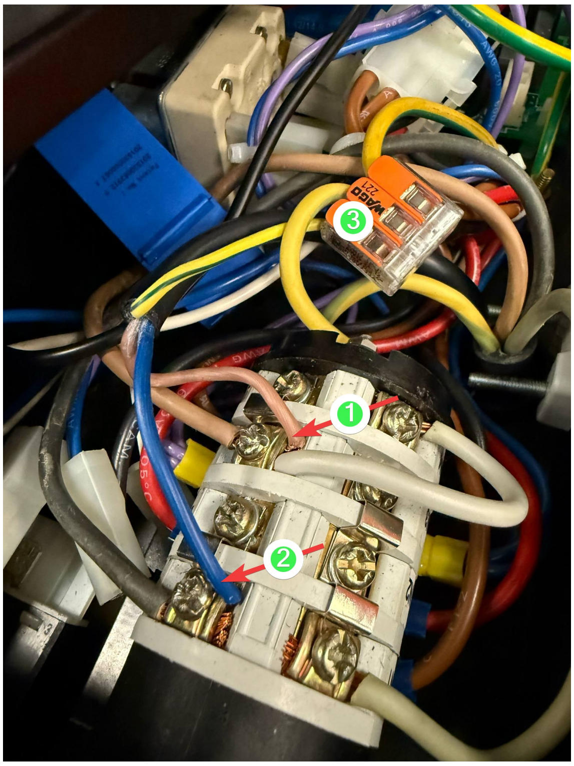


Figure B.2.1: Electrical connections on the main switch with a separate cable from the PSU connected to the corresponding phases. 1) The first phase L1. 2) The 2nd phase L2. 3) Common grounding

B.2.2 Power Supply **MMS** | *IBM*

There are a variety of factors that contributed to the final decision on selecting a suitable power supply, these being:

5V output: Since both the RPI-5 and ESP32 have to be powered by a stable 5V, the output of the power supply has to be rated for that specific voltage.

Resistance to input surge voltage: The power supply itself is rated for operation at 88 - 264V (AC), however for a whole 5 seconds, it can sustain a surge of 300V (AC).

Power: For powering the whole monitoring system, not a lot of power was needed, in fact, by approximation we determined that 50W is more than enough, besides, this was already the lowest power rating for an ideal power supply candidate.

Figure B.2.2: LS50-5 Power Supply Unit, taken from [98].

PCB Design

IBM | *MMS*

The project is about monitoring data and information from sensors that have been placed on the espresso machine. These sensors all have inputs/outputs and follow certain criteria and protocols (I2C, SPI, GPIO). We figured that it would be better that a PCB would be designed, to minimize the cable management in a way. Going for the PCB approach, we limit the wiring/cables that exit the machine, to just one cable, or potentially no cable, depending on whether the system is going to be fully wireless.
The PCB is going to be powered from an internal transformer that is going to be housed in the espresso machine. This transformer is going to convert 240V taken from the internal 3-phase 240V system. Over to 5V to supply the PCB with ground and voltage to power everything the PCB is interfacing with.

Designing the PCB was a back and forth process, where discussions between the electrical members and some of the data students led to certain design choices. The components being the sensors, Real Time Clock (RTC), ADC, Filters, MCU ESP-32.

The PCB for this project was designed on KiCad, with its default library, and some components were found online to satisfy our vision. This being either a footprint, symbol, 3D model, or all 3 combined. The 3D models were not necessary for the design, but were a nice feature to have in the design process, to visualize the progress. Some of the footprints that were found online were altered using data-sheets to expand certain footprints to fit more pins.

Nearing the end of the design process, we encountered an obstacle when it came to ordering the PCB. Involving the PCB being designed with hidden/buried vias in mind. We sent our design to JLCPCB to be quoted, and then a couple of hours later, received a notification that JLCPCB does not currently support hidden/buried via's. Reading this, we thought to ourselves "we should have checked that". We gauged our options, either of which to redesign the PCB and change all of the hidden/buried via's over to regular vias, or, ordering the PCB from another distributor. We went first with the latter and checked other distributors whether they supported hidden/buried vias and the price for manufacturing our board. We found that PCBWay supports hidden/buried vias, but at an increased cost, around two times the price as JLCPCB. We eventually chose to redesign the board, replacing all the vias and changing the sizes of the Surface-Mounted Device (SMD) resistors and SMD capacitors, from 0603 to 1206 SMD resistors and 0603 to 1206 SMD capacitors.[1] After the redesign the PCB got accepted by JLCPCB and then manufactured.

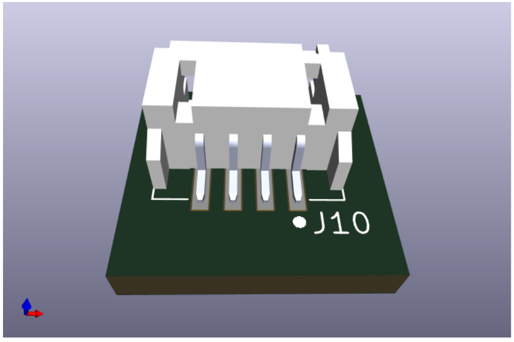
[1] This was after we had a meeting Ole Eirik Solberg Seljordslia where he commented that we should change the size of the components so that we could be able to solder the SMD's by hand, if we needed, and that we should go for regular via's

B.3.1 Component **IBM** | *MMS*

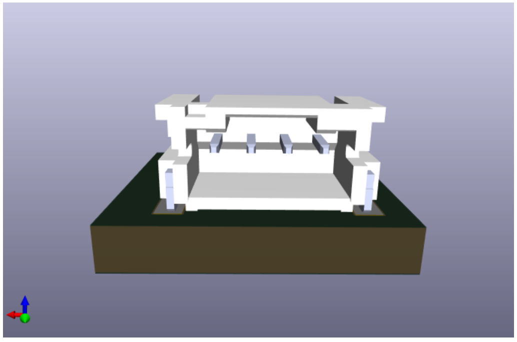
This section is about the components that are utilized by the PCB as following.

B.3.1.1 Headers **IBM** | *MMS*

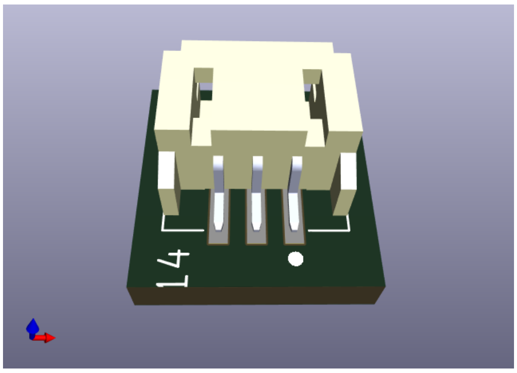
The board utilizes Japan Solderless Terminal (JST) Male 2.0P SMD Headers in 3 pin and 4 pin formats. The JSTs are the main inputs/outputs from the PCB that let the PCB gather information from our sensors. 3D models of these headers can be seen in Fig. B.3.1a, B.3.1b, B.3.1c, and B.3.1d



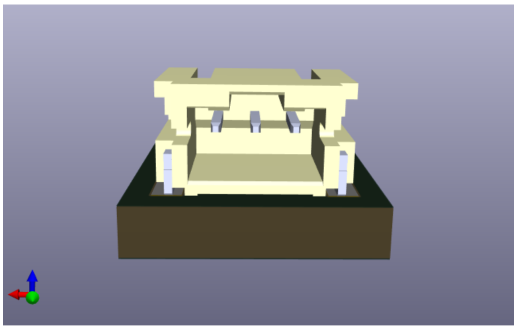
(a) 4 pin Male SMD Header back view



(b) 4 pin Male SMD Header front view



(c) 3 pin Male SMD Header back view



(d) 3 pin Male SMD Header front view

Figure B.3.1: JST male 2.0P SMD Headers

The PCB also uses "regular" female headers so that an ESP32 can be connected directly on the PCB and is designed with three female headers specifically for the thermal amplifiers, and a single 2 port screw-block terminal, for the input voltage of the PCB.

B.3.1.2 Sensors **IBM** | *MMS*

The PCB is designed to have 10 sensor inputs. All these sensors have different purposes, signal types, and protocols. Digital sensors and amplifiers are placed on the first layer shown in Fig. B.3.11, while analog sensors are placed on the bottom layer of the PCB, shown in Fig. B.3.13.

Digital Sensors:

The PCB is designed to take in three digital sensors that utilize an I2C protocol, where all of them share the same SCL and SDA from the ESP32. Using the properties of the I2C protocol we are able to assign addresses to each sensor so that none of them overlap each other. The sensors that we are talking about are the accelerometer SEN0409, Pressure Sensor NPI-19J-200G2, and Wattmeter SEN0291. The PCB is also designed for three digital sensors that are connected to amplifiers that utilize SPI protocol. And just like the I2C share certain pins from the ESP32. This time being Serial Clock (SCK), Master Output Slave Input (MOSI), and Master Input Slave Output (MISO). From there the ESP32 becomes the Master, and the three amplifiers become slaves, if the system requires it.

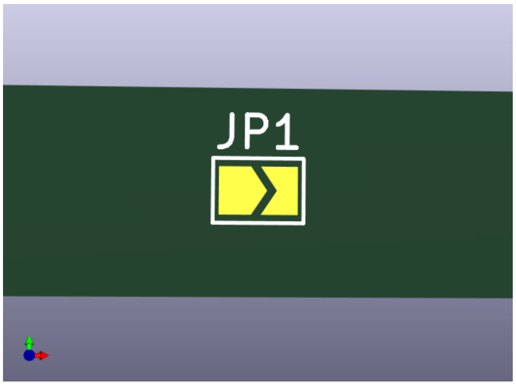
Analog sensors:

The PCB is also designed with 4 analog sensor inputs in mind. All of the analog inputs utilize a 3 pin JST Male 2.0P SMD Headers and are connected to an ADC that is integrated into the PCB, which then connects directly into the I2C line of the ESP32. Currently the PCB only has a need for two analog inputs, but is designed so that if it ever has need for more analog inputs, we have it covered. Or up to four in total analog inputs that utilize a 3 pin JST input as shown in Sec. B.3.1.1

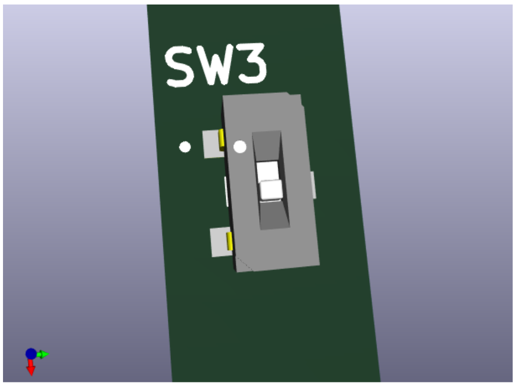
B.3.1.3 Dip Switches

IBM | *MMS*

During the planning phase of the PCB, a discussion came about, whether it would be better to utilize a solder bridge shown in Fig B.3.2a or a dip switch shown in Fig. B.3.2b. Using a dip switch for the ADC works, on the grounds that it is used to change the slave address on the ADC, by either sending a high signal or no signal. The discussion was mainly about the RC filters. Where the signal output of a analog sensor would either be directly sent to the ADC, or to pass through a RC filter first, then to the ADC. The design started with solder bridges in mind. Since the design principle of the solder bridge is to bridge two solder pads together with solder, or a 0 Ω resistor [80] to close the circuit, alternatively to cut the pad if it is a NC solder bridge. This soldering technique is used in some PCBs to re-route the circuit in certain circumstances or to cut certain function off [64]. Theoretically, the solder bridges are big enough to house resistors and capacitors, so they could be interchanged if their values did not meet the requirement for the RC filter. It is also not recommended to unsolder the bridges to many times since this might lead them to deteriorate, and cutting a NC solder bridge is only recommended if you don't plan on using that channel again, since it is not recommended to solder over a cut NC solder bridge.



(a) Solder Bridge



(b) Dip Switch

Figure B.3.2: Options For Switching Tracks

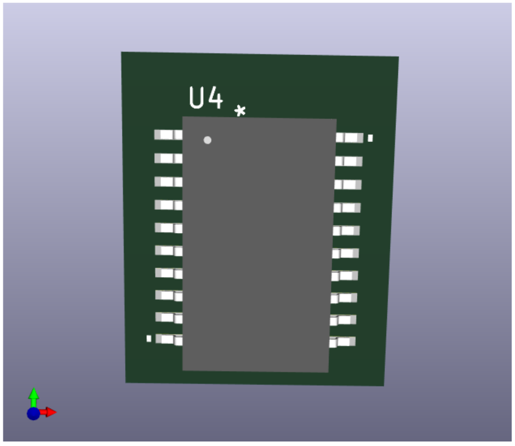
In the end, dip switches were selected to toggle between sending an analog signal through a filter or to directly go to the ADC, and then the ESP32. Using dip switches gives the possibility to toggle between the states, [83] and change the components of the LP filters values when needed. The analog filters and dip switches can be seen more in Sec. B.3.2.4.

B.3.1.4 ADC **MMS** | *IBM*

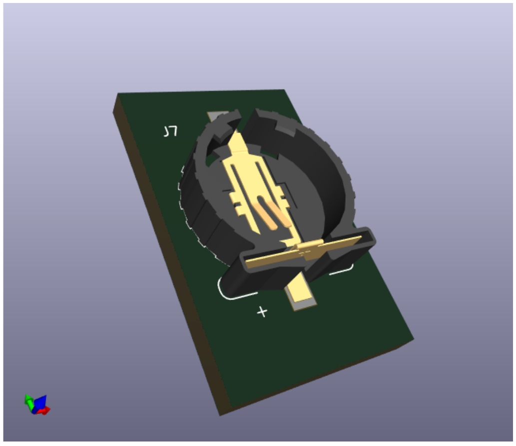
Initially, the ADC was tested through an ADS1115 module, because of its impressive capabilities and accuracy measurements, it was decided that the very same ADC could be designed on the PCB, and traced out accordingly.
DFRobot has designed their own components and their datasheets and schematics are open for the public. Ideally it was to implement the very same schematic integrated in the PCB.

B.3.1.5 RTC **IBM** | *MMS*

While developing the PCB a DS3232 Precise RTC was used. The RTC worked, but considering that this would just be an extra module hanging of the ESP32, why not just just implement it into the board. All the components of the DS3232 were found from the datasheet that came from the distributor and then implemented into the design. The RTC consists of a DS3232SN Crystal oscillator shown in Fig. B.3.3a, two decoupling capacitors, two pull-up resistors, and a button-cell battery holder (S8411-45R) shown in Fig. B.3.3b. The symbols of these components can be seen in Fig. B.3.7.



(a) RTC - DS3232SN



(b) Button-cell battery holder - S8411-45R

Figure B.3.3: Main RTC Components

B.3.2 Main Schematic

IBM | *MMS*

The schematic section of PCB design is one of the first steps when it comes to designing a PCB. In the design, symbols are placed for components, and connection points are added between components. The symbols can be formatted to have footprints and 3D models assigned to them for a later phase, as shown in Sec. B.3.3. Here lies the schematic of the PCB as shown in Fig. B.3.4. It is hard to see but the schematic is divided into parts and sections, and we will go over them individually.

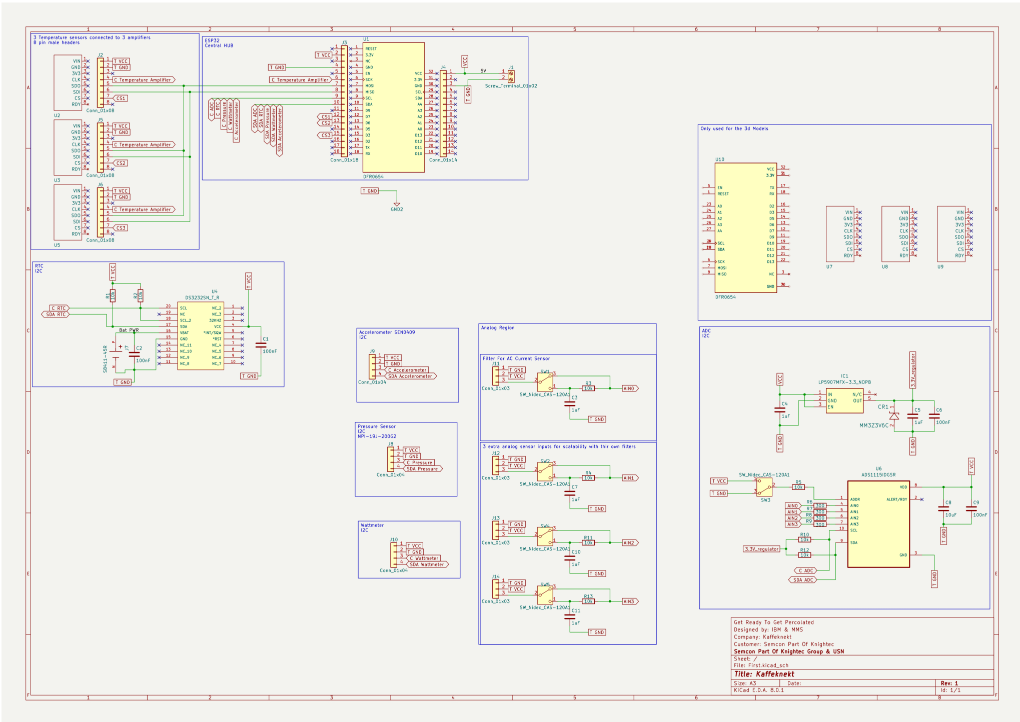


Figure B.3.4: Schematic for the PCB

B.3.2.1 ADC Schematic **IBM** | *MMS*

In Fig. B.3.5 is an ADC which utilizes an I2C protocol. During the design we thought about having an ADC module that the sensors connect into and then into the PCB, but we figured that we could implement an ADC into the PCB itself. In this schematic block we have a dip switch that changes the I2C slave address of the ADC, this switch alters between a voltage and no voltage. A voltage regulator that is connected to 5V and regulates the voltage to 3.3V. Both the voltage regulator and ADC have decoupling capacitors to act as voltage reservoirs and prevent voltage spikes. These decoupling capacitors should be placed as close as possible to their Integrated circuit (IC)'s. The ADC has a couple of pull up resistors aswell. Being 300 Ω resistors for analog inputs.(AIN0, AIN1, AIN2, AIN3). And two 10k Ω resistors between the voltage regulator and I2C lines (SDA, SCL). The entirety of the ADC ensemble is located on the analog layer.

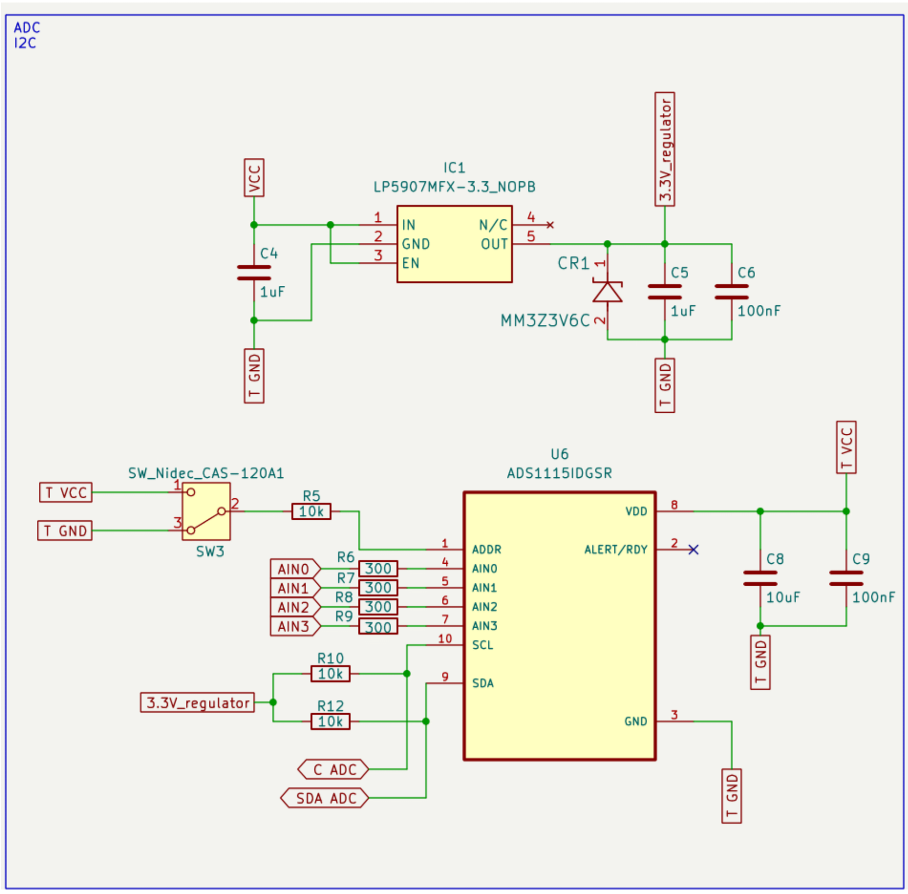


Figure B.3.5: Close up of ADC

B.3.2.2 ESP32 and RTD Amplifiers Schematic

IBM | *MMS*

In Fig. B.3.6 is the ESP32 labeled as the Central Hub, and the inputs for the RTD amplifiers. The RTD amplifiers (MAX31865 RTD PT100) have an SPI interface and are connected to the PCB using 8 pin headers. All three of the amplifiers share the same VCC, GND, CLK, MOSI, and MISO from the ESP32 except for the chip selects on the amplifiers. These chip selects have been assigned to their own pins on the ESP32.
The symbol for the ESP32 can be ignored, since it was only used as a reference for the pin inputs/outputs. The ESP32 has two sets of headers, an 18 pin header, and a 14 pin header. The 14 pin header only takes the VCC and GND from a screw-block terminal that is connected to a voltage supply. While the 18 pin header connects to most, if not all the other components. Here the ESP32 interfaces with SPI and I2C. Where SCL and SDA connect to all of the I2C components, while SCK, MOSI and MISO connect to the SPI components.

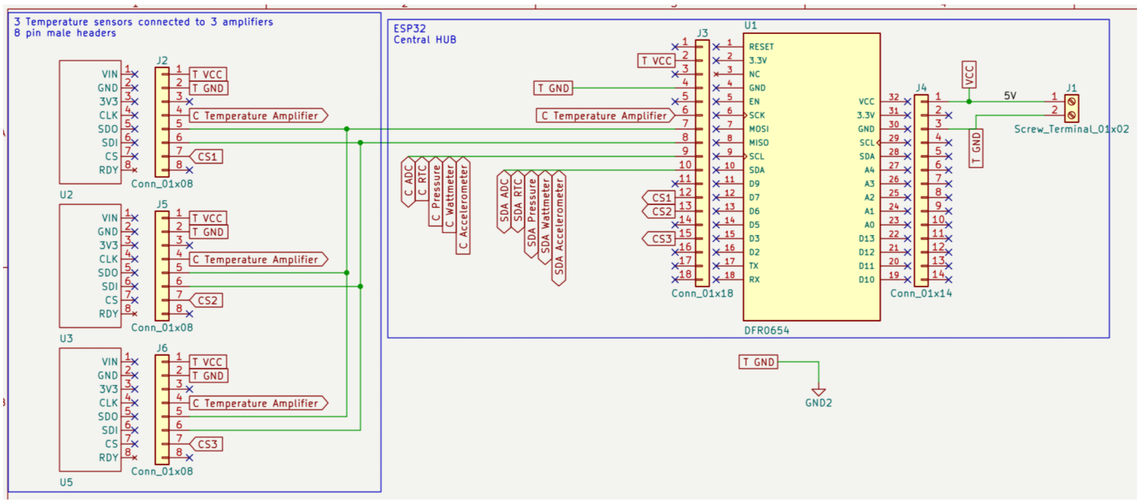


Figure B.3.6: Close up of the ESP32 and RTD amplifier inputs

B.3.2.3 RTC Schematic **IBM** | *MMS*

We had the option of using an RTC that could have been connected into the PCB, but we decided to implement an RTC directly into the board. In Fig. B.3.7 lies the RTC which utilizes an I2C protocol. The main component in this schematic is the DS3232SN crystal Oscillator. This IC is used to keep and track time in case of an power outage, so that the system does not lose track of time. The RTC is connected to a 3V VCC under normal operation, but schematic also includes a SMT Coin Cell Holder, that houses a battery that powers the RTC if the system would ever go offline. This battery is only meant to be a backup for the RTC, so that the RTC will keep counting the date and time, when the entire system is not connected to power.
This part of the schematic also includes a decoupling capacitor. One for the battery and one for the RTC, IC, and again should be placed as close to the component as possible. The design also utilizes 10k Ω pull-up resistors for the I2C lines (SCL, SDA).

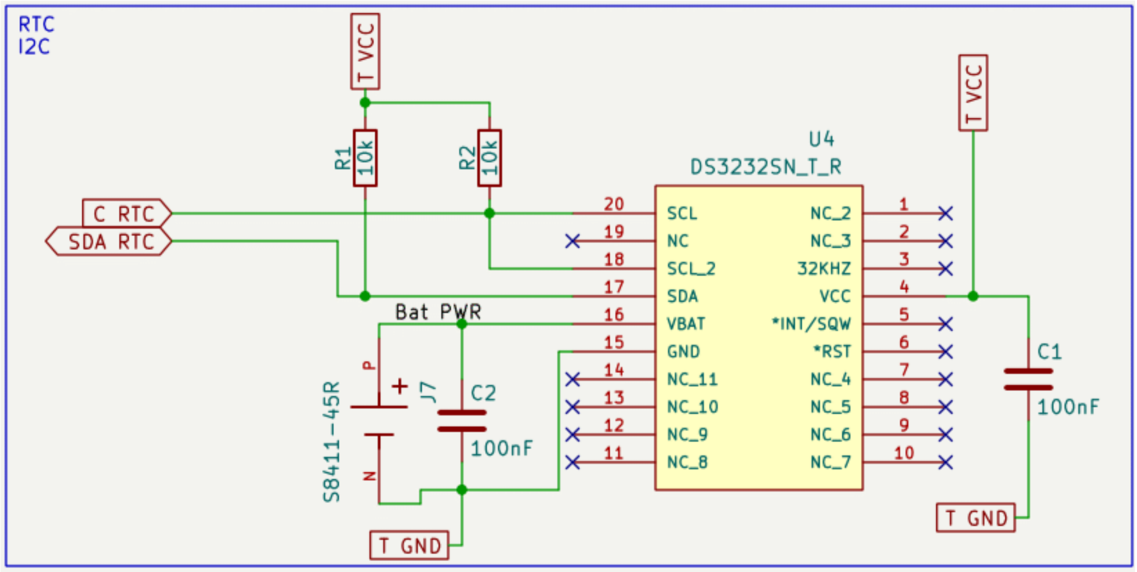


Figure B.3.7: Close up of RTC

B.3.2.4 Analog Region

IBM | *MMS*

The analog Region shown in Fig B.3.8. The reason we call it the analog region is because this part of the schematic is located on the last layer of the PCB. The analog region comprises mostly of 3 pin JST male connectors seen in B.3.1c, resistors, capacitors, and dip switches. All four of the analog inputs are designed in the same way, where a JST male port receives sensor data, and sends it directly to the ADC to convert the data. Alternatively, through a Low-pass filter to filter out noise and then to the ADC. This is toggled by using a dip switch on the board.

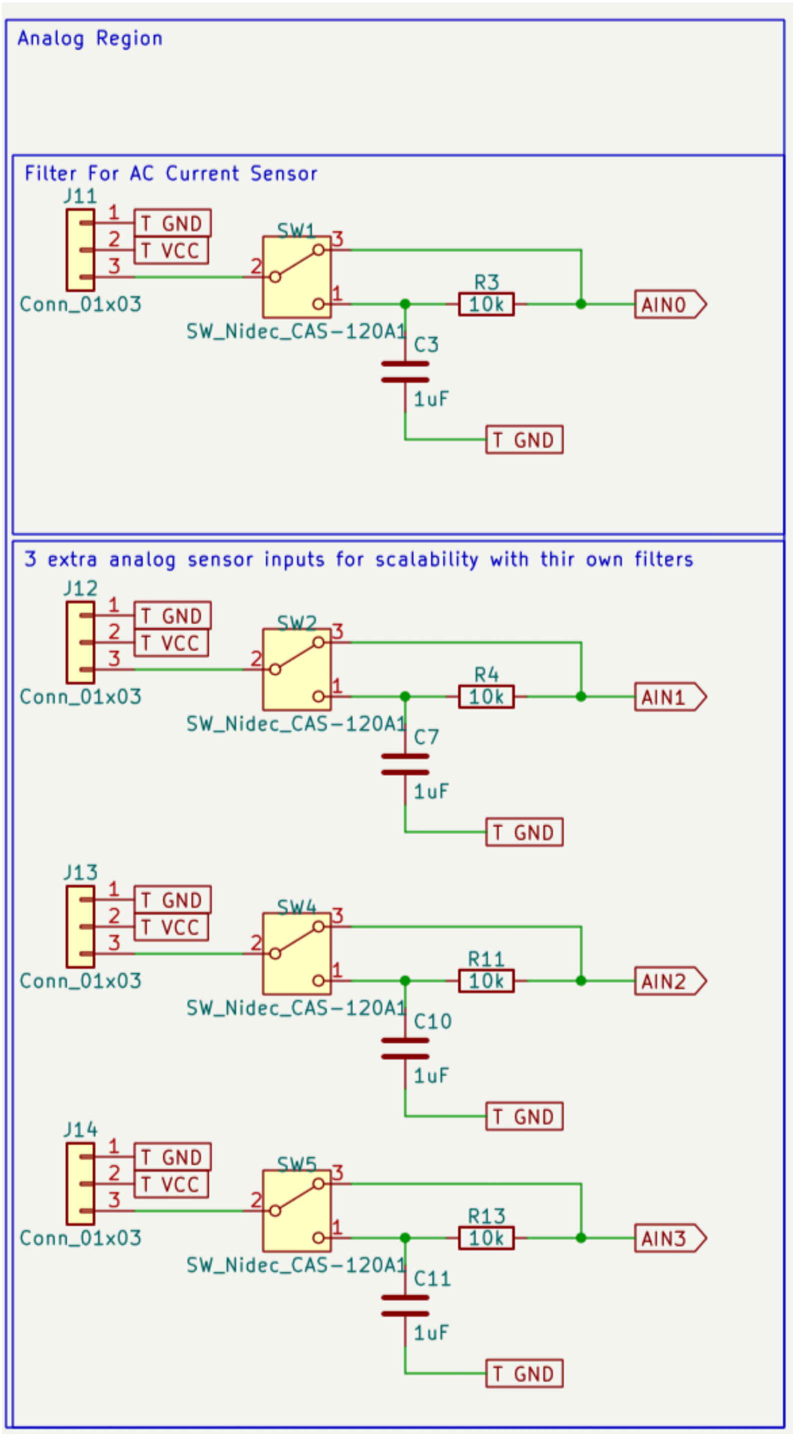


Figure B.3.8: Close up of the Analog Region

B.3.2.5 Digital Region

IBM | *MMS*

The Digital Region shown in Fig B.3.9 are the simplest schematic blocks here. They consist only of three 4 pin JST male connects shown in Fig. B.3.1a. These headers only have two purposes, to provide VCC and GND to external sensors, and send back information from sensors utilizing I2C interfaces (SCL, SDA).

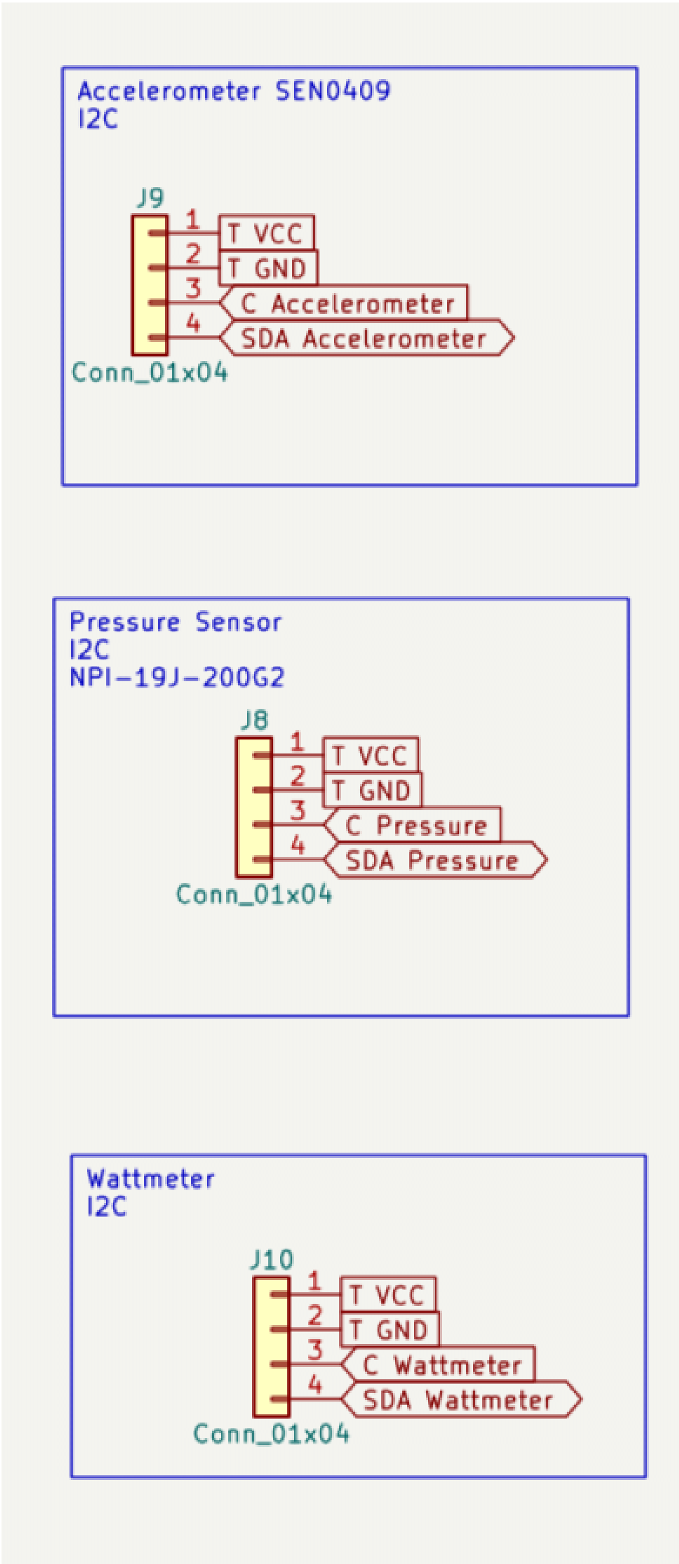


Figure B.3.9: Close up of the Digital Region

B.3.2.6 Symbols for 3D models **IBM** | *MMS*

This schematic block is only used for 3D models. The symbol marked as DFR0654 is the ESP32, and the three unmarked symbols are all MAX31865 RTD amplifiers. This is for the 3D viewer of the PCB and helps visualize the process. Most if not all components used in the schematic have 3D models assigned to them, but these components are going to be physically soldered on the board. While the ESP32 and MAX31865 RTD amplifiers are going to be mounted on headers and then held down with screws.

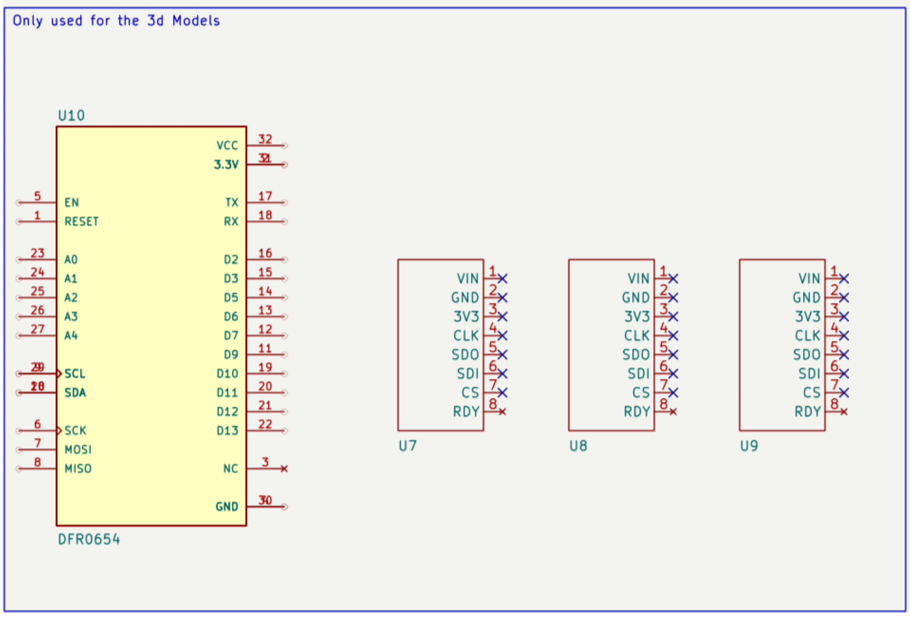


Figure B.3.10: Close up of the symbols only used for 3D models

B.3.3 Layout **IBM** | *MMS*

The layout section of PCB design, is the final stretch of the design process. In KiCad, the design was selected to have four layers. Where the first layer is designated as a digital layer where silkscreens, footprints pad-stacks, and traces, are placed to determine where the components are placed on the final PCB. Second layer is designated as the ground plane, Third layer is designated as the power plane, and fourth layer is designated as the analog layer. The reasoning for why there are distinct layers is so that the process is streamlined, and routing traces around components on the first and fourth layers is simpler, since the utilization of ground and power planes simplifies the process.

B.3.3.1 First layer **IBM** | *MMS*

The first layer can be seen in Fig B.3.11. Shown in the figure are "red lines", these lines are known as traces and are used to bridge between connection points and currently have a width of 0.2 mm. The red squares are SMD pad-stacks and either used for soldering points between components and or mounting pads for components, a closer look can be seen in Fig B.3.1. On the board there are four mounting holes labeled as "REF**" for the PCB, and ten mounting holes for an ESP32 and RTD Amplifiers. The first layer utilizes vias to connect to the second layer as seen in Sec B.3.3.2, and third layer as seen in Sec. B.3.3.3 to provide components power and ground, as mentioned in Sec. B.3.2.3, some components have to be placed a certain distance from specific components. This being the RTC IC and its decoupling capacitor. After all, the components are placed in the desired area, an "edge cut" function can be used to set the board dimension, giving the final form of the first layer.

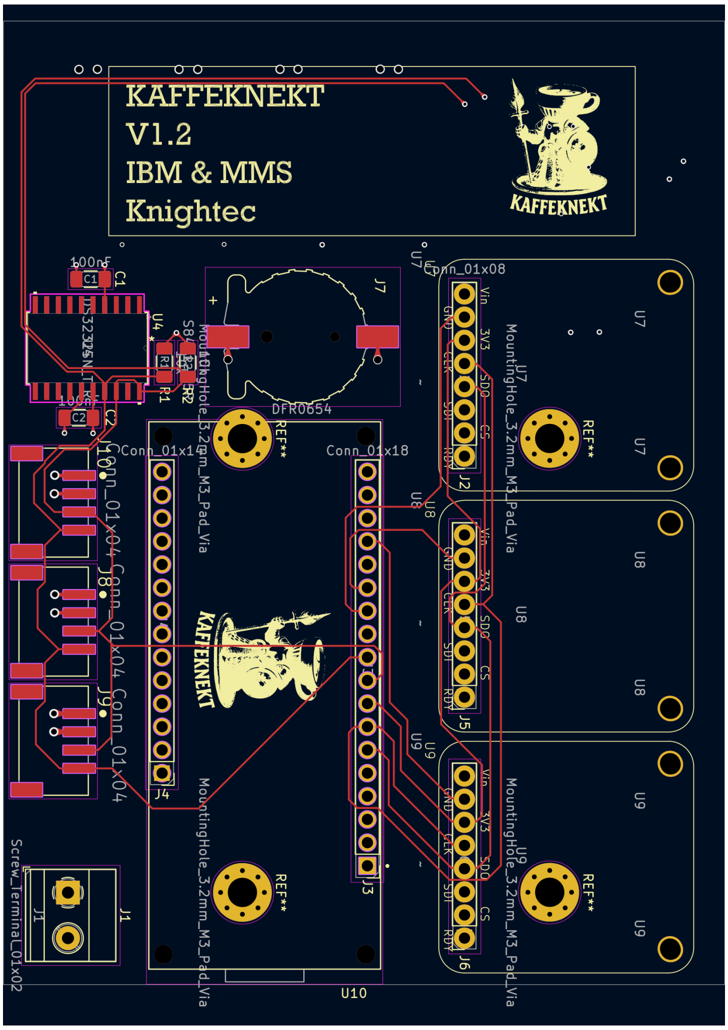


Figure B.3.11: First Layer - Red Squares = SMD Padstacks, Yellow Lines = Silkscreens, Red Lines = Traces

B.3.3.2 Second layer **IBM** | *MMS*

The main purpose of the second layer is grounding all the components. A ground plane is made, visualized as a green square in Fig. B.3.12 usually on the second layer of a four layered PCB. This layer is connected to the first and fourth layer using vias. These vias are not placed directly under the solder pads, since this could increase manufacturing costs. Instead, they are placed at the sides of the solder-pads and then using traces to connect to their required destinations. This is only for the SMD, but the through hole mounts work differently. Since the component is going through the entire board, making certain layers avoid certain through holes and connecting some through holes to the layer itself.

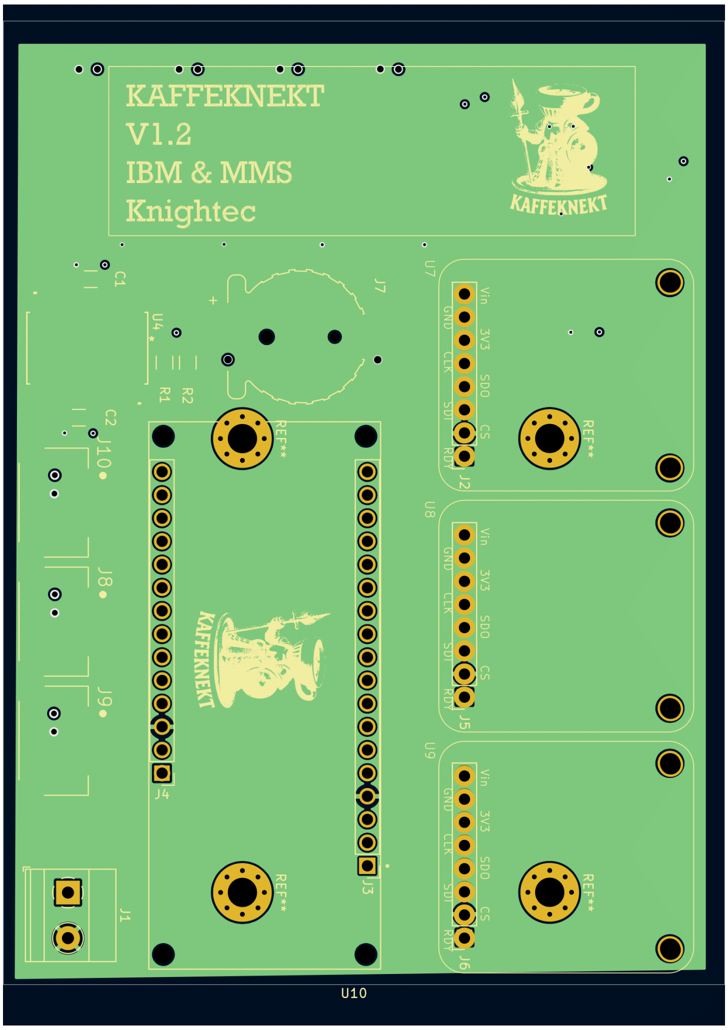


Figure B.3.12: Second Layer - Ground

B.3.3.3 Third layer **IBM** | *MMS*

The third layer is generalized as the Power layer. It has the same function as the second layer shown in Sec. B.3.12 although, with power planes. This layer is concerned with diverting voltage/amperage around the board, connecting to the first layer, and the fourth layer. Unlike the ground layer, this layer has 3 planes which can be seen in Fig. B.3.13. Starting with the smallest plane, is the battery power plane. This plane is only connected to the RTC and can be seen in Sec. B.3.2.3. The second largest plane is the 5V plane. It's only connected to two components. This being the input screw-block-terminal and the input voltage of the ESP32 as seen in Sec. B.3.2.2, and the voltage regulator in the ADC schematic as seen in Sec. B.3.2.1. The third and largest plane is the 3V plane that is powered by the ESP32, this plane powers most of the components, such as amplifiers, sensors, and the main voltage source of the RTC.

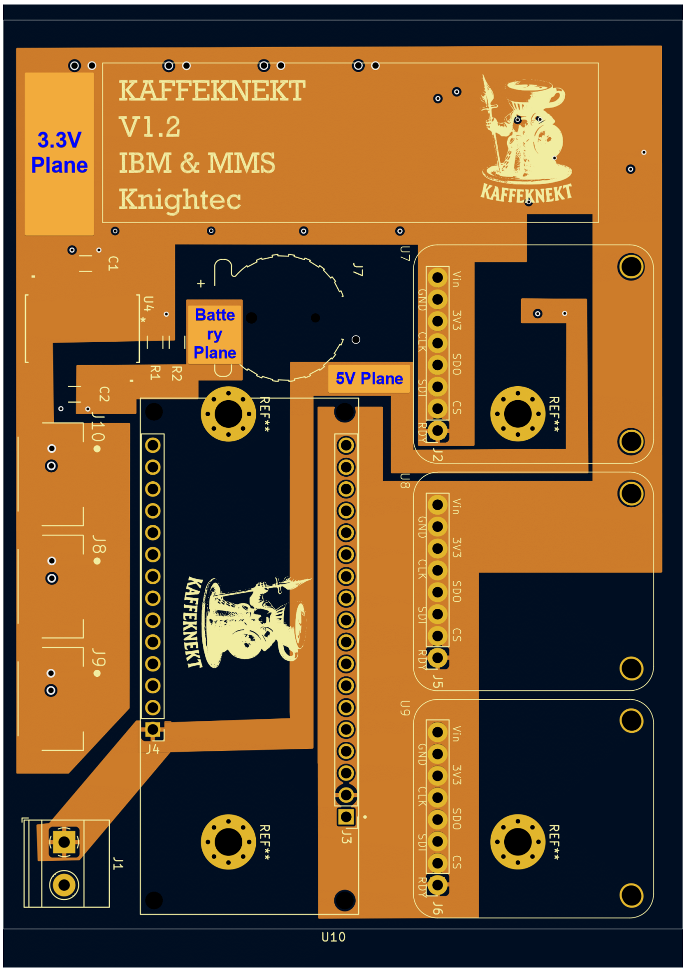


Figure B.3.13: Third Layer - Power

B.3.3.4 Fourth layer **IBM** | *MMS*

The fourth and final layer seen in Fig. B.3.14 is the analog layer. This layer contains the ADC schematic seen in Sec. B.3.2.1 and the analog schematic seen in B.3.2.4. This layer has multiple dip switches, resistors, capacitors, and four JSTs. The "layout" principle of this layer is the same as the first layer shown in Sec. B.3.11, however, colored blue. The square rectangles are SMD pad-stacks, and the blue lines are traces used to connect between the components, and the trace width is at 0.2 mm.

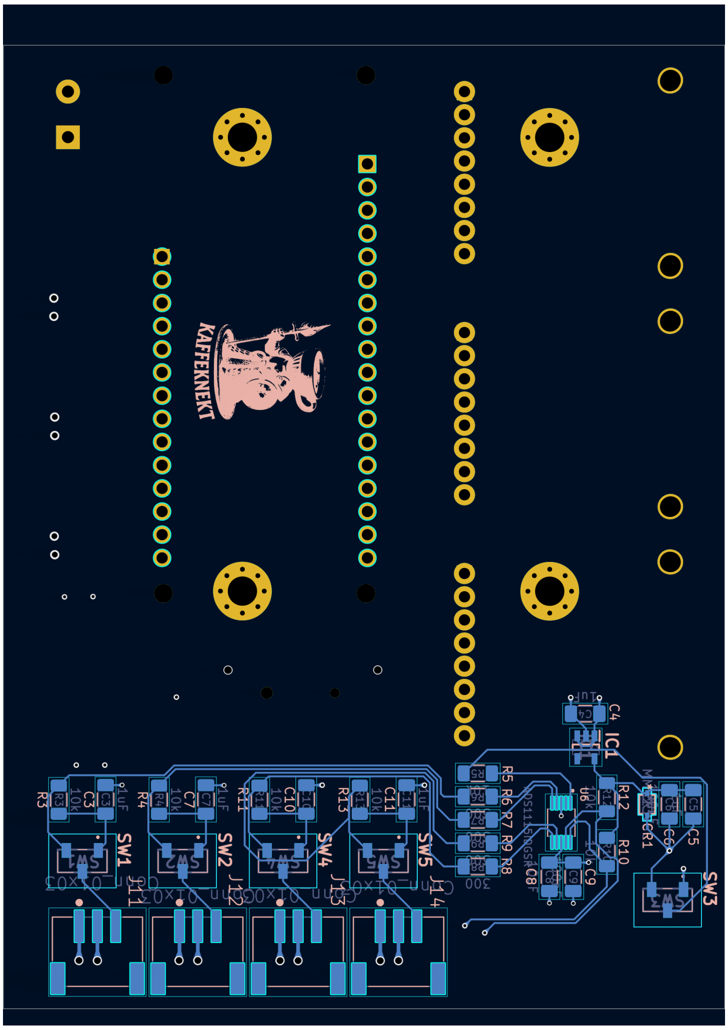


Figure B.3.14: Back Layer - Blue Squares = SMD Padstacks, Blue Lines = Traces

B.3.4 Surface Level Testing **IBM** | *MMS*

After retrieving the PCB from the distributor JLCPCb, a week after ordering. The PCB was set to the side, since the design of the container was currently a higher priority. More of the container can be read in Sec. B.7.1. Because the PCB arrived during printing. After fitting to see whether the PCB would fit into the container that was designed to for the PCB and the PSU, which it did. Then it came to testing, and the reason why this section only covers surface level testing that can be read more in Sec. B.3.5. The only tests that were able to be done to the PCB, was a standard connection confirmation as shown in Appx. A.4 (T-PCB1.0 - T-PCB1.4). This being that a multimeter was used to see whether the pads on the PCB had connections to one another. In other words, if the four layers of the PCB were connected properly through vias. The layers can be seen in Sec. B.3.3. All the pins that should have been grounded, were grounded, and all the different power planes were connected to their desired pins.

B.3.5 Late Arrival **IBM** | *MMS*

There was an issue during the later phase of the project when it came to the PCB. This problem being that the components for the PCB haven't arrived. This can be matched to a risk that we could unfortunately encounter during the bachelor period that can be seen in Appx. A.1 (RU-26.0) The team was notified by Semcon Part of Knightec Group that the components got ordered a week before the deadline. Because of this inconvenience, a full test of the PCB can not be done, until after the deadline.[2]

[2]The finished product of PCB with components and tests will be added to the USB stick after the deadline. If the components arrive on time

Mechanical Integration **MMS** | *IBM*

As we've investigated the espresso machine more thoroughly, it became apparent that it would be unlikely that we'd be able to find a sensor which could be directly screwed inside the Grouphead (for temperature measurement) or to the heat exchanger (for pressure measurement). This section presents the mechanical considerations and tools involved in the installation of threaded sensors.

B.4.1 Couplings and Adapters **MMS** | *IBM*

In a hydraulic system of an espresso machine which introduces pressure inside it, components which could be threaded were of great interest to us. This would contribute to very good tight installations which also would measure precisely the conditions occurring in the system.

When the threaded components were chosen, a vast amount of time was researched upon understanding how to couple them and screw them inside compatible adapters which have both the same pitch and thread dimensions.

Since our threaded sensors are of an imperial dimension, it was difficult to find compatible reduction couplings and adapters, although they were found.

B.4.2 Sealing and Leak Prevention **MMS** | *IBM*

For all of our threaded sensors, it was necessary to apply a thread seal tape (PTFE) which prevents leaks from pipe threads in plumbing projects. Without it, a leakage was most definitely going to occur, in fact it did even as our sealing was not wrapped with adequate amount of layers on the threading.

B.4.3 Pressure **MMS** | *IBM*

Since pressure measurement is one of the most desired requirements for our customer. It was important to actually implement this into the system, with a sensor which is capable of resisting the temperature conditions where it is installed.

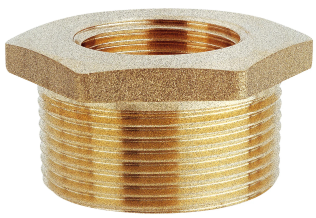
At the first iteration, it was decided a T-coupling would be necessary to install into the system. This was already becoming inconvenient both for the customer and for us, as this is an invasive modification to the system. This was communicated to our customer and finally it was decided that we'd get the green light to move with that modification. The requirement for pressure measurement set for ourselves is for it to be precise, for that to be satisfied, the measurement should take place right before the Grouphead at the same time after the heat exchanger output. An ideal position would be at number 8 in the figure shown in Fig. 2.2.5.

As we've explored more of the espresso machine's water system. A discovery was made that the heat exchangers have a brass fitting plug which is screwed at a branch connection at their outputs. This was unscrewed as we determined that we won't need to modify the machine with a T-coupling.

Figure B.4.1: Water pressure sensor installed at the heat exchanger output.

We've installed the water pressure sensor which we've ordered on our first component list, although it was discovered that this specific sensor won't tolerate the temperature which occurs inside the heat exchanger, this being above the boiling point. It was decided that a different sensor should be installed for this system, one which would be capable of that temperature, and the pressure measurement range.

The chosen sensor which is explained in more depth in Sec. 4.4.1, has a thread dimension of 1/8" and the branch connection only fits with dimensions at 1/4". What was needed was both a brass elbow fitting, and a threaded reducing coupling 1/4" to 1/8" shown in Fig B.4.2a, and Fig. B.4.2b



(a) 1/4" to 1/8" (male-to-female) reducing coupling

(b) 1/4" to 1/4" (male-to-female) Brass elbow fitting

Figure B.4.2: Brass fitting parts taken from Watski

Brew-Event Detection **IBM |**

Brew-Event Detection has been a main topic for the group since we started the project. As in how are we going to detect when an espresso shot is being poured and how do we count these so called "events". This being that brew detection has been one of the main requirements that we had to follow since the beginning of the project, but pushed to the side during development. Then later brought up, when we were asked how the process of finding such a method was going.

There was a debate between the group, on how we were supposed to implement this into the system, what method, how would it work, function, would it be too invasive, and whether we are able to implement it into the system.

B.5.1 Methods **IBM |**

In this section, in no particular order, we are going to mention/define some of the methods that we thought about using for brew detection. And then ending with why we made the decision we made.

- The team has squabbled between using a load cell to detect a weight difference for when a espresso is being made.
- A microphone that is connected to a filter, that filters out all noise except for when the machine is actually brewing. The microphone would be placed in the near vicinity of the brew-head.
- Using the same AC current sensor that is used to measure how much power the machine has used, and extracting the data for when it spikes in operation. But using this solution might be problematic, since the machine uses more amperage when the boiler is being filled, and when the heating elements engage.
- Connecting into the "Purge" valve on the brewing head and extrapolating the delta voltage to see if that can be used to determine "espresso output". This method could have been used, but then we would have needed to step down the voltage to a more manageable level.
- Utilizing an optocoupler in conjunction with the "Purge" valve. Since the valve in Normally Open (NO) when the machine is not brewing, and when an espresso is being made, the valve closes when a voltage is introduced to it. This voltage could be taken and stepped down to a manageable voltage for an optocoupler to function. The optocoupler works by having a light emitting diode and a light sensitive transistor, where when the LED lights up, it short circuits the transistor within the chip causing a free path for a signal/voltage to travel. This is where the input of the LED diode is connected to the input of the valve, and the output of the LED is grounded. Then the input of the transistor is connected to a voltage, in this case probably 3 to 5 volts, and the output to a pin on the ESP32 that can read the data.
  On paper, this option was attractive for the electrical members of the group, but we started to run out of time to develop optocouplers into the system. If this solution would have come sooner in the development process, it might have gotten used, and implemented into the PCB as well.
- Using another AC current sensor, but utilizing them in the input panel, where we are able to detect when a button has been pressed. This method was the last one we thought of, before settling on a method. We ended up choosing this method. This being that we had a spare AC current sensor. The day after this method was being conceived, it got tested with the system and worked for our purposes. As in, it was able to detect an amperage spike, or an "event" with simple code, which could flag the event and track the brew time.

B.5.2 Implementation **MMS** | *IBM*

It certainly was a foresight on the group's part to order multiples of components, mainly to confront hardware failures, should they have occurred. Since we still had a spare AC current sensor, it was decided to implement this as a "trigger" for the start and stop of brewing events. Although how the journey led us to that decision will be presented further in more depth.

B.5.2.1 Solenoid Valve

MMS | *IBM*

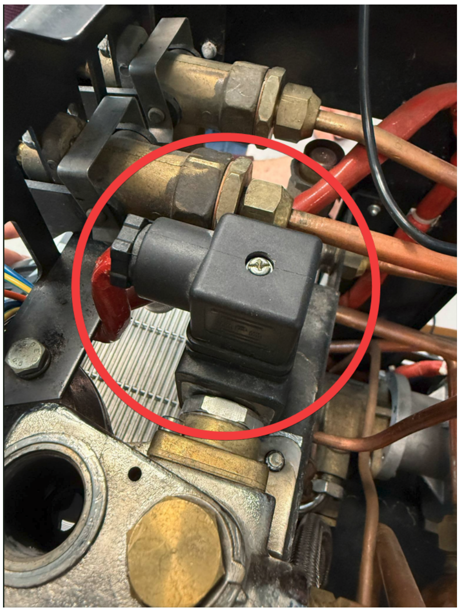


Figure B.5.1: The solenoid valve (circled in red) is controlling pressure release after brewing

It was observed that the solenoid valve seen if Fig. B.5.1 is related to the brewing in some way. Since it wasn't evident to us what its purpose was, by doing a bit of research on forums, examining datasheets of individual parts, in addition to verifying it with earlier tests and maintenance, we understood that this component is concerned with depressurizing the group-head right after the brewing has finished.

Our approach to solenoids was at a novice level at best, we assumed that it had similar functions to that of a relay. Although that wasn't the case. Solenoids have a "continuity" they need to satisfy, its nothing more than just the validation if the solenoid coil isn't broken, one of the very first things was to find out where this continuity has a connection. To do this, a resistance measurement was applied, between terminals 1 and 2 as seen in the illustraion seen in Fig. B.5.2.

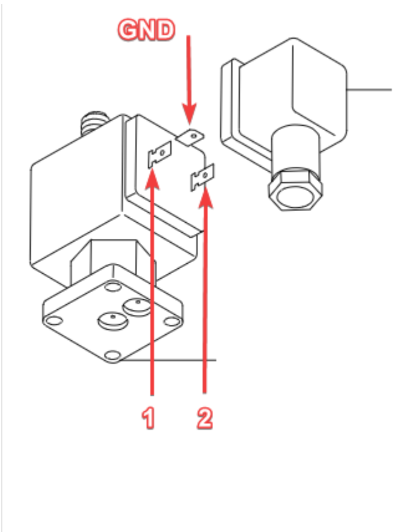


Figure B.5.2: Schematic of the solenoid valve taken from [59]. (Edited with Sharex)

We verified the solenoid as a "Normally Open" variant by identifying two specific functions associated with that configuration:

Measurement: To determine that a solenoid valve is operating accordingly, the voltage between two terminals has to be measured, namely, (1 and 2). Between these terminals an AC voltage of 240V was measured, however only when the brewing had started, and not in idle mode (no brewing).

Pressure release: The solenoid valve discharges the pressure when the valve is "deactivated" or "de-energized".

B.5.2.2 Non-invasive detection **MMS** | *IBM*

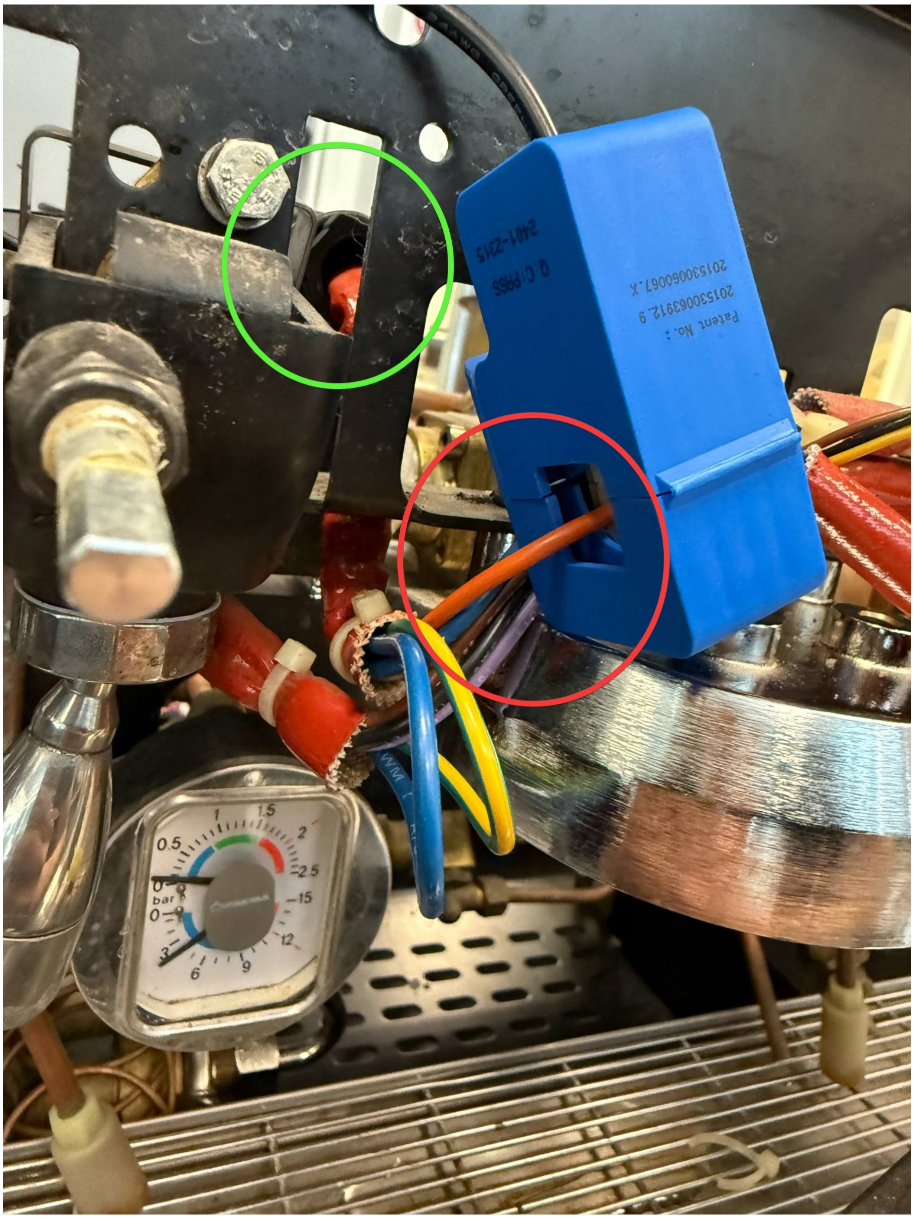


Figure B.5.3: One of the phase wires from a solenoid terminal, clamped with an AC current sensor (encircled in red). Partly visible cable coming out of the solenoid valve (encircled in green).

By clamping around the wire which is connected to one of the terminals as shown in Fig. B.5.3, we can detect an instantaneous change in voltage. However, the journey to that discovery lies in how we understood the circuit in the control board of the espresso machine. Since there aren't any available datasheets out on the website of the original board. We've found a board manufactured by Chinese designers, which to our luck, was compatible with other espresso machines, among them of course, Faema, and with a schematic that corresponded to the same electrical connections in our control board.

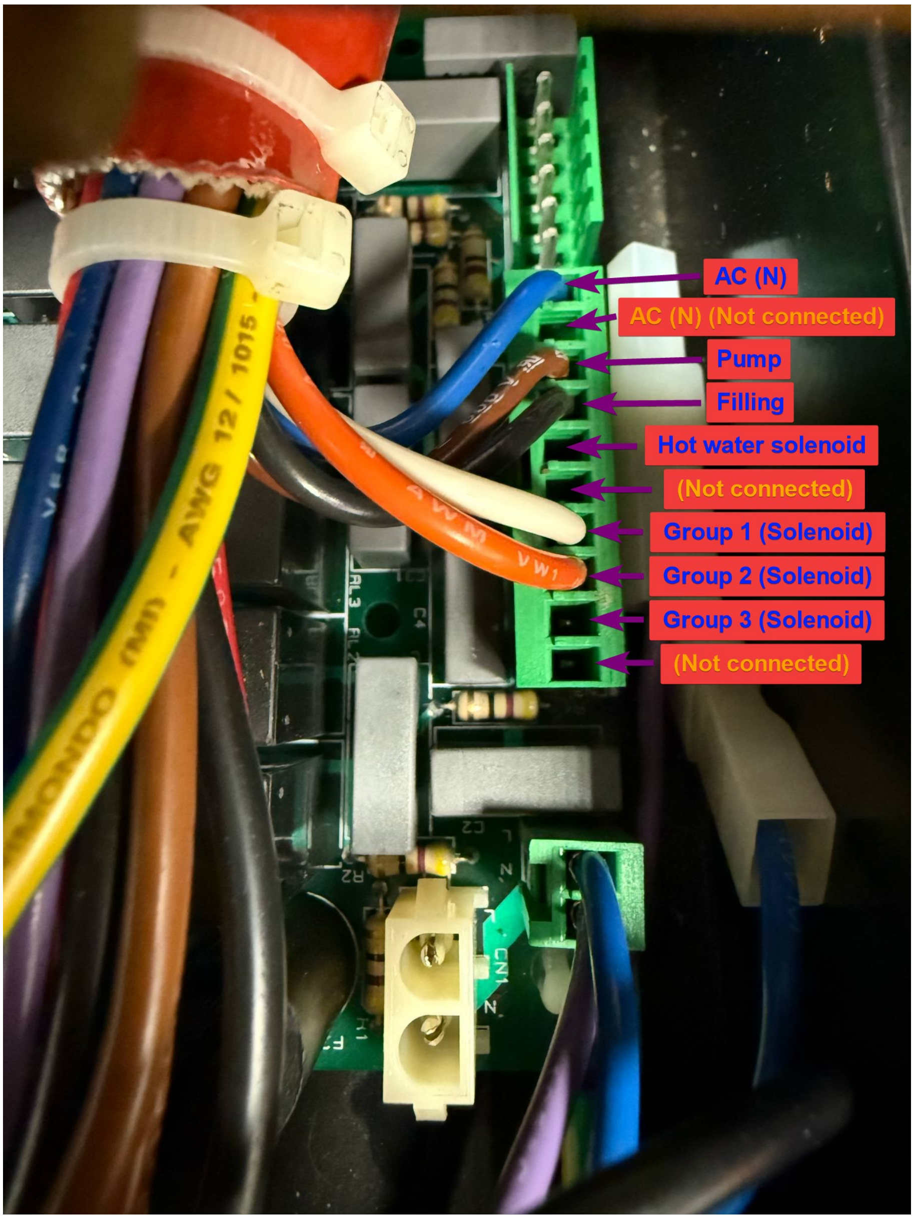


Figure B.5.4: Terminal block used for controlling solenoid valves and managing AC power distribution. Each connection is labelled with their corresponding outputs.

The connections labeled are compared with the schematic of the Chinese made control board. Notice the gray and orange wires shown in Fig B.5.4 which are the outputs to their separate solenoids. Since we are monitoring the orange wire, which is connected to one of the terminals of the left solenoid, we can efficiently monitor the brewing events.

Control Board

MMS | *IBM*

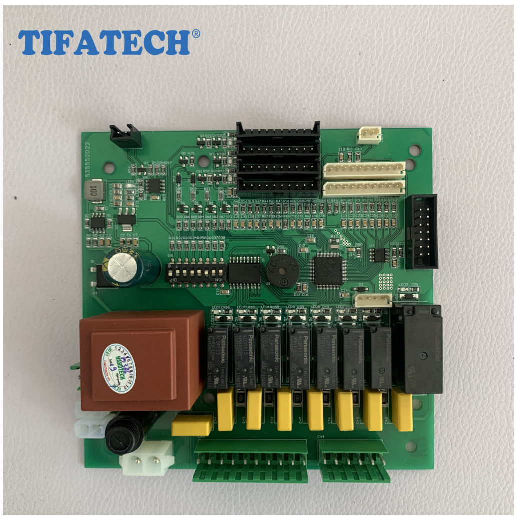
The espresso machine has a vast amount of electronics which need to work together, this to for example extract the right yield of an espresso, programming purposes for the panel, or to reduce waste. The control board is concerned with connecting all of these components directly to a microprocessor which translates these signals and stores them for the user. This part of the report is concerned with elaborating how we understood the control panel, and adapted despite not having enough technical information about the panel itself.

B.6.1 Schematic

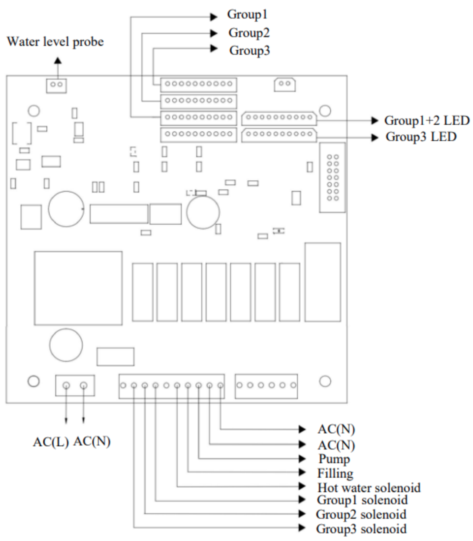
MMS | *IBM*

One of the disadvantages linked to the product is that it is a commercial one. No technical documentation is to be found, except for the user manual. Anything which is concerned with electrical components or even the control board are nowhere to be found.

Luckily, there are chinese-made variants of control boards which are compatible with a vast array of espresso machines, including FAEMA, ours specifically. They are available for the public to buy and they come with a datasheet. A variant shown is shown in Fig. B.6.1a, and it's schematic in Fig. B.6.1b.



(a) The physical motherboard



(b) Schematic of the motherboard

Figure B.6.1: HTCCA coffee machine motherboard taken from [107].

If we look into the control board in our espresso machine and its connections, intuitively the connections correspond to the chinese-made variant of it. Notice the connections on our control board showed inf Fig. B.6.2.

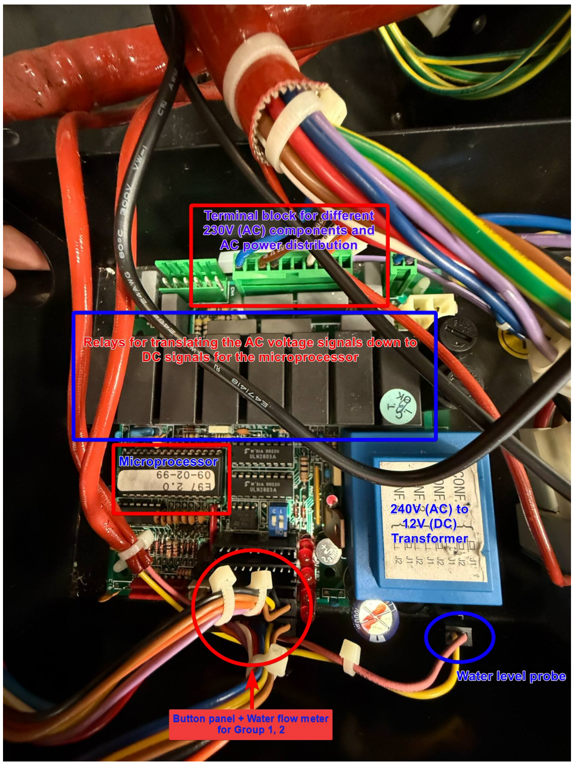


Figure B.6.2: Overview of the control board connections, with highlighted information.
-

3

[3] Both motherboard and control board are the same expressions for the unit, however for order purposes, we'll constraint ourselves to referring it to a control panel.

3D Printing

B.7.1 PCB Container **IBM** | *MMS*

Considering that the PCB is going to house a ESP32, it is a great idea to have a container that contains the PCB. This is where container comes in. The container was designed to house the PCB and a 5 V PSU that powers the PCB. The container was designed to fit next to the power panel of the espresso machine, and slightly under the boiler. The container is also designed too have fans to draw in cooler air from the bottom of the chassis of the espresso machine, and then pushing out the warmer air as it rises.[4] With this in mind, the cover of the casing is designed in a way so that air can escape, and is removable. The casing also utilizes magnets on the cover of the container, so that it is removable, and on the bottom of the container, so that it would move less during operation. [5] Isometric images of the container can be seen in Fig. B.7.1a, and Fig. B.7.1b, and a side profile of the container with the components in Fig. B.7.2. The current design of the container that has been printed, is a prototype, and was printed using PET-G filament. The filament was chosen because of its durability and high resistance to heat.

(a) Isometric view from the top **(b)** Isometric view from the bottom

Figure B.7.1: Isometric views of the container

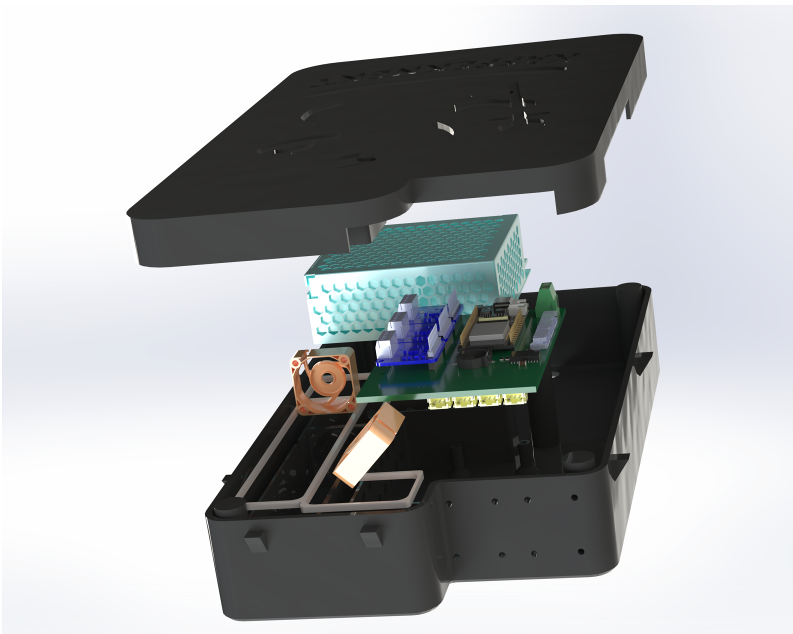


Figure B.7.2: Side view with components in place

[4]The SolidWorks file is will be in the USB stick

[5]With great assistance creating and designing the container for the PCB and PSU came from Rasmus Ullestad Relling, from the planning phase, design phase, and to the printing phase

B.7.2 Touch screen and RPI casing **MT** | *DAB*

The touch screen we selected for the system does not have an enclosure (the electronics are exposed). We had to develop an enclosure of our own. We had help from other groups in developing the casing [6].

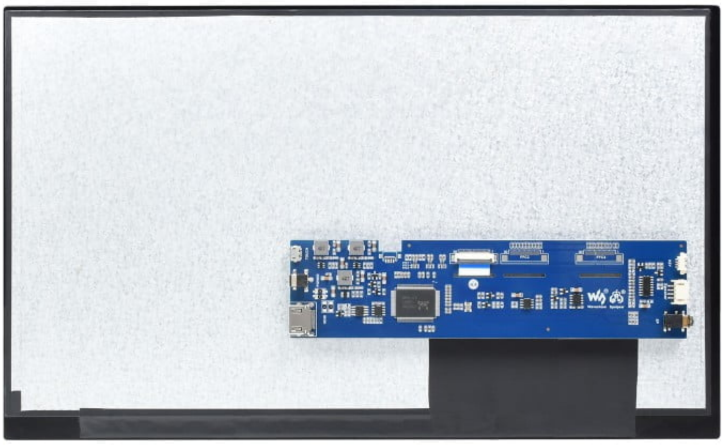


Figure B.7.3: Exposed electronics on screen.

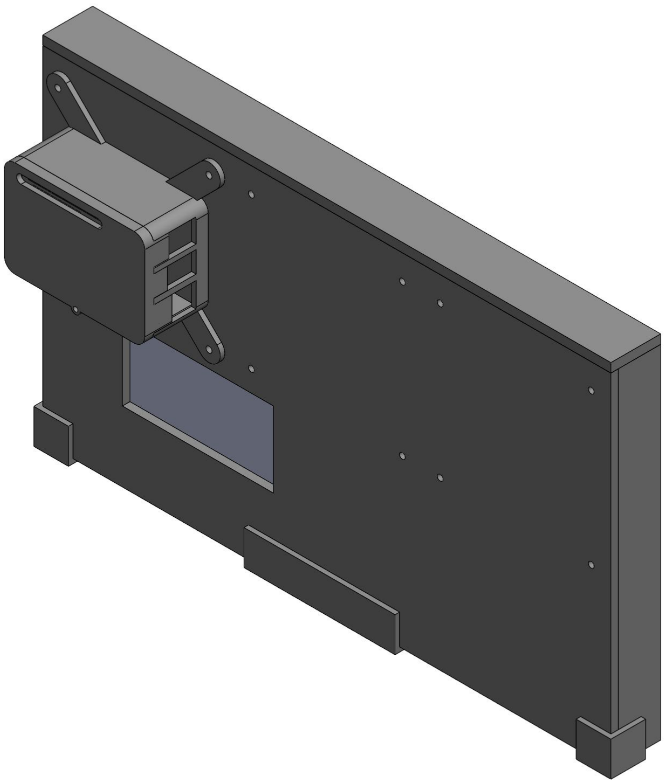
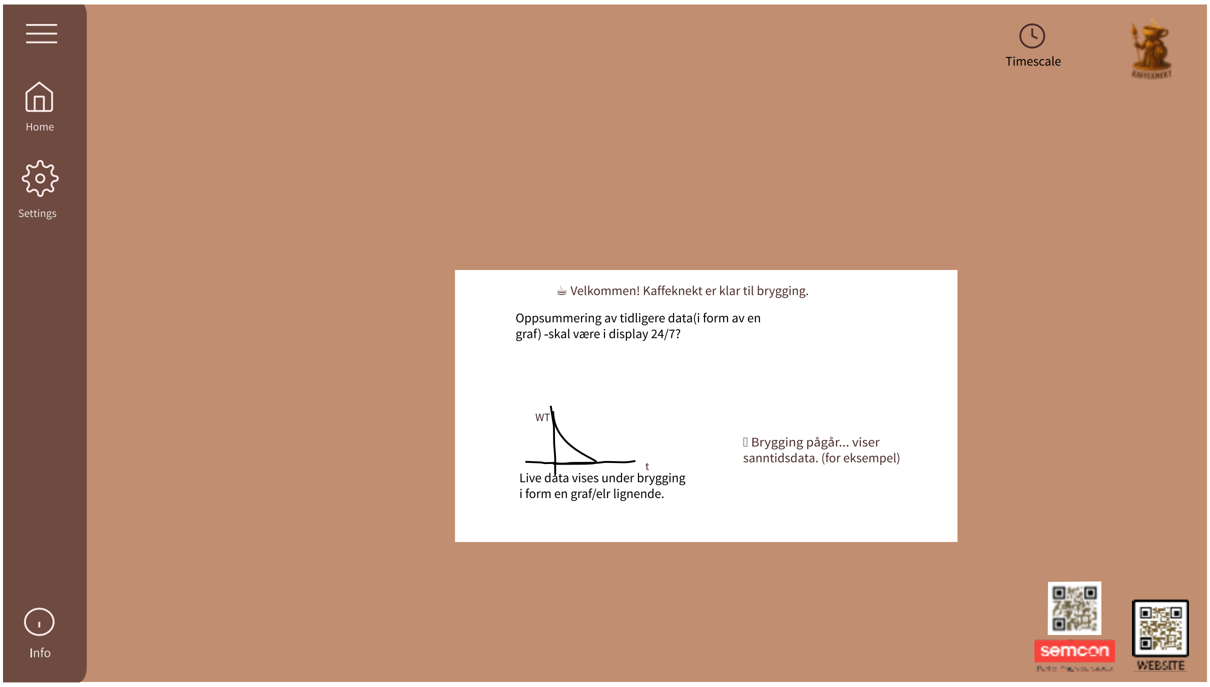


Figure B.7.4: 3D-model of final design.

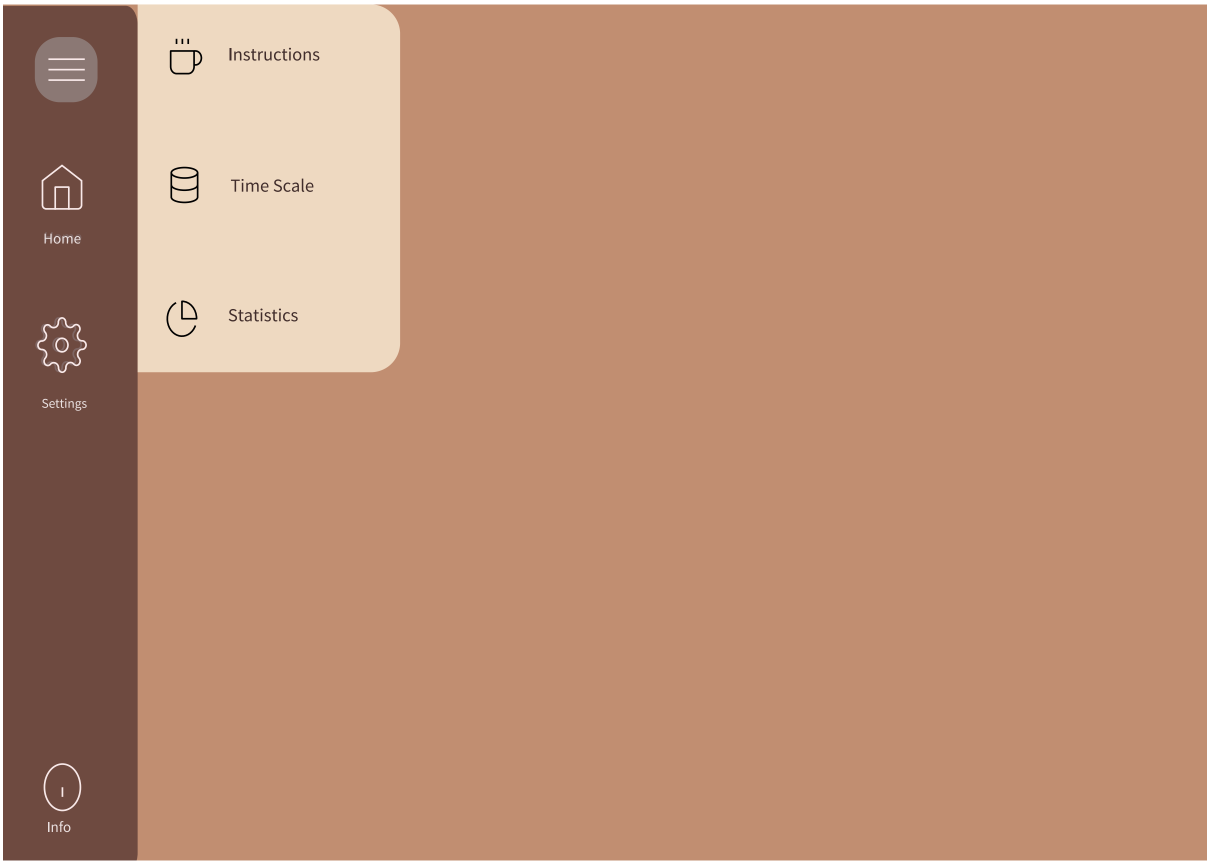
[6] For the first iteration we had help from Vinaj from Deeptech Hydraulics. Further into the development we had to improve on the design so we could mount the RPI in a casing on the screen. This time Ole Martin from Power Clamp Kongsberg helped with consulting, creating the 3D-drawings and performing the 3D-printing of the final design.

GUI Visualization

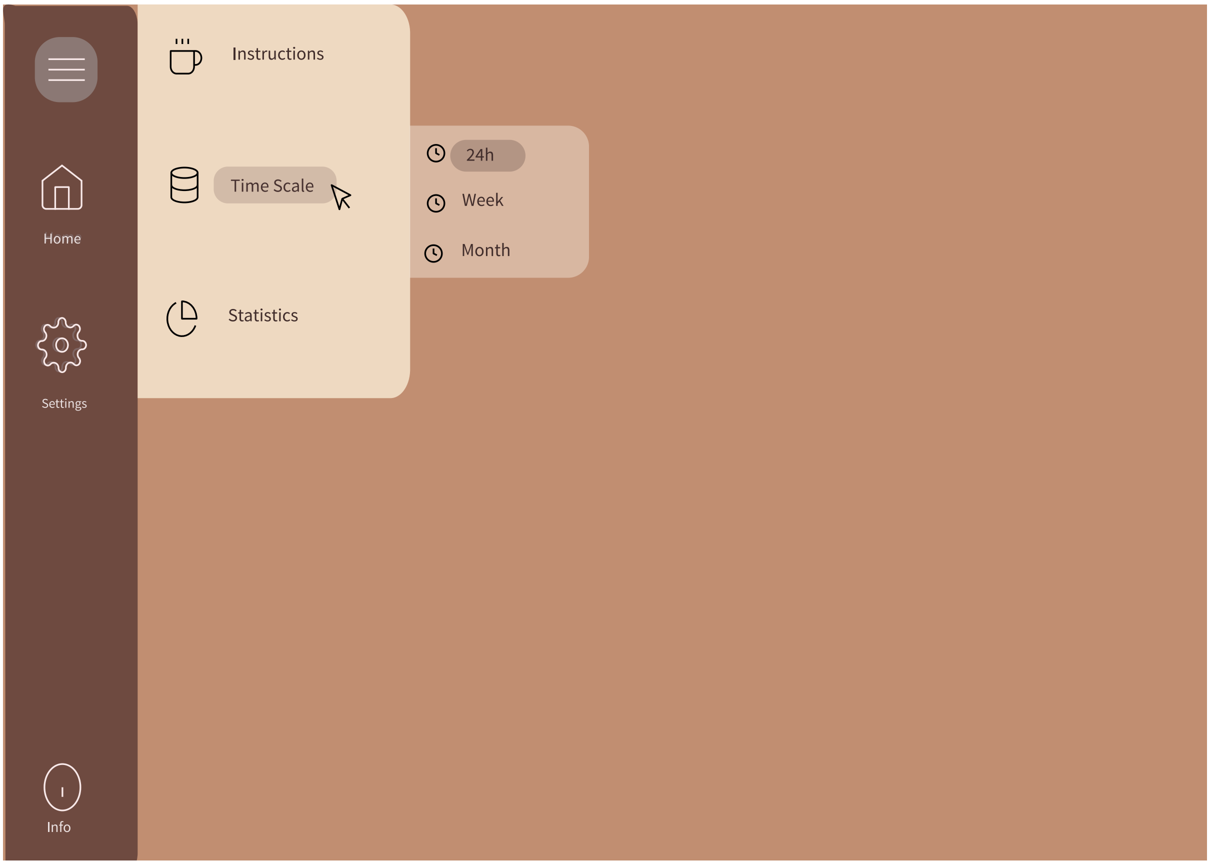
SC |

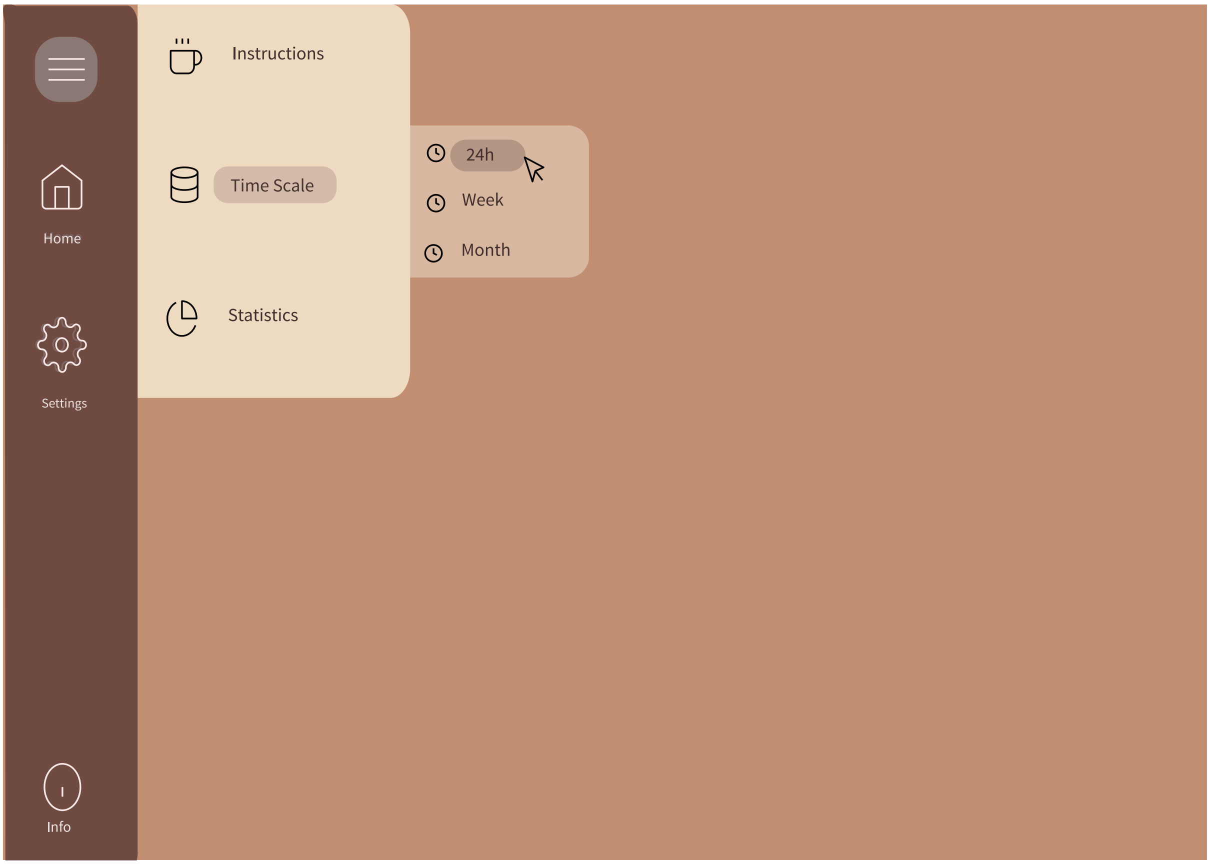

Home
Settings
Info
Timescale
Velkommen! Kaffeknekt er klar til brygging.
Oppsummering av tidligere data(i form av en graf) -skal være i display 24/7?
WT
t
Live data vises under brygging i form en graf/elr lignende.
Brygging pågår... viser sanntidsdata. (for eksempel)
semcon
WEBSITE

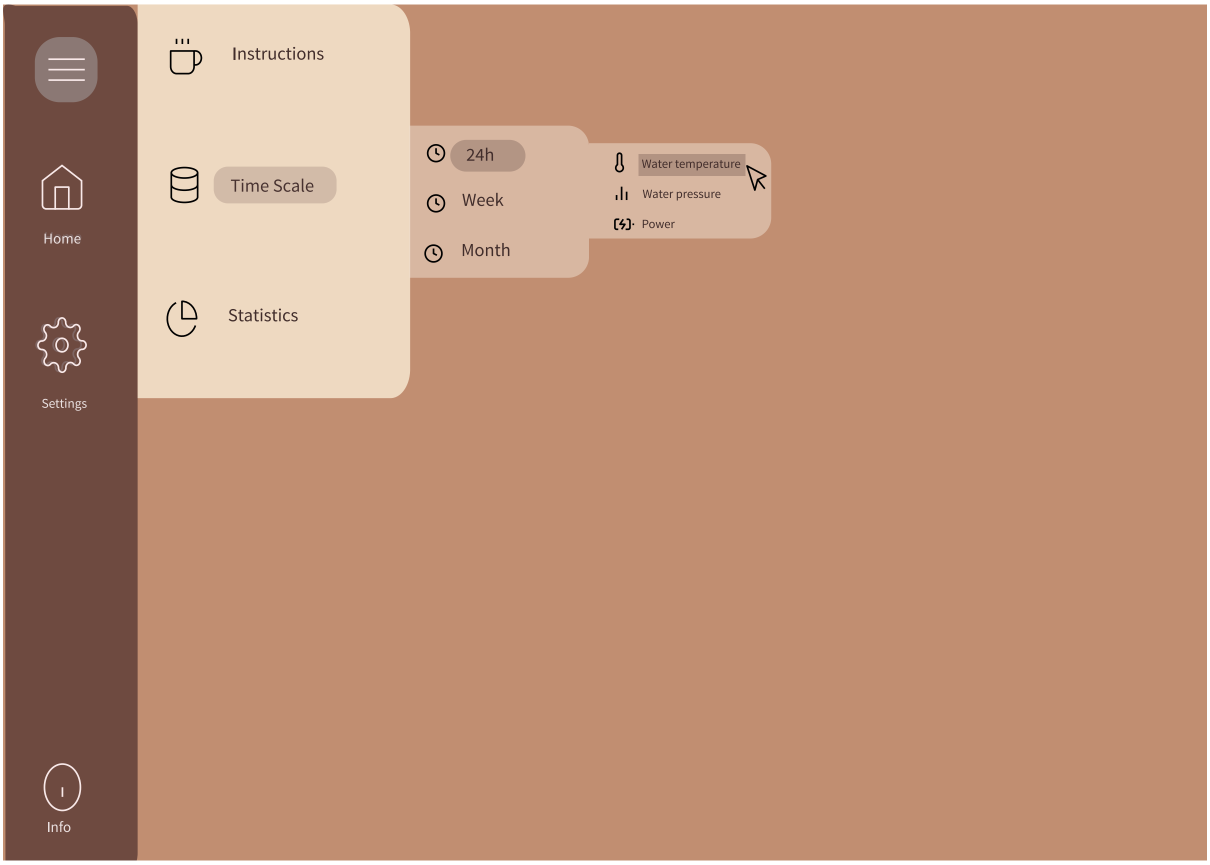

Home
Settings
Info

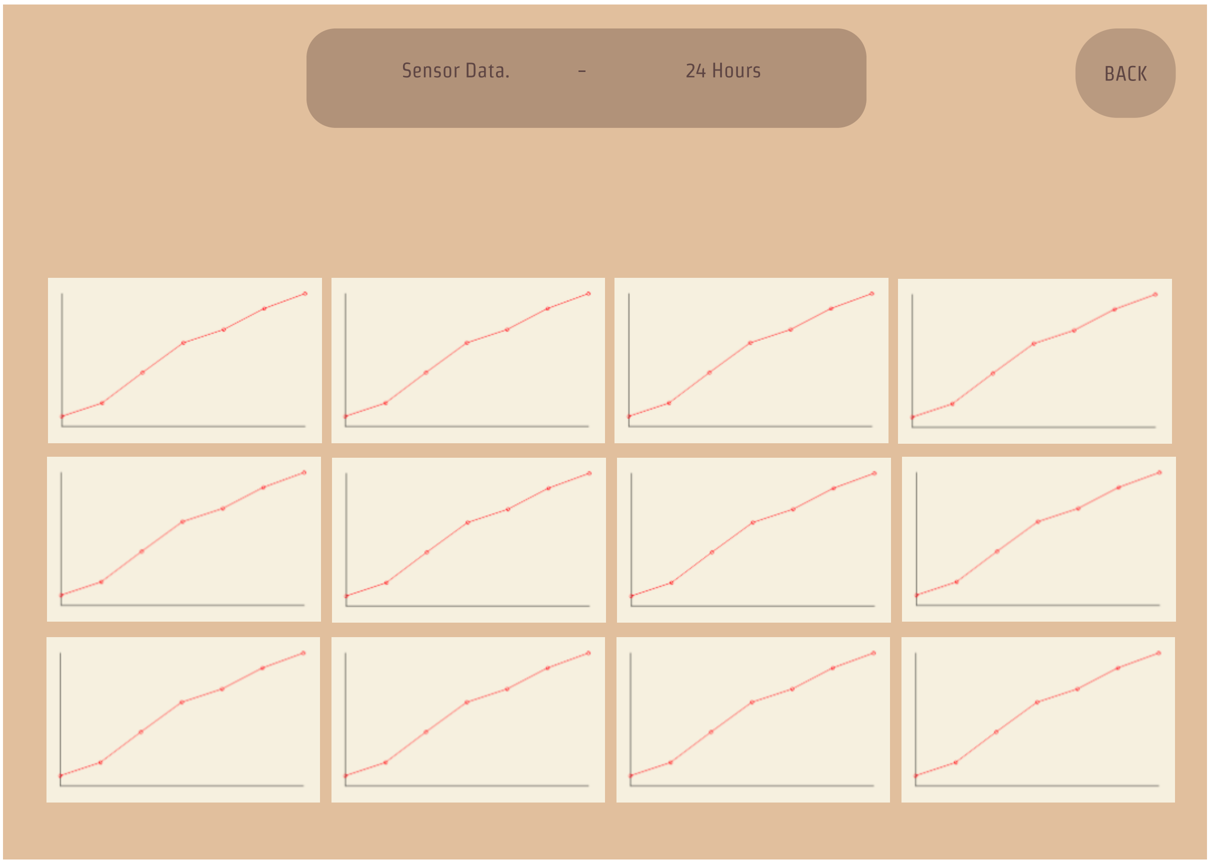

Instructions
Time Scale
Statistics
Home
Settings
Info


Instructions
Time Scale
Statistics
Home
Settings
Info


Instructions
Time Scale
24h
Week
Month
Home
Statistics
Settings
Info


Instructions
Time Scale
24h
Week
Month
Home
Statistics
Settings
Info


Instructions
Time Scale
Statistics
Home
Settings
Info
24h
Week
Month
Water temperature
Water pressure
Power


Sensor Data. - 24 Hours
BACK


Home
Settings
Info


Home
Settings
Info
Guide
The Kaffeknekt GUI Is An Intuitive Application Designed To Visualize And Display Real-Time Data From This Faema Espresso Machine, While Also Providing Access To Previously Recorded Brewing Data. This Interface Helps Users Monitor Key Brewing Parameters To Ensure Optimal Coffee Quality, Whether Brewing Live Or Reviewing Past Performance.
For Detailed Instructions On Using This App, Click Here

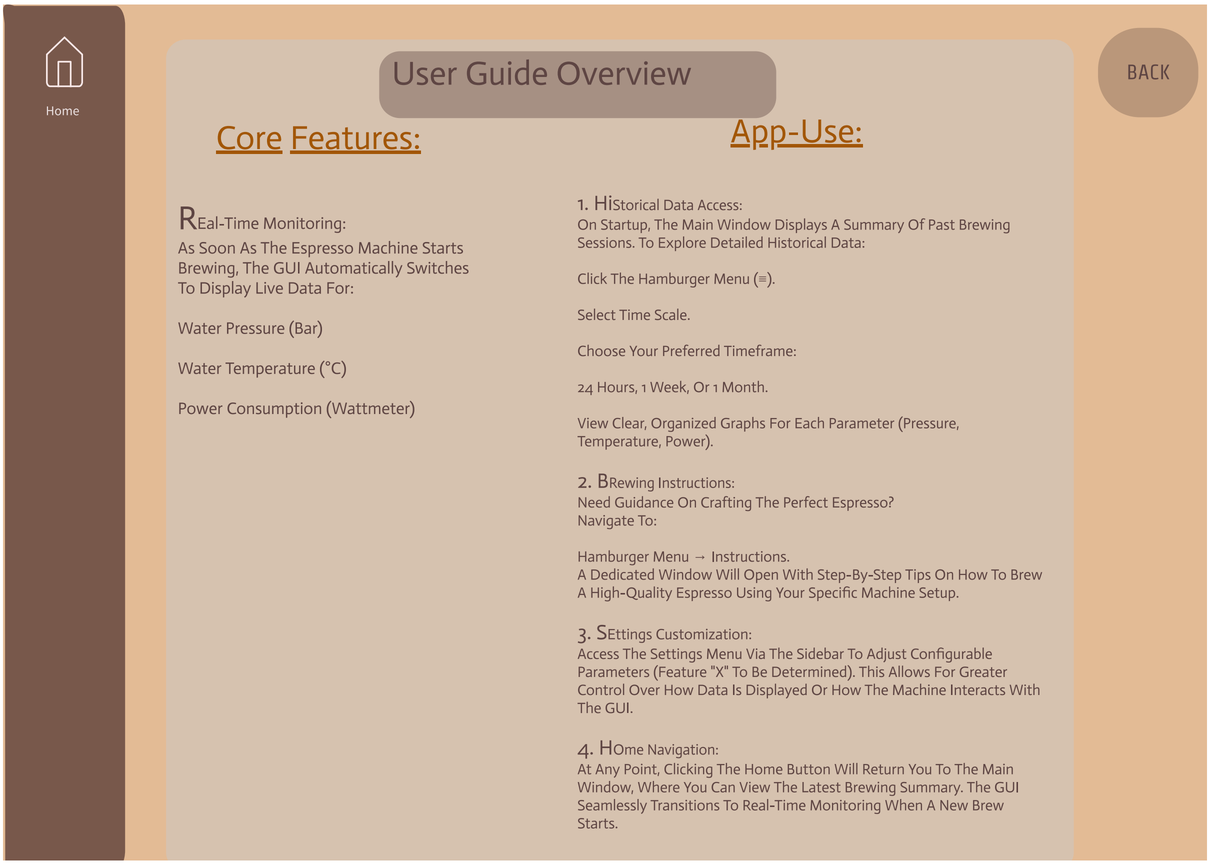

Home
User Guide Overview
BACK
Core Features:
REal-Time Monitoring:
As Soon As The Espresso Machine Starts Brewing, The GUI Automatically Switches To Display Live Data For:
Water Pressure (Bar)
Water Temperature (°C)
Power Consumption (Wattmeter)
App-Use:
1. HIstorical Data Access:
On Startup, The Main Window Displays A Summary Of Past Brewing Sessions. To Explore Detailed Historical Data:
Click The Hamburger Menu (≡).
Select Time Scale.
Choose Your Preferred Timeframe:
24 Hours, 1 Week, Or 1 Month.
View Clear, Organized Graphs For Each Parameter (Pressure, Temperature, Power).
2. BRewing Instructions:
Need Guidance On Crafting The Perfect Espresso?
Navigate To:
Hamburger Menu → Instructions.
A Dedicated Window Will Open With Step-By-Step Tips On How To Brew A High-Quality Espresso Using Your Specific Machine Setup.
3. SEttings Customization:
Access The Settings Menu Via The Sidebar To Adjust Configurable Parameters (Feature "X" To Be Determined). This Allows For Greater Control Over How Data Is Displayed Or How The Machine Interacts With The GUI.
4. HOme Navigation:
At Any Point, Clicking The Home Button Will Return You To The Main Window, Where You Can View The Latest Brewing Summary. The GUI Seamlessly Transitions To Real-Time Monitoring When A New Brew Starts.

Figure B.8.1: Initial real-time plot of ESP32 sensor data displayed in the GUI.

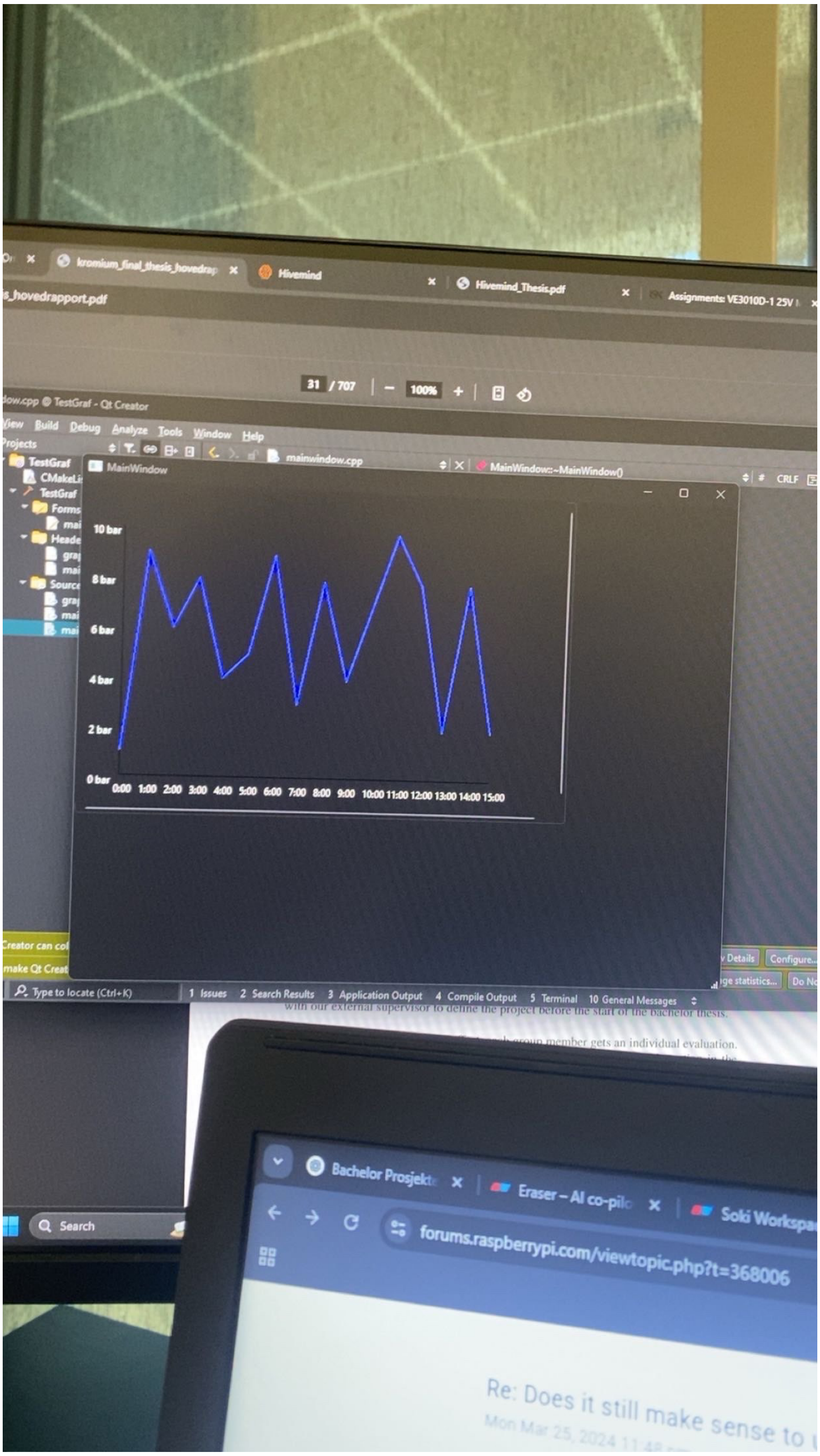


Figure B.8.2: Plot of a Test Graph Generating Fake Random Values

System Diagrams

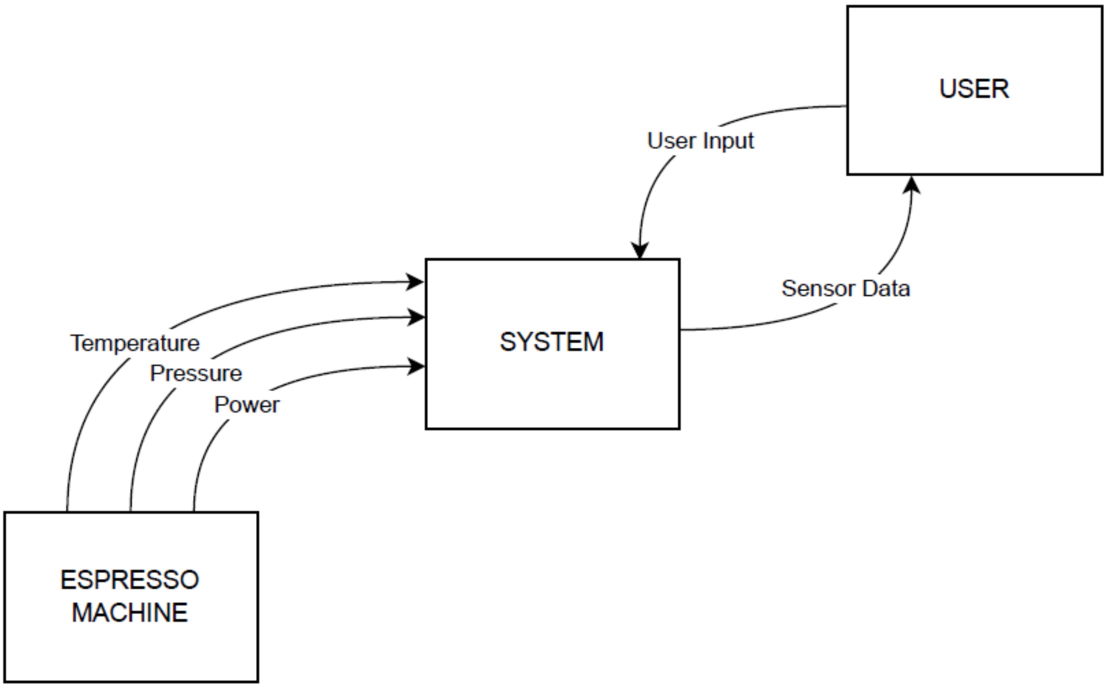


Figure B.9.1: System Context Diagram

Activity Diagram: Event is Saved and Displayed

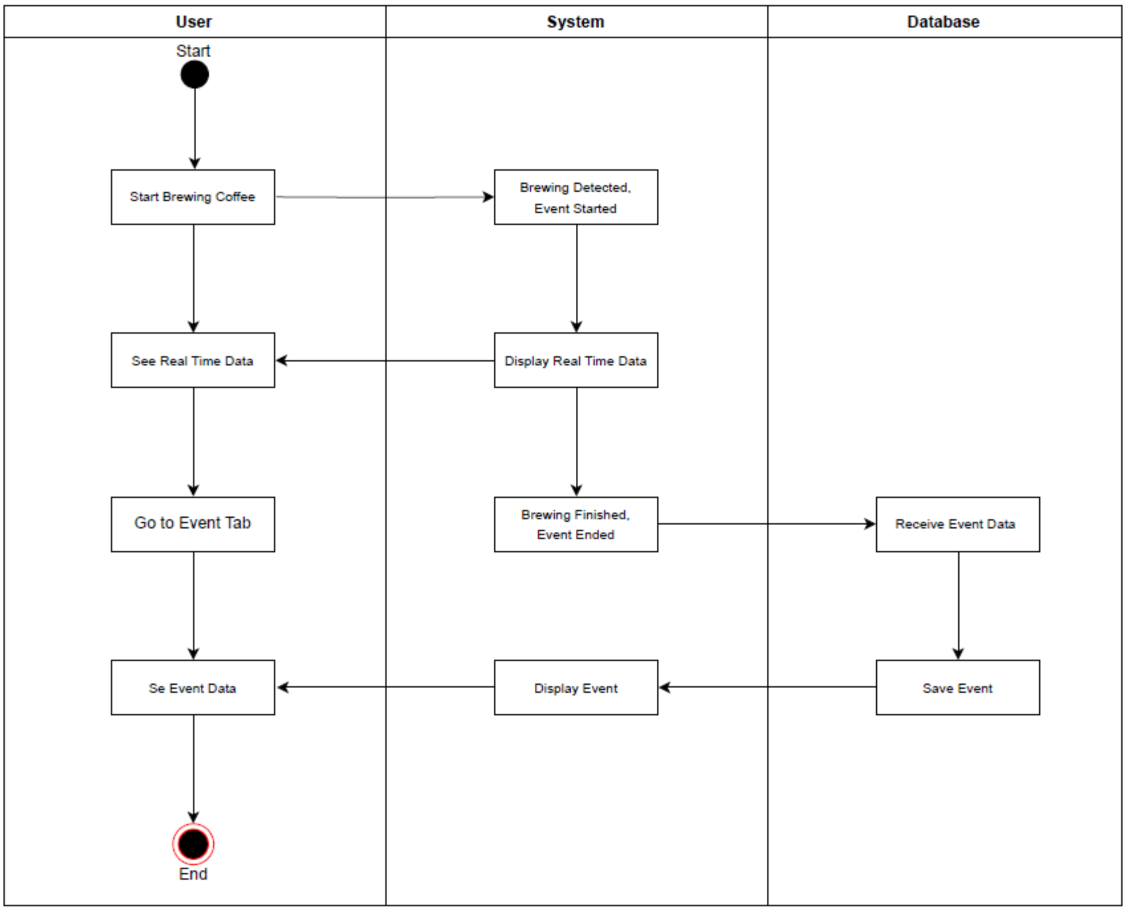


Figure B.9.2: Activity Diagram of Event Handling

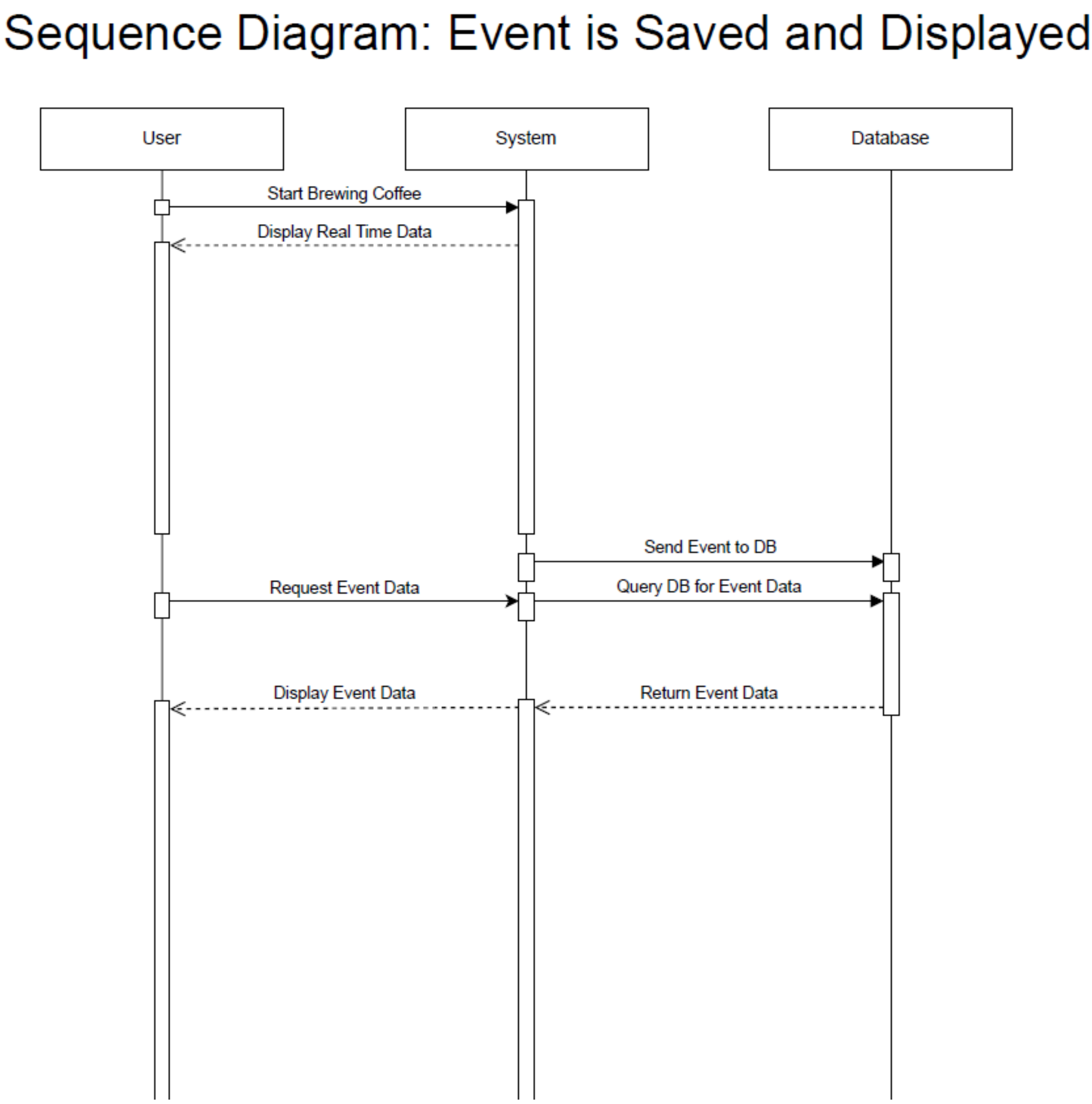


Figure B.9.3: Sequence Diagram of Event Handling

Appendix C

Code Documentation

ESP32 SOURCE FILES

Generated by Doxygen 1.13.2

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 Measurement Struct Reference

```
#include <TaskShared.h>
```

Public Attributes

- uint64_t timestamp
- int flag
- float pressure
- float temperature
- float ACPower

3.1.1 Detailed Description

Definition at line 12 of file TaskShared.h.

3.1.2 Member Data Documentation

3.1.2.1 ACPower

```
float Measurement::ACPower
```

Definition at line 17 of file TaskShared.h.

3.1.2.2 flag

```
int Measurement::flag
```

Definition at line 14 of file TaskShared.h.

3.1.2.3 pressure

```
float Measurement::pressure
```

Definition at line 15 of file TaskShared.h.

3.1.2.4 temperature

```
float Measurement::temperature
```

Definition at line 16 of file TaskShared.h.

3.1.2.5 timestamp

```
uint64_t Measurement::timestamp
```

Definition at line 13 of file TaskShared.h.

The documentation for this struct was generated from the following file:

- include/TaskShared.h

Chapter 4

File Documentation

4.1 include/TaskACPower.h File Reference

Handles AC power monitoring.

```
#include <Arduino.h>
#include <Wire.h>
#include <DFRobot_ADS1115.h>
```

Functions

- void TaskACPower (void *pvParameters)

  FreeRTOS task that periodically reads AC current and updates power measurements.

4.1.1 Detailed Description

Handles AC power monitoring.

Detailed description here.

Definition in file TaskACPower.h.

4.1.2 Function Documentation

4.1.2.1 TaskACPower()

```
void TaskACPower (
            void * pvParameters)
```

FreeRTOS task that periodically reads AC current and updates power measurements.

Uses the SEN0211 sensor and ADS1115 ADC to read current from two input channels. One channel is used to compute the power (ACPower), and the other sets a flag based on whether the current exceeds 0.06A.

Parameters

pvParameters	Unused parameter required by FreeRTOS.

Definition at line 104 of file TaskACPower.cpp.

```
00104                                     {
00105   (void)pvParameters;
00106
00107   // // Initial delay to allow system to stabilize
00108   vTaskDelay(2000 / portTICK_PERIOD_MS);
00109
00110
00111   while (true) {
00112     float ACPower = 300.0;
00113     float current = readACCurrentValue(0);
00114     ACPower = current * VOLTAGE_REFERENCE;
00115
00116
00117
00118       sharedMeasurement.ACPower = ACPower;
00119
00120
00121     current = readACCurrentValue(1);
00122
00123
00124     if(current > 0.06){
00125       sharedMeasurement.flag = 1;
00126     } else {
00127       sharedMeasurement.flag = 0;
00128     }
00129
00130
00131
00132
00133
00134     // Wait before next reading (2 seconds)
00135     vTaskDelay(40 / portTICK_PERIOD_MS);
00136   }
00137 }
```

4.2 TaskACPower.h

Go to the documentation of this file.

```


#pragma once
#include <Arduino.h>
#include <Wire.h>
#include <DFRobot_ADS1115.h>

void TaskACPower(void *pvParameters);
```

4.3 include/TaskCalculateTime.h File Reference

Calculates time.

```
#include <Arduino.h>
```

Functions

- void TaskCalculateTime (void *pvParameters)

  Shared structure containing sensor measurements including timestamp.

4.3.1 Detailed Description

Calculates time.

Detailed description here.

Definition in file TaskCalculateTime.h.

4.3.2 Function Documentation

4.3.2.1 TaskCalculateTime()

```
void TaskCalculateTime (
              void * pvParameters)
```

Shared structure containing sensor measurements including timestamp.

FreeRTOS task to calculate the current UNIX timestamp from the RTC and store it.

Periodically reads the date and time from the RTC via I^2C using a mutex to ensure safe access. The retrieved time is converted to a UNIX timestamp in nanoseconds using `mktime()` and stored in `sharedMeasurement.↵ timestamp`.

Parameters

pvParameters	Unused parameter required by FreeRTOS task signature.

Definition at line 29 of file TaskCalculateTime.cpp.

```
00029                                           {
00030   (void)pvParameters;
00031
00032   while (true) {
00033     uint16_t year;
00034     uint8_t month, date, hour, minute, second;
00035
00036
00037
00038
00039       if (xSemaphoreTake(i2cMutex, portMAX_DELAY) == pdTRUE) {
00040         year   = rtc.getYear();
00041         month  = rtc.getMonth();
00042         date   = rtc.getDate();
00043         hour   = rtc.getHour();
00044         minute = rtc.getMinute();
00045         second = rtc.getSecond();
00046         xSemaphoreGive(i2cMutex);
00047       }
00048
00049
00050
00051     struct tm timeinfo;
00052     timeinfo.tm_year = year - 1900;
00053     timeinfo.tm_mon  = month - 1;
00054     timeinfo.tm_mday = date;
00055     timeinfo.tm_hour = hour;
00056     timeinfo.tm_min  = minute;
00057     timeinfo.tm_sec  = second;
00058     timeinfo.tm_isdst = 0;
00059
00060     time_t unix_seconds = mktime(&timeinfo);
00061     uint64_t unix_nanos = (uint64_t)unix_seconds * 1000000000ULL;
00062
00063
00064       sharedMeasurement.timestamp = unix_nanos;
00065
00066
00067     vTaskDelay(200 / portTICK_PERIOD_MS);
00068   }
00069 }
```

4.4 TaskCalculateTime.h

Go to the documentation of this file.

```
#pragma once
#include <Arduino.h>

void TaskCalculateTime(void *pvParameters);
```

4.5 include/TaskPressure.h File Reference

Handles pressure monitoring.

```
#include <Arduino.h>
#include <Wire.h>
```

Functions

- void TaskPressure (void *pvParameters)

  Mutex for guarding I^{2}C access across tasks.

4.5.1 Detailed Description

Handles pressure monitoring.

Detailed description here.

Definition in file TaskPressure.h.

4.5.2 Function Documentation

4.5.2.1 TaskPressure()

```
void TaskPressure (
            void * pvParameters)
```

Mutex for guarding I^{2}C access across tasks.

FreeRTOS task that reads pressure data from a sensor and updates the shared measurement.

This task waits 1 second before starting, then enters a loop where it:

- Takes the I^{2}C mutex.
- Requests 4 bytes of data from a digital pressure sensor at address 0x28.
- Parses the 14-bit raw pressure reading.
- Converts it to PSI, then to bar.
- Releases the mutex and updates the global `sharedMeasurement.pressure` value.

If the sensor doesn't respond properly, an error message is printed over Serial.

Parameters

pvParameters	Unused parameter required by FreeRTOS task signature.

Definition at line 37 of file TaskPressure.cpp.

```
00037                                    {
00038   (void)pvParameters;
00039
00040   vTaskDelay(1000 / portTICK_PERIOD_MS);
00041
00042   while (true) {
00043     float pressure = 0.0;
00044
00045     if (xSemaphoreTake(i2cMutex, portMAX_DELAY) == pdTRUE) {
00046
00047
00048
00049     // Request data from the pressure sensor
00050     Wire.requestFrom(SENSOR_ADDR, (uint8_t)4);
00051
00052     if (Wire.available() == 4) {
00053       uint8_t buffer[4];
00054       for (int i = 0; i < 4; ++i) {
00055         buffer[i] = Wire.read();
00056       }
00057
00058       // Extract 14-bit raw pressure
00059       uint16_t raw_pressure = ((buffer[0] & 0x3F) << 8) | buffer[1];
00060
00061       // Convert to pressure
00062       float pressure_psi = ((float)(raw_pressure - OUTPUT_MIN) / (OUTPUT_MAX - OUTPUT_MIN)) *
      PRESSURE_MAX_PSI;
00063       pressure = pressure_psi * PSI_TO_BAR;
00064
00065
00066     } else {
00067       Serial.println("Failed to read pressure sensor data");
00068     }
00069     xSemaphoreGive(i2cMutex);
00070 }
00071
00072       sharedMeasurement.pressure = pressure;
00073
00074
00075     vTaskDelay(pdMS_TO_TICKS(2000));
00076   }
00077 }
```

4.6 TaskPressure.h

Go to the documentation of this file.

```
#pragma once
#include <Arduino.h>
#include <Wire.h>

void TaskPressure(void *pvParameters);
```

4.7 include/TaskPublish.h File Reference

Publishes sensor data over usb.

```
#include <Arduino.h>
```

Functions

- void TaskPublish (void ∗pvParameters)

    FreeRTOS task that publishes sensor data as JSON over Serial.

4.7.1 Detailed Description

Publishes sensor data over usb.

Detailed description here.

Definition in file TaskPublish.h.

4.7.2 Function Documentation

4.7.2.1 TaskPublish()

```
void TaskPublish (
            void * pvParameters)
```

FreeRTOS task that publishes sensor data as JSON over Serial.

Every 300 ms, this task creates a JSON object containing pressure, temperature, power, timestamp, and a status flag from the shared measurement structure. It prints the JSON to the Serial port for external logging or monitoring.

Parameters

pvParameters	Unused parameter required by FreeRTOS task signature.

Definition at line 25 of file TaskPublish.cpp.

```
00025                                     {
00026   (void)pvParameters;
00027
00028   while (true) {
00029     Measurement localCopy;
00030
00031
00032       localCopy = sharedMeasurement;
00033
00034
00035     StaticJsonDocument<128> doc;
00036     doc["pressure"] = localCopy.pressure;
00037     doc["temperature"] = localCopy.temperature;
00038     doc["power"] = localCopy.ACPower;
00039     doc["timestamp"] = localCopy.timestamp;
00040     doc["flag"] = localCopy.flag;
00041     serializeJson(doc, Serial);
00042     Serial.println();
00043
00044     vTaskDelay(pdMS_TO_TICKS(300));
00045   }
00046 }
```

4.8 TaskPublish.h

Go to the documentation of this file.

```


#pragma once
#include <Arduino.h>

void TaskPublish(void *pvParameters);
```

4.9 include/TaskReceiveTime.h File Reference

Handles incoming signal to adjust the clock.

```
#include <Arduino.h>
```

Functions

- void TaskReceiveTime (void *pvParameters)

    FreeRTOS task to receive and process time-setting commands over Serial.

4.9.1 Detailed Description

Handles incoming signal to adjust the clock.

Detailed description here.

Definition in file TaskReceiveTime.h.

4.9.2 Function Documentation

4.9.2.1 TaskReceiveTime()

```
void TaskReceiveTime (
            void * pvParameters)
```

FreeRTOS task to receive and process time-setting commands over Serial.

Expects a JSON payload such as:

```
{
  "set_time": true,
  "year": 2025,
  "month": 5,
  "day": 8,
  "hour": 14,
  "minute": 30,
  "second": 0
}
```

If valid, it sets the RTC accordingly and replies with a success message.

Parameters

pvParameters	Unused parameter required by FreeRTOS task signature.

Definition at line 36 of file TaskReceiveTime.cpp.

```
00036                                         {
00037   (void)pvParameters;
00038
00039   String input;
00040
00041   while (true) {
00042     if (Serial.available()) {
00043       input = Serial.readStringUntil('\n');
00044
00045       StaticJsonDocument<200> doc;
00046       DeserializationError err = deserializeJson(doc, input);
00047
00048       if (err) {
00049         Serial.println("{\"error\":\"Invalid JSON\"}");
00050         continue;
00051       }
00052
00053       if (doc["set_time"] == true &&
00054           doc.containsKey("year") &&
00055           doc.containsKey("month") &&
00056           doc.containsKey("day") &&
00057           doc.containsKey("hour") &&
00058           doc.containsKey("minute") &&
00059           doc.containsKey("second")) {
00060
00061         int year = doc["year"].as<int>();
00062         int month = doc["month"].as<int>();
00063         int day = doc["day"].as<int>();
00064         int hour = doc["hour"].as<int>();
00065         int minute = doc["minute"].as<int>();
00066         int second = doc["second"].as<int>();
00067
00068         rtc.setTime(year, month, day, hour, minute, second);
00069
00070         Serial.println("{\"status\":\"RTC updated\"}");
00071       }
00072     }
00073
00074     vTaskDelay(pdMS_TO_TICKS(100));
00075   }
00076 }
```

4.10 TaskReceiveTime.h

Go to the documentation of this file.

```


#pragma once
#include <Arduino.h>

void TaskReceiveTime(void *pvParameters);
```

4.11 include/TaskShared.h File Reference

Structure of sensor data.

```
#include <Arduino.h>
```

Classes

- struct Measurement

Variables

- Measurement sharedMeasurement
- SemaphoreHandle_t measurementMutex

4.11.1 Detailed Description

Structure of sensor data.

Detailed description here.

Definition in file TaskShared.h.

4.11.2 Variable Documentation

4.11.2.1 measurementMutex

```
SemaphoreHandle_t measurementMutex [extern]
```

4.11.2.2 sharedMeasurement

```
Measurement sharedMeasurement [extern]
```

Definition at line 28 of file main.cpp.

```
{};
```

4.12 TaskShared.h

Go to the documentation of this file.

```


#pragma once
#include <Arduino.h>

// Shared data structure
struct Measurement {
  uint64_t timestamp;
  int flag;
  float pressure;
  float temperature;
  float ACPower;
};

// Global shared instance (defined in main.cpp)
extern Measurement sharedMeasurement;

// Mutex to protect shared access to the Measurements
extern SemaphoreHandle_t measurementMutex;
```

4.13 include/TaskTemperature.h File Reference

Handles temperature monitoring.

```
#include <Arduino.h>
```

Functions

- void TaskTemperature (void *pvParameters)
  FreeRTOS task that reads temperature sensor data and processes it.

4.13.1 Detailed Description

Handles temperature monitoring.

Detailed description here.

Definition in file TaskTemperature.h.

4.13.2 Function Documentation

4.13.2.1 TaskTemperature()

```
void TaskTemperature (
            void * pvParameters)
```

FreeRTOS task that reads temperature sensor data and processes it.

Parameters

pvParameters	Pointer to parameters passed to the task (unused).

FreeRTOS task that reads temperature sensor data and processes it.

Periodically reads the RTD value, computes the temperature, and stores it in a shared structure.

Parameters

pvParameters	Unused parameter required by FreeRTOS task signature.

Initial delay to allow other initializations to complete

SPI initialization for temperature sensor

Initialize the MAX31865 with 2WIRE configuration

Initial reading to check if sensor is working

Task loop

Read raw RTD value

Default value

Only process if we got a valid reading

Calculate temperature using the library function

Check for out-of-range values

Check for sensor faults

Clear fault

Try to reinitialize the sensor

Definition at line 37 of file TaskTemperature.cpp.

```
00037                                             {
00038    (void)pvParameters;
00039
00041    vTaskDelay(2000 / portTICK_PERIOD_MS);
00042
00043    Serial.println("Temperature sensor task starting...");
00044
00046    SPI.begin(); //redundant?
00047
00049    thermo.begin(MAX31865_2WIRE);
00050
00052    uint16_t rtd_initial = thermo.readRTD();
00053    Serial.print("Initial RTD reading: ");
00054    Serial.println(rtd_initial);
00055
00056    if (rtd_initial == 0) {
00057      Serial.println("WARNING: Temperature sensor not responding, check connections");
00058    }
00059
00061    while (true) {
00063      uint16_t rtd = thermo.readRTD();
00064
00065      float temperature = 22.0;
00066
00068      if (rtd > 0) {
00070        temperature = thermo.temperature(RNOMINAL, RREF) - 7.0;
00071
00072
00073        float ratio = rtd;
00074        ratio /= 32768;
00075        float resistance = RREF * ratio;
00076
00077
00079        if (temperature < -50 || temperature > 200) {
00080          Serial.println("Temperature out of reasonable range, using default");
00081          temperature = 22.0;
00082        }
00083      } else {
00084        Serial.println("Invalid RTD reading, using default temperature");
00085      }
00086
00088      uint8_t fault = thermo.readFault();
00089      if (fault) {
00090        Serial.print("Fault detected 0x");
00091        Serial.println(fault, HEX);
00092
00094        thermo.clearFault();
00095
00097        thermo.begin(MAX31865_2WIRE);
00098        vTaskDelay(100 / portTICK_PERIOD_MS);
00099      }
00100
00101        sharedMeasurement.temperature = temperature;
00102
00103      vTaskDelay(2000 / portTICK_PERIOD_MS);
00104    }
00105 }
```

4.14 TaskTemperature.h

Go to the documentation of this file.

```

#pragma once
#include <Arduino.h>

void TaskTemperature(void *pvParameters);
```

4.15 src/main.cpp File Reference

Main entry point for the FreeRTOS system. Initializes hardware and starts tasks.

```
#include <Arduino.h>
#include <Wire.h>
#include "DFRobot_DS323X.h"
#include "TaskShared.h"
#include "TaskACPower.h"
#include "TaskPublish.h"
#include "TaskCalculateTime.h"
#include "TaskReceiveTime.h"
#include "TaskTemperature.h"
#include "TaskPressure.h"
```

Macros

- #define SDA_PIN 21
- #define SCL_PIN 22

Functions

- void setup ()

  Arduino setup function. Initializes peripherals and starts FreeRTOS tasks.
- void loop ()

  Arduino loop function. Left empty since all functionality is handled by tasks.

Variables

- DFRobot_DS323X rtc

  Mutex for guarding I^2C access across tasks.
- DFRobot_ADS1115 ads & Wire
- Measurement sharedMeasurement = {}

  RTC module used to get the current time.
- SemaphoreHandle_t i2cMutex

4.15.1 Detailed Description

Main entry point for the FreeRTOS system. Initializes hardware and starts tasks.

Sets up I^2C, ADC (ADS1115), and RTC (DS3232). Creates and pins various sensor-reading and data-publishing FreeRTOS tasks to core 1. A shared measurement structure and an I^2C mutex are used for safe task communication and peripheral access.

Definition in file main.cpp.

4.15.2 Macro Definition Documentation

4.15.2.1 SCL_PIN

```
#define SCL_PIN 22
```

Definition at line 23 of file main.cpp.

4.15.2.2 SDA_PIN

```
#define SDA_PIN 21
```

Definition at line 22 of file main.cpp.

4.15.3 Function Documentation

4.15.3.1 loop()

```
void loop ()
```

Arduino loop function. Left empty since all functionality is handled by tasks.

Definition at line 70 of file main.cpp.

```
00070             {
00071   // Tasks do the work
00072 }
```

4.15.3.2 setup()

```
void setup ()
```

Arduino setup function. Initializes peripherals and starts FreeRTOS tasks.

Definition at line 34 of file main.cpp.

```
00034              {
00035   Serial.begin(115200);
00036   Wire.begin(SDA_PIN, SCL_PIN);
00037 
00038   // SPI.begin();
00039 
00040   ads.setAddr_ADS1115(ADS1115_IIC_ADDRESS0);  // 0x48
00041   ads.setGain(eGAIN_ONE);          // ±4.096V range
00042   ads.setMode(eMODE_SINGLE);       // Single-shot mode
00043   ads.setRate(eRATE_128);          // 128 samples per second
00044   ads.setOSMode(eOSMODE_SINGLE); // Start a single conversion
00045   ads.init();
00046 
00047 
00048   while (!rtc.begin()) {
00049     Serial.println("Failed to init DS3232 RTC chip.");
00050     delay(1000);
00051   }
00052   Serial.println("DS3232 RTC initialized successfully!");
00053 
00054   i2cMutex = xSemaphoreCreateMutex();
00055 
00056 
00057   xTaskCreatePinnedToCore(TaskCalculateTime, "CalculateTime", 4096, NULL, 1, NULL, 1);
00058   xTaskCreatePinnedToCore(TaskPressure, "Pressure", 4096, NULL, 1, NULL, 1);
00059   xTaskCreatePinnedToCore(TaskTemperature, "Temperature", 4096, NULL, 1, NULL, 1);
00060   xTaskCreatePinnedToCore(TaskACPower, "ACPower", 4096, NULL, 2, NULL, 1);
00061   xTaskCreatePinnedToCore(TaskPublish, "Publish", 4096, NULL, 1, NULL, 1);
00062   //xTaskCreatePinnedToCore(TaskReceiveTime, "RecieveTime", 4096, NULL, 1, NULL, 1);
00063   //xTaskCreatePinnedToCore(TaskBrewTrigger, "BrewTrigger", 4096, NULL, 2, NULL, 1);
00064 
00065 }
```

4.15.4 Variable Documentation

4.15.4.1 i2cMutex

```
SemaphoreHandle_t i2cMutex
```

Definition at line 29 of file main.cpp.

4.15.4.2 rtc

```
DFRobot_DS323X rtc
```

Mutex for guarding I^2C access across tasks.

Global RTC object used to set the time.

Definition at line 25 of file main.cpp.

4.15.4.3 sharedMeasurement

```
Measurement sharedMeasurement = {}
```

RTC module used to get the current time.

Shared measurement structure used across tasks.

Shared structure containing the latest sensor measurements.

Definition at line 28 of file main.cpp.

```
00028 {};
```

4.15.4.4 Wire

```
DFRobot_ADS1115 ads& Wire
```

Definition at line 26 of file main.cpp.

4.16 main.cpp

Go to the documentation of this file.

```cpp


#include <Arduino.h>
#include <Wire.h>
#include "DFRobot_DS323X.h"
#include "TaskShared.h"
#include "TaskACPower.h"
#include "TaskPublish.h"
#include "TaskCalculateTime.h"
#include "TaskReceiveTime.h"
#include "TaskTemperature.h"
#include "TaskPressure.h"
//#include "TaskBrewTrigger.h"

#define SDA_PIN 21
#define SCL_PIN 22

DFRobot_DS323X rtc;  // global instance
DFRobot_ADS1115 ads(&Wire);

Measurement sharedMeasurement = {};
SemaphoreHandle_t i2cMutex;

void setup() {
  Serial.begin(115200);
  Wire.begin(SDA_PIN, SCL_PIN);

  // SPI.begin();

  ads.setAddr_ADS1115(ADS1115_IIC_ADDRESS0);  // 0x48
  ads.setGain(eGAIN_ONE);         // ±4.096V range
  ads.setMode(eMODE_SINGLE);      // Single-shot mode
  ads.setRate(eRATE_128);         // 128 samples per second
  ads.setOSMode(eOSMODE_SINGLE); // Start a single conversion
  ads.init();


  while (!rtc.begin()) {
    Serial.println("Failed to init DS3232 RTC chip.");
    delay(1000);
  }
  Serial.println("DS3232 RTC initialized successfully!");

  i2cMutex = xSemaphoreCreateMutex();


  xTaskCreatePinnedToCore(TaskCalculateTime, "CalculateTime", 4096, NULL, 1, NULL, 1);
  xTaskCreatePinnedToCore(TaskPressure, "Pressure", 4096, NULL, 1, NULL, 1);
  xTaskCreatePinnedToCore(TaskTemperature, "Temperature", 4096, NULL, 1, NULL, 1);
  xTaskCreatePinnedToCore(TaskACPower, "ACPower", 4096, NULL, 2, NULL, 1);
  xTaskCreatePinnedToCore(TaskPublish, "Publish", 4096, NULL, 1, NULL, 1);
  //xTaskCreatePinnedToCore(TaskReceiveTime, "RecieveTime", 4096, NULL, 1, NULL, 1);
  //xTaskCreatePinnedToCore(TaskBrewTrigger, "BrewTrigger", 4096, NULL, 2, NULL, 1);

}

void loop() {
  // Tasks do the work
}
```

4.17 src/TaskACPower.cpp File Reference

FreeRTOS task to measure AC current using a SEN0211 sensor and update power readings.

```cpp
#include <Arduino.h>
#include <Wire.h>
#include "TaskShared.h"
#include "TaskACPower.h"
#include "I2CLock.h"
```

Macros

- #define CURRENT_DETECTION_RANGE 20
- #define VOLTAGE_REFERENCE 240.0
- #define CALIBRATION_FACTOR 1.070

Functions

- float readACCurrentValue (int pinNumber)

  Reads the RMS AC current value from a specified ADS1115 channel.
- float calculatePower (float current)

  Calculates power in watts from current using a fixed voltage reference.
- void TaskACPower (void ∗pvParameters)

  FreeRTOS task that periodically reads AC current and updates power measurements.

Variables

- Measurement sharedMeasurement
- DFRobot_ADS1115 ads
- SemaphoreHandle_t i2cMutex
- const int CURRENT_PIN = 0
- const int SAMPLES = 10

4.17.1 Detailed Description

FreeRTOS task to measure AC current using a SEN0211 sensor and update power readings.

This task uses an ADS1115 ADC and a SEN0211 current sensor to measure the AC current, calculate power consumption, and update a shared measurement structure. It also sets a status flag based on a secondary current threshold.

Definition in file TaskACPower.cpp.

4.17.2 Macro Definition Documentation

4.17.2.1 CALIBRATION_FACTOR

```
#define CALIBRATION_FACTOR 1.070
```

Definition at line 25 of file TaskACPower.cpp.

4.17.2.2 CURRENT_DETECTION_RANGE

```
#define CURRENT_DETECTION_RANGE 20
```

Definition at line 23 of file TaskACPower.cpp.

4.17.2.3 VOLTAGE_REFERENCE

```
#define VOLTAGE_REFERENCE 240.0
```

Definition at line 24 of file TaskACPower.cpp.

4.17.3 Function Documentation

4.17.3.1 calculatePower()

```
float calculatePower (
            float current)
```

Calculates power in watts from current using a fixed voltage reference.

Parameters

current	Current in amperes.

Returns

Power in watts.

Definition at line 91 of file TaskACPower.cpp.

```
00091                                   {
00092   return VOLTAGE_REFERENCE * current;
00093 }
```

4.17.3.2 readACCurrentValue()

```
float readACCurrentValue (
            int pinNumber)
```

Reads the RMS AC current value from a specified ADS1115 channel.

Takes multiple ADC readings, calculates peak voltage, converts it to RMS voltage, then to current using the sensor's range and calibration factor.

Parameters

pinNumber	Analog input pin on ADS1115 (e.g., 0 for A0, 1 for A1).

Returns

Measured current in amperes.

Definition at line 43 of file TaskACPower.cpp.

```
00043                                            {
00044     float peakVoltage = 0;
00045     float voltageRMS = 0;
00046     float currentValue = 0;
00047 
00048     // Take multiple samples for more stable reading
00049     const int SAMPLES = 10;
00050 
00051     if (xSemaphoreTake(i2cMutex, portMAX_DELAY) == pdTRUE) {
00052       for (int i = 0; i < SAMPLES; i++) {
00053         int rawReading = ads.readVoltage(pinNumber); // Reading in mV
00054         peakVoltage += rawReading;
00055         vTaskDelay(2 / portTICK_PERIOD_MS);
00056   ; // Small delay between readings
00057       }
00058       xSemaphoreGive(i2cMutex);
00059     }
00060 
00061 
00062 
00063     // Average the readings
00064     peakVoltage = peakVoltage / SAMPLES;
00065 
00066     // Convert peak to RMS (root mean square) - using 0.707 factor
00067     voltageRMS = peakVoltage * 0.707;
00068 
00069     // Convert the ADC millivolts to actual volts
00070     voltageRMS = voltageRMS / 1000.0; // Convert mV to V
00071 
00072     // Account for circuit amplification
00073     voltageRMS = voltageRMS / 2;
00074 
00075     // Convert voltage to current using detection range
00076     currentValue = voltageRMS * CURRENT_DETECTION_RANGE;
00077 
00078     // Apply calibration factor to correct the reading
00079     currentValue = currentValue * CALIBRATION_FACTOR;
00080 
00081     return currentValue;
00082   }
```

4.17.3.3 TaskACPower()

```
void TaskACPower (
            void * pvParameters)
```

FreeRTOS task that periodically reads AC current and updates power measurements.

Uses the SEN0211 sensor and ADS1115 ADC to read current from two input channels. One channel is used to compute the power (ACPower), and the other sets a flag based on whether the current exceeds 0.06A.

Parameters

pvParameters	Unused parameter required by FreeRTOS.

Definition at line 104 of file TaskACPower.cpp.

```
00104                                       {
00105   (void)pvParameters;
00106 
00107   // // Initial delay to allow system to stabilize
00108   vTaskDelay(2000 / portTICK_PERIOD_MS);
00109 
00110 
00111   while (true) {
00112     float ACPower = 300.0;
00113     float current = readACCurrentValue(0);
00114     ACPower = current * VOLTAGE_REFERENCE;
```

```



      sharedMeasurement.ACPower = ACPower;


    current = readACCurrentValue(1);


    if(current > 0.06){
      sharedMeasurement.flag = 1;
    } else {
      sharedMeasurement.flag = 0;
    }





    // Wait before next reading (2 seconds)
    vTaskDelay(40 / portTICK_PERIOD_MS);
  }
}
```

4.17.4 Variable Documentation

4.17.4.1 ads

```
DFRobot_ADS1115 ads [extern]
```

4.17.4.2 CURRENT_PIN

```
const int CURRENT_PIN = 0
```

Definition at line 28 of file TaskACPower.cpp.

4.17.4.3 i2cMutex

```
SemaphoreHandle_t i2cMutex [extern]
```

Definition at line 29 of file main.cpp.

4.17.4.4 SAMPLES

```
const int SAMPLES = 10
```

Definition at line 31 of file TaskACPower.cpp.

4.17.4.5 sharedMeasurement

```
Measurement sharedMeasurement [extern]
```

Definition at line 28 of file main.cpp.

```
{};
```

4.18 TaskACPower.cpp

Go to the documentation of this file.

```
#include <Arduino.h>
#include <Wire.h>
#include "TaskShared.h"
#include "TaskACPower.h"
#include "I2CLock.h"

// External shared variables
extern Measurement sharedMeasurement;
extern DFRobot_ADS1115 ads;
extern SemaphoreHandle_t i2cMutex;


// SEN0211 sensor parameters
#define CURRENT_DETECTION_RANGE 20    // Set your sensor's current detection range (5A, 10A, 20A)
#define VOLTAGE_REFERENCE 240.0       // Reference voltage for your AC line (220V, 110V, etc.)
#define CALIBRATION_FACTOR 1.070      // Calibration factor: actual/measured (4.76/4.45)

// Input pins on ADS1115
const int CURRENT_PIN = 0;  // ADS1115 A0 for current measurement

// Number of samples to average for more stable readings
const int SAMPLES = 10;


float readACCurrentValue(int pinNumber) {
    float peakVoltage = 0;
    float voltageRMS = 0;
    float currentValue = 0;

    // Take multiple samples for more stable reading
    const int SAMPLES = 10;

    if (xSemaphoreTake(i2cMutex, portMAX_DELAY) == pdTRUE) {
      for (int i = 0; i < SAMPLES; i++) {
        int rawReading = ads.readVoltage(pinNumber); // Reading in mV
        peakVoltage += rawReading;
        vTaskDelay(2 / portTICK_PERIOD_MS);
  ; // Small delay between readings
      }
      xSemaphoreGive(i2cMutex);
    }



    // Average the readings
    peakVoltage = peakVoltage / SAMPLES;

    // Convert peak to RMS (root mean square) - using 0.707 factor
    voltageRMS = peakVoltage * 0.707;

    // Convert the ADC millivolts to actual volts
    voltageRMS = voltageRMS / 1000.0; // Convert mV to V

    // Account for circuit amplification
    voltageRMS = voltageRMS / 2;

    // Convert voltage to current using detection range
    currentValue = voltageRMS * CURRENT_DETECTION_RANGE;

    // Apply calibration factor to correct the reading
    currentValue = currentValue * CALIBRATION_FACTOR;

    return currentValue;
  }


float calculatePower(float current) {
  return VOLTAGE_REFERENCE * current;
}

void TaskACPower(void *pvParameters) {
  (void)pvParameters;

  // // Initial delay to allow system to stabilize
  vTaskDelay(2000 / portTICK_PERIOD_MS);


  while (true) {
    float ACPower = 300.0;
    float current = readACCurrentValue(0);
```

```
    ACPower = current * VOLTAGE_REFERENCE;


      sharedMeasurement.ACPower = ACPower;

    current = readACCurrentValue(1);

    if(current > 0.06){
      sharedMeasurement.flag = 1;
    } else {
      sharedMeasurement.flag = 0;
    }

    // Wait before next reading (2 seconds)
    vTaskDelay(40 / portTICK_PERIOD_MS);
  }
}
```

4.19 src/TaskCalculateTime.cpp File Reference

FreeRTOS task that reads time from an RTC over I^{2}C and updates a UNIX timestamp.

```
#include "TaskCalculateTime.h"
#include "TaskShared.h"
#include "DFRobot_DS323X.h"
#include "I2CLock.h"
```

Functions

- void TaskCalculateTime (void *pvParameters)

  Shared structure containing sensor measurements including timestamp.

Variables

- SemaphoreHandle_t i2cMutex
- DFRobot_DS323X rtc

  Mutex for guarding I^{2}C access across tasks.
- Measurement sharedMeasurement

  RTC module used to get the current time.

4.19.1 Detailed Description

FreeRTOS task that reads time from an RTC over I^{2}C and updates a UNIX timestamp.

This task reads the current date and time from an RTC module using I^{2}C, converts it to a UNIX timestamp in nanoseconds, and stores it in the shared `Measurement` structure.

Definition in file TaskCalculateTime.cpp.

4.19.2 Function Documentation

4.19.2.1 TaskCalculateTime()

```
void TaskCalculateTime (
            void * pvParameters)
```

Shared structure containing sensor measurements including timestamp.

FreeRTOS task to calculate the current UNIX timestamp from the RTC and store it.

Periodically reads the date and time from the RTC via I^{2}C using a mutex to ensure safe access. The retrieved time is converted to a UNIX timestamp in nanoseconds using `mktime()` and stored in `sharedMeasurement.timestamp`.

Parameters

pvParameters	Unused parameter required by FreeRTOS task signature.

Definition at line 29 of file TaskCalculateTime.cpp.

```
00029                                          {
00030   (void)pvParameters;
00031
00032   while (true) {
00033     uint16_t year;
00034     uint8_t month, date, hour, minute, second;
00035
00036
00037
00038
00039       if (xSemaphoreTake(i2cMutex, portMAX_DELAY) == pdTRUE) {
00040         year   = rtc.getYear();
00041         month  = rtc.getMonth();
00042         date   = rtc.getDate();
00043         hour   = rtc.getHour();
00044         minute = rtc.getMinute();
00045         second = rtc.getSecond();
00046         xSemaphoreGive(i2cMutex);
00047       }
00048
00049
00050
00051     struct tm timeinfo;
00052     timeinfo.tm_year = year - 1900;
00053     timeinfo.tm_mon  = month - 1;
00054     timeinfo.tm_mday = date;
00055     timeinfo.tm_hour = hour;
00056     timeinfo.tm_min  = minute;
00057     timeinfo.tm_sec  = second;
00058     timeinfo.tm_isdst = 0;
00059
00060     time_t unix_seconds = mktime(&timeinfo);
00061     uint64_t unix_nanos = (uint64_t)unix_seconds * 1000000000ULL;
00062
00063
00064       sharedMeasurement.timestamp = unix_nanos;
00065
00066
00067     vTaskDelay(200 / portTICK_PERIOD_MS);
00068   }
00069 }
```

4.19.3 Variable Documentation

4.19.3.1 i2cMutex

```
SemaphoreHandle_t i2cMutex  [extern]
```

Definition at line 29 of file main.cpp.

4.19.3.2 rtc

```
DFRobot_DS323X rtc [extern]
```

Mutex for guarding I^{2}C access across tasks.

Definition at line 25 of file main.cpp.

4.19.3.3 sharedMeasurement

```
Measurement sharedMeasurement [extern]
```

RTC module used to get the current time.

Definition at line 28 of file main.cpp.

```
00028 {};
```

4.20 TaskCalculateTime.cpp

Go to the documentation of this file.

```
#include "TaskCalculateTime.h"
#include "TaskShared.h"
#include "DFRobot_DS323X.h"
#include "I2CLock.h"
extern SemaphoreHandle_t i2cMutex;
extern DFRobot_DS323X rtc;
extern Measurement sharedMeasurement;
void TaskCalculateTime(void *pvParameters) {
  (void)pvParameters;
  while (true) {
    uint16_t year;
    uint8_t month, date, hour, minute, second;
      if (xSemaphoreTake(i2cMutex, portMAX_DELAY) == pdTRUE) {
        year   = rtc.getYear();
        month  = rtc.getMonth();
        date   = rtc.getDate();
        hour   = rtc.getHour();
        minute = rtc.getMinute();
        second = rtc.getSecond();
        xSemaphoreGive(i2cMutex);
      }
    struct tm timeinfo;
    timeinfo.tm_year = year - 1900;
    timeinfo.tm_mon  = month - 1;
    timeinfo.tm_mday = date;
    timeinfo.tm_hour = hour;
    timeinfo.tm_min  = minute;
    timeinfo.tm_sec  = second;
    timeinfo.tm_isdst = 0;

    time_t unix_seconds = mktime(&timeinfo);
    uint64_t unix_nanos = (uint64_t)unix_seconds * 1000000000ULL;
      sharedMeasurement.timestamp = unix_nanos;
    vTaskDelay(200 / portTICK_PERIOD_MS);
  }
}
```

4.21 src/TaskPressure.cpp File Reference

FreeRTOS task that reads data from an I^2C pressure sensor and updates shared measurements.

```
#include "TaskPressure.h"
#include "TaskShared.h"
```

Functions

- void TaskPressure (void *pvParameters)

  Mutex for guarding I^2C access across tasks.

Variables

- const uint8_t SENSOR_ADDR = 0x28
- const uint16_t OUTPUT_MIN = 1638
- const uint16_t OUTPUT_MAX = 14746
- const float PRESSURE_MAX_PSI = 200.0
- const float PSI_TO_BAR = 0.0689476
- Measurement sharedMeasurement

  RTC module used to get the current time.
- SemaphoreHandle_t i2cMutex

4.21.1 Detailed Description

FreeRTOS task that reads data from an I^2C pressure sensor and updates shared measurements.

This task communicates with a digital pressure sensor over I^2C, converts the raw 14-bit pressure reading to bar, and stores the result in the shared `Measurement` structure.

Definition in file TaskPressure.cpp.

4.21.2 Function Documentation

4.21.2.1 TaskPressure()

```
void TaskPressure (
            void * pvParameters)
```

Mutex for guarding I^2C access across tasks.

FreeRTOS task that reads pressure data from a sensor and updates the shared measurement.

This task waits 1 second before starting, then enters a loop where it:

- Takes the I^2C mutex.
- Requests 4 bytes of data from a digital pressure sensor at address 0x28.
- Parses the 14-bit raw pressure reading.
- Converts it to PSI, then to bar.
- Releases the mutex and updates the global `sharedMeasurement.pressure` value.

If the sensor doesn't respond properly, an error message is printed over Serial.

Parameters

pvParameters	Unused parameter required by FreeRTOS task signature.

Definition at line 37 of file TaskPressure.cpp.

```
00037                                   {
00038   (void)pvParameters;
00039
00040   vTaskDelay(1000 / portTICK_PERIOD_MS);
00041
00042   while (true) {
00043     float pressure = 0.0;
00044
00045     if (xSemaphoreTake(i2cMutex, portMAX_DELAY) == pdTRUE) {
00046
00047
00048
00049     // Request data from the pressure sensor
00050     Wire.requestFrom(SENSOR_ADDR, (uint8_t)4);
00051
00052     if (Wire.available() == 4) {
00053       uint8_t buffer[4];
00054       for (int i = 0; i < 4; ++i) {
00055         buffer[i] = Wire.read();
00056       }
00057
00058       // Extract 14-bit raw pressure
00059       uint16_t raw_pressure = ((buffer[0] & 0x3F) « 8) | buffer[1];
00060
00061       // Convert to pressure
00062       float pressure_psi = ((float)(raw_pressure - OUTPUT_MIN) / (OUTPUT_MAX - OUTPUT_MIN)) *
      PRESSURE_MAX_PSI;
00063       pressure = pressure_psi * PSI_TO_BAR;
00064
00065
00066     } else {
00067       Serial.println("Failed to read pressure sensor data");
00068     }
00069     xSemaphoreGive(i2cMutex);
00070 }
00071
00072       sharedMeasurement.pressure = pressure;
00073
00074
00075     vTaskDelay(pdMS_TO_TICKS(2000));
00076   }
00077 }
```

4.21.3 Variable Documentation

4.21.3.1 i2cMutex

`SemaphoreHandle_t i2cMutex [extern]`

Definition at line 29 of file main.cpp.

4.21.3.2 OUTPUT_MAX

`const uint16_t OUTPUT_MAX = 14746`

Definition at line 14 of file TaskPressure.cpp.

4.21.3.3 OUTPUT_MIN

`const uint16_t OUTPUT_MIN = 1638`

Definition at line 13 of file TaskPressure.cpp.

4.21.3.4 PRESSURE_MAX_PSI

```
const float PRESSURE_MAX_PSI = 200.0
```

Definition at line 15 of file TaskPressure.cpp.

4.21.3.5 PSI_TO_BAR

```
const float PSI_TO_BAR = 0.0689476
```

Definition at line 16 of file TaskPressure.cpp.

4.21.3.6 SENSOR_ADDR

```
const uint8_t SENSOR_ADDR = 0x28
```

Definition at line 12 of file TaskPressure.cpp.

4.21.3.7 sharedMeasurement

```
Measurement sharedMeasurement [extern]
```

RTC module used to get the current time.

Definition at line 28 of file main.cpp.

```
00028 {};
```

4.22 TaskPressure.cpp

Go to the documentation of this file.

```
00001
00008
00009 #include "TaskPressure.h"
00010 #include "TaskShared.h"
00011
00012 const uint8_t SENSOR_ADDR = 0x28;
00013 const uint16_t OUTPUT_MIN = 1638;
00014 const uint16_t OUTPUT_MAX = 14746;
00015 const float PRESSURE_MAX_PSI = 200.0;
00016 const float PSI_TO_BAR = 0.0689476;
00017
00018 extern Measurement sharedMeasurement;
00019
00020 extern SemaphoreHandle_t i2cMutex;
00021
00022
00037 void TaskPressure(void *pvParameters) {
00038   (void)pvParameters;
00039
00040   vTaskDelay(1000 / portTICK_PERIOD_MS);
00041
00042   while (true) {
00043     float pressure = 0.0;
00044
00045     if (xSemaphoreTake(i2cMutex, portMAX_DELAY) == pdTRUE) {
00046
00047
00048
00049     // Request data from the pressure sensor
00050     Wire.requestFrom(SENSOR_ADDR, (uint8_t)4);
```

```
    if (Wire.available() == 4) {
      uint8_t buffer[4];
      for (int i = 0; i < 4; ++i) {
        buffer[i] = Wire.read();
      }

      // Extract 14-bit raw pressure
      uint16_t raw_pressure = ((buffer[0] & 0x3F) << 8) | buffer[1];

      // Convert to pressure
      float pressure_psi = ((float)(raw_pressure - OUTPUT_MIN) / (OUTPUT_MAX - OUTPUT_MIN)) *
  PRESSURE_MAX_PSI;
      pressure = pressure_psi * PSI_TO_BAR;


    } else {
      Serial.println("Failed to read pressure sensor data");
    }
    xSemaphoreGive(i2cMutex);
}

      sharedMeasurement.pressure = pressure;


    vTaskDelay(pdMS_TO_TICKS(2000));
  }
}
```

4.23 src/TaskPublish.cpp File Reference

FreeRTOS task that serializes and publishes sensor data over Serial as JSON.

```
#include "TaskPublish.h"
#include "TaskShared.h"
#include <ArduinoJson.h>
```

Functions

- void TaskPublish (void ∗pvParameters)

  FreeRTOS task that publishes sensor data as JSON over Serial.

Variables

- Measurement sharedMeasurement

  Shared structure containing the latest sensor measurements.

4.23.1 Detailed Description

FreeRTOS task that serializes and publishes sensor data over Serial as JSON.

This task reads the most recent sensor measurements from a shared structure and sends them as a JSON object over the Serial interface at regular intervals.

Definition in file TaskPublish.cpp.

4.23.2 Function Documentation

4.23.2.1 TaskPublish()

```
void TaskPublish (
            void * pvParameters)
```

FreeRTOS task that publishes sensor data as JSON over Serial.

Every 300 ms, this task creates a JSON object containing pressure, temperature, power, timestamp, and a status flag from the shared measurement structure. It prints the JSON to the Serial port for external logging or monitoring.

Parameters

pvParameters	Unused parameter required by FreeRTOS task signature.

Definition at line 25 of file TaskPublish.cpp.

```
00025                                    {
00026   (void)pvParameters;
00027
00028   while (true) {
00029     Measurement localCopy;
00030
00031
00032       localCopy = sharedMeasurement;
00033
00034
00035     StaticJsonDocument<128> doc;
00036     doc["pressure"] = localCopy.pressure;
00037     doc["temperature"] = localCopy.temperature;
00038     doc["power"] = localCopy.ACPower;
00039     doc["timestamp"] = localCopy.timestamp;
00040     doc["flag"] = localCopy.flag;
00041     serializeJson(doc, Serial);
00042     Serial.println();
00043
00044     vTaskDelay(pdMS_TO_TICKS(300));
00045   }
00046 }
```

4.23.3 Variable Documentation

4.23.3.1 sharedMeasurement

```
Measurement sharedMeasurement  [extern]
```

Shared structure containing the latest sensor measurements.

Shared structure containing the latest sensor measurements.

Definition at line 28 of file main.cpp.

```
00028 {};
```

4.24 TaskPublish.cpp

Go to the documentation of this file.

```
#include "TaskPublish.h"
#include "TaskShared.h"
#include <ArduinoJson.h>

extern Measurement sharedMeasurement;

void TaskPublish(void *pvParameters) {
  (void)pvParameters;

  while (true) {
    Measurement localCopy;


      localCopy = sharedMeasurement;


    StaticJsonDocument<128> doc;
    doc["pressure"] = localCopy.pressure;
    doc["temperature"] = localCopy.temperature;
    doc["power"] = localCopy.ACPower;
    doc["timestamp"] = localCopy.timestamp;
    doc["flag"] = localCopy.flag;
    serializeJson(doc, Serial);
    Serial.println();

    vTaskDelay(pdMS_TO_TICKS(300));
  }
}
```

4.25 src/TaskReceiveTime.cpp File Reference

FreeRTOS task that listens for JSON input over Serial to update the RTC.

```
#include "TaskReceiveTime.h"
#include <ArduinoJson.h>
#include "DFRobot_DS323X.h"
```

Functions

- void TaskReceiveTime (void *pvParameters)

  FreeRTOS task to receive and process time-setting commands over Serial.

Variables

- DFRobot_DS323X rtc

  Global RTC object used to set the time.

4.25.1 Detailed Description

FreeRTOS task that listens for JSON input over Serial to update the RTC.

This task waits for a JSON message containing time information via the serial port. If the JSON is valid and contains a `set_time` command along with time fields, it updates the RTC module using the provided values.

Definition in file TaskReceiveTime.cpp.

4.25.2 Function Documentation

4.25.2.1 TaskReceiveTime()

```
void TaskReceiveTime (
        void * pvParameters)
```

FreeRTOS task to receive and process time-setting commands over Serial.

Expects a JSON payload such as:

```
{
  "set_time": true,
  "year": 2025,
  "month": 5,
  "day": 8,
  "hour": 14,
  "minute": 30,
  "second": 0
}
```

If valid, it sets the RTC accordingly and replies with a success message.

Parameters

pvParameters	Unused parameter required by FreeRTOS task signature.

Definition at line 36 of file TaskReceiveTime.cpp.

```
00036                                       {
00037   (void)pvParameters;
00038
00039   String input;
00040
00041   while (true) {
00042     if (Serial.available()) {
00043       input = Serial.readStringUntil('\n');
00044
00045       StaticJsonDocument<200> doc;
00046       DeserializationError err = deserializeJson(doc, input);
00047
00048       if (err) {
00049         Serial.println("{\"error\":\"Invalid JSON\"}");
00050         continue;
00051       }
00052
00053       if (doc["set_time"] == true &&
00054           doc.containsKey("year") &&
00055           doc.containsKey("month") &&
00056           doc.containsKey("day") &&
00057           doc.containsKey("hour") &&
00058           doc.containsKey("minute") &&
00059           doc.containsKey("second")) {
00060
00061         int year = doc["year"].as<int>();
00062         int month = doc["month"].as<int>();
00063         int day = doc["day"].as<int>();
00064         int hour = doc["hour"].as<int>();
00065         int minute = doc["minute"].as<int>();
00066         int second = doc["second"].as<int>();
00067
00068         rtc.setTime(year, month, day, hour, minute, second);
00069
00070         Serial.println("{\"status\":\"RTC updated\"}");
00071       }
00072     }
00073
00074     vTaskDelay(pdMS_TO_TICKS(100));
00075   }
00076 }
```

4.25.3 Variable Documentation

4.25.3.1 rtc

```
DFRobot_DS323X rtc  [extern]
```

Global RTC object used to set the time.

Global RTC object used to set the time.

Definition at line 25 of file main.cpp.

4.26 TaskReceiveTime.cpp

Go to the documentation of this file.

```
#include "TaskReceiveTime.h"
#include <ArduinoJson.h>
#include "DFRobot_DS323X.h"

extern DFRobot_DS323X rtc;

void TaskReceiveTime(void *pvParameters) {
  (void)pvParameters;

  String input;

  while (true) {
    if (Serial.available()) {
      input = Serial.readStringUntil('\n');

      StaticJsonDocument<200> doc;
      DeserializationError err = deserializeJson(doc, input);

      if (err) {
        Serial.println("{\"error\":\"Invalid JSON\"}");
        continue;
      }

      if (doc["set_time"] == true &&
          doc.containsKey("year") &&
          doc.containsKey("month") &&
          doc.containsKey("day") &&
          doc.containsKey("hour") &&
          doc.containsKey("minute") &&
          doc.containsKey("second")) {

        int year = doc["year"].as<int>();
        int month = doc["month"].as<int>();
        int day = doc["day"].as<int>();
        int hour = doc["hour"].as<int>();
        int minute = doc["minute"].as<int>();
        int second = doc["second"].as<int>();

        rtc.setTime(year, month, day, hour, minute, second);

        Serial.println("{\"status\":\"RTC updated\"}");
      }
    }

    vTaskDelay(pdMS_TO_TICKS(100));
  }
}
```

4.27 src/TaskTemperature.cpp File Reference

FreeRTOS task that reads data from a MAX31865 temperature sensor and updates the shared measurement structure.

```
#include <Arduino.h>
#include <Adafruit_MAX31865.h>
#include <SPI.h>
#include "TaskShared.h"
```

Macros

- #define MAX31865_CS_PIN 14
- #define MAX31865_DI_PIN 23
- #define MAX31865_DO_PIN 19
- #define MAX31865_CLK_PIN 18
- #define RREF 430.0

  The value of the Rref resistor. Use 430.0 for PT100 and 4300.0 for PT1000.
- #define RNOMINAL 100.0

  The 'nominal' 0-degrees-C resistance of the sensor.

Functions

- void TaskTemperature (void *pvParameters)

  FreeRTOS task to read temperature from the MAX31865 sensor.

Variables

- Measurement sharedMeasurement

  Shared measurement structure used across tasks.
- Adafruit_MAX31865 thermo = Adafruit_MAX31865(MAX31865_CS_PIN, MAX31865_DI_PIN, MAX31865_DO_PIN, MAX31865_CLK_PIN)

  Use software SPI: CS, DI, DO, CLK.

4.27.1 Detailed Description

FreeRTOS task that reads data from a MAX31865 temperature sensor and updates the shared measurement structure.

Definition in file TaskTemperature.cpp.

4.27.2 Macro Definition Documentation

4.27.2.1 MAX31865_CLK_PIN

```
#define MAX31865_CLK_PIN 18
```

Definition at line 18 of file TaskTemperature.cpp.

4.27.2.2 MAX31865_CS_PIN

```
#define MAX31865_CS_PIN 14
```

Definition at line 15 of file TaskTemperature.cpp.

4.27.2.3 MAX31865_DI_PIN

```
#define MAX31865_DI_PIN 23
```

Definition at line 16 of file TaskTemperature.cpp.

4.27.2.4 MAX31865_DO_PIN

```
#define MAX31865_DO_PIN 19
```

Definition at line 17 of file TaskTemperature.cpp.

4.27.2.5 RNOMINAL

```
#define RNOMINAL 100.0
```

The 'nominal' 0-degrees-C resistance of the sensor.

Definition at line 26 of file TaskTemperature.cpp.

4.27.2.6 RREF

```
#define RREF 430.0
```

The value of the Rref resistor. Use 430.0 for PT100 and 4300.0 for PT1000.

Definition at line 24 of file TaskTemperature.cpp.

4.27.3 Function Documentation

4.27.3.1 TaskTemperature()

```
void TaskTemperature (
             void * pvParameters)
```

FreeRTOS task to read temperature from the MAX31865 sensor.

FreeRTOS task that reads temperature sensor data and processes it.

Periodically reads the RTD value, computes the temperature, and stores it in a shared structure.

Parameters

pvParameters	Unused parameter required by FreeRTOS task signature.

Initial delay to allow other initializations to complete

SPI initialization for temperature sensor

Initialize the MAX31865 with 2WIRE configuration

Initial reading to check if sensor is working

Task loop

Read raw RTD value

Default value

Only process if we got a valid reading

Calculate temperature using the library function

Check for out-of-range values

Check for sensor faults

Clear fault

Try to reinitialize the sensor

Definition at line 37 of file TaskTemperature.cpp.

```
00037                                       {
00038   (void)pvParameters;
00039
00041   vTaskDelay(2000 / portTICK_PERIOD_MS);
00042
00043   Serial.println("Temperature sensor task starting...");
00044
00046   SPI.begin(); //redundant?
00047
00049   thermo.begin(MAX31865_2WIRE);
00050
00052   uint16_t rtd_initial = thermo.readRTD();
00053   Serial.print("Initial RTD reading: ");
00054   Serial.println(rtd_initial);
00055
00056   if (rtd_initial == 0) {
00057     Serial.println("WARNING: Temperature sensor not responding, check connections");
00058   }
00059
00061   while (true) {
00063     uint16_t rtd = thermo.readRTD();
00064
00065     float temperature = 22.0;
00066
00068     if (rtd > 0) {
00070       temperature = thermo.temperature(RNOMINAL, RREF) - 7.0;
00071
00072
00073       float ratio = rtd;
00074       ratio /= 32768;
00075       float resistance = RREF * ratio;
00076
00077
00079       if (temperature < -50 || temperature > 200) {
00080         Serial.println("Temperature out of reasonable range, using default");
00081         temperature = 22.0;
00082       }
00083     } else {
00084       Serial.println("Invalid RTD reading, using default temperature");
00085     }
```

```

    uint8_t fault = thermo.readFault();
    if (fault) {
      Serial.print("Fault detected 0x");
      Serial.println(fault, HEX);

      thermo.clearFault();

      thermo.begin(MAX31865_2WIRE);
      vTaskDelay(100 / portTICK_PERIOD_MS);
    }

      sharedMeasurement.temperature = temperature;

    vTaskDelay(2000 / portTICK_PERIOD_MS);
  }
}
```

4.27.4 Variable Documentation

4.27.4.1 sharedMeasurement

`Measurement sharedMeasurement [extern]`

Shared measurement structure used across tasks.

Shared measurement structure used across tasks.

Shared structure containing the latest sensor measurements.

Definition at line 28 of file main.cpp.

```
00028 {};
```

4.27.4.2 thermo

`Adafruit_MAX31865 thermo = Adafruit_MAX31865(MAX31865_CS_PIN, MAX31865_DI_PIN, MAX31865_DO_PIN, MAX31865_CLK_PIN)`

Use software SPI: CS, DI, DO, CLK.

Definition at line 21 of file TaskTemperature.cpp.

4.28 TaskTemperature.cpp

Go to the documentation of this file.

```
00001
00006
00007 #include <Arduino.h>
00008 #include <Adafruit_MAX31865.h>
00009 #include <SPI.h>
00010 #include "TaskShared.h"
00011
00013 extern Measurement sharedMeasurement;
00014
00015 #define MAX31865_CS_PIN    14
00016 #define MAX31865_DI_PIN    23
00017 #define MAX31865_DO_PIN    19
00018 #define MAX31865_CLK_PIN   18
00019
00021 Adafruit_MAX31865 thermo = Adafruit_MAX31865(MAX31865_CS_PIN, MAX31865_DI_PIN, MAX31865_DO_PIN,
      MAX31865_CLK_PIN);
00022
00024 #define RREF      430.0
```

```
#define RNOMINAL  100.0


void TaskTemperature(void *pvParameters) {
  (void)pvParameters;

  vTaskDelay(2000 / portTICK_PERIOD_MS);

  Serial.println("Temperature sensor task starting...");

  SPI.begin(); //redundant?

  thermo.begin(MAX31865_2WIRE);

  uint16_t rtd_initial = thermo.readRTD();
  Serial.print("Initial RTD reading: ");
  Serial.println(rtd_initial);

  if (rtd_initial == 0) {
    Serial.println("WARNING: Temperature sensor not responding, check connections");
  }

  while (true) {
    uint16_t rtd = thermo.readRTD();

    float temperature = 22.0;

    if (rtd > 0) {
      temperature = thermo.temperature(RNOMINAL, RREF) - 7.0;


      float ratio = rtd;
      ratio /= 32768;
      float resistance = RREF * ratio;


      if (temperature < -50 || temperature > 200) {
        Serial.println("Temperature out of reasonable range, using default");
        temperature = 22.0;
      }
    } else {
      Serial.println("Invalid RTD reading, using default temperature");
    }

    uint8_t fault = thermo.readFault();
    if (fault) {
      Serial.print("Fault detected 0x");
      Serial.println(fault, HEX);

      thermo.clearFault();

      thermo.begin(MAX31865_2WIRE);
      vTaskDelay(100 / portTICK_PERIOD_MS);
    }

      sharedMeasurement.temperature = temperature;

    vTaskDelay(2000 / portTICK_PERIOD_MS);
  }
}
```

ESP32 Development Log

MT | *DAB*

April 3, 2025

Installed PlatformIO on a Linux machine provided by Semcon. Installed PlatformIO both as a tool in the commandline and as a plugin in VScode.

Figure C.1.1: Installation PlatformIO

Figure C.1.2: PlatformIO VSCode Plugin

April 4, 2025

Checked out test driven development in PlatformIO. Concluded that we will just do unit testing and not full blown test driven development.

Unit test: T-ESP32-1.0 - Temperature sensor testing (Appx. A.4).

Established initial contact with temperature sensor (this was already tested directly on Raspberry Pi but we are migrating the functionality over to ESP32).
Working example code was provided from DFRobot [24]

To get the code to work we just had to do some minor adjustments and familiarization with using platformio.ini file to include libraries.

Code used for testing: GitHub [1]

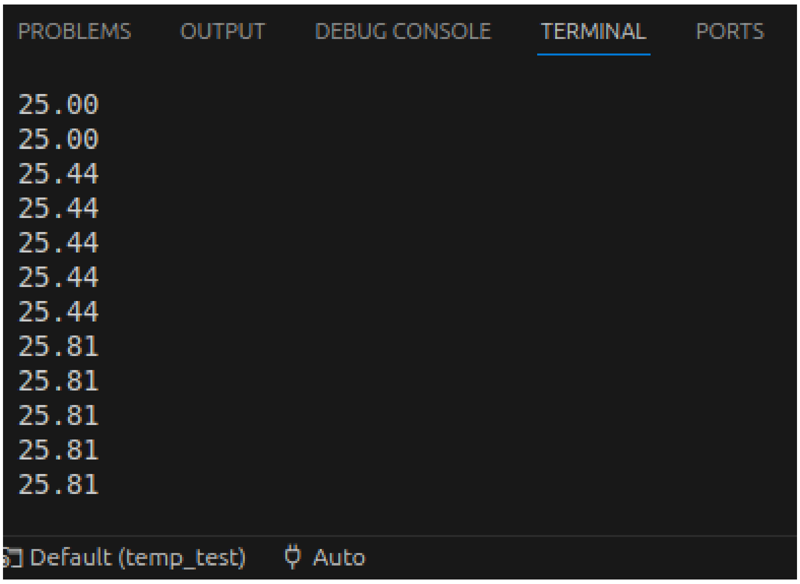


Figure C.1.3: Terminal readout temperature sensor

Conclusion: Test passes. Sensor works.
Notes: Temperature changes when we touch the probe. Cannot conclude if accurate.

[1] https://github.com/martintara/kaffeknekt/blob/martin/temp_test/src/main.cpp Copy of this file also on USB.

April 7, 2025

Unit test: T-ESP32-1.1 - DC current sensor testing (Appx. A.4)

Code used for testing: GitHub [2]
Example code provided from DFRobot [23].
We just had to do adjustments in platformio.ini to import the correct libraries to get the code running on ESP32.
Library found at DFRobot's GitHub [3].

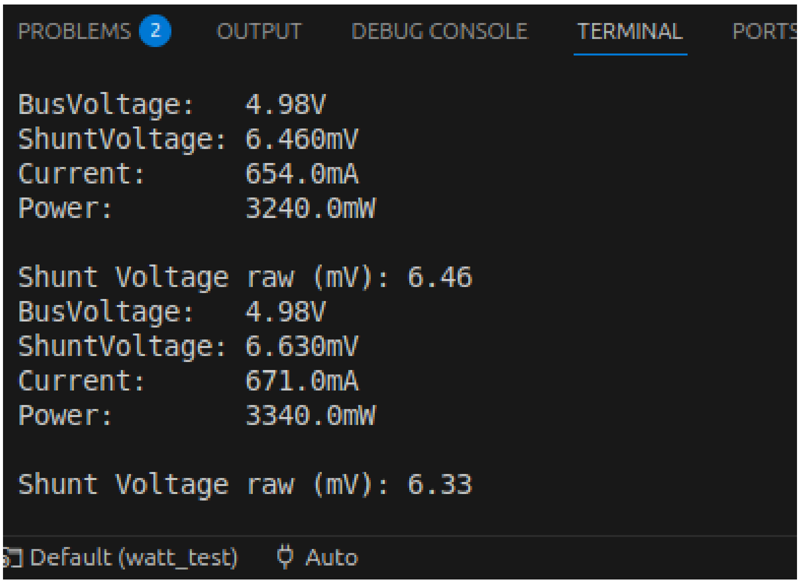


Figure C.1.4: Terminal readout dc current sensor

Conclusion: Test passes. Sensor works.
Note: Values are changing. Cannot conclude if accurate. Datasheet mentions calibration.

[2] https://github.com/martintara/kaffeknekt/blob/martin/watt_test/src/main.cpp Copy of this file also on USB.
[3] https://github.com/DFRobot/DFRobot_INA219

Unit test: T-ESP32-1.2 - DC current sensor calibration (Appx. A.4)

The calibration works accordingly:
Step 1. Set the in219Reading_mA variable to value observed in terminal readout.
Step 2: Set extMeterReading_mA variable to measured current.
We used a digital multimeter for reference.

Figure C.1.5: Before calibration.

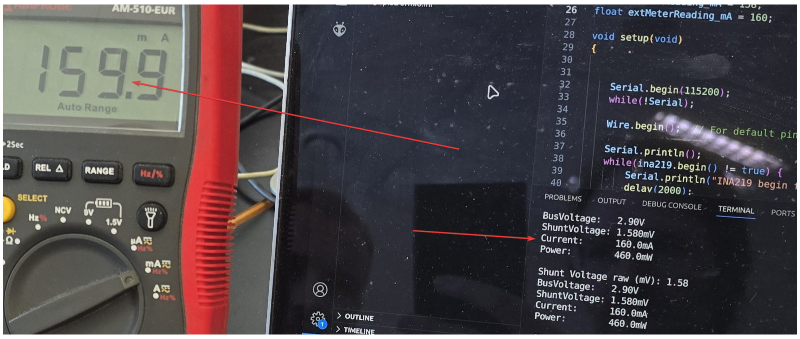


Figure C.1.6: After calibration.

Conclusion: Test passes. Sensor is now calibrated.

Unit test: T-ESP32-1.3 - AC current sensor testing (Appx. A.4)
Setup: Sensor connected with 1 pin (excluding ground and power). Sensor is clamped around he electrical wire of 5v power supply for Raspberry Pi.

Example code provided from DFRobot [21]

Code used for testing: GitHub [4]

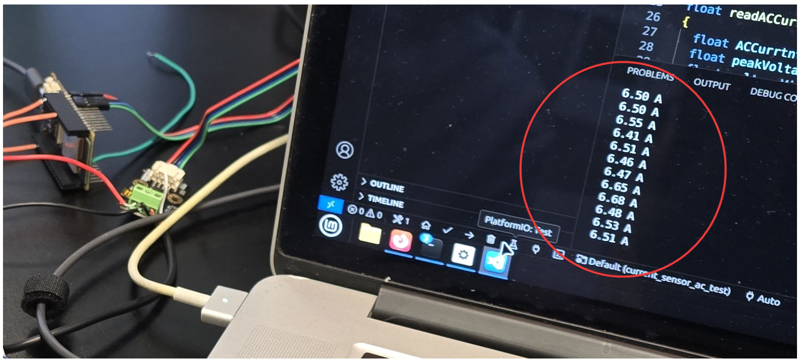


Figure C.1.7: Terminal readout.

Conclusion: Test fails. We tested this on power supply that outputs 5V for a Raspeberry Pi. The power supply does not run hot. I cant make these numbers to make sense. Further testing and consulting with the electrical team needed.

Unit test: T-ESP32-1.4 - USB communication test (Appx. A.4)
Tested using pyserial for communication between ESP32 and Raspberry Pi. Forgot to screenshot. It was just a basic python script using pyserial library.

Conclusion: Test passes. We have communication! And can inform rest of dev team about potential interface between signal coming in to Raspberry Pi from ESP32 for then to be used to push data into Influx DB.
Note: The idea is to structure the data being sent over usb by using JSON format since it is easy to read and understand.

[4]https://github.com/martintara/kaffeknekt/blob/3bcda6b2c9aee31192f155499cf180b84258a6b0/current_sensor_ac_test/src/main.cpp Copy of this file also on USB.

April 8, 2025

Unit test: T-ESP32-1.5 - Combining two sensors (Appx. A.4)

Code used for testing: GitHub [5]

Approach: Used Claude AI to combine the already working code snippets for temperature and current sensor.

Conclusion: Test passes. Tasking works.

Getting readouts from both sensors. Forgot screenshot.

Note: Need to start planning how to structure the code. Using ai to combine the tasks is all ready starting to get a bit messy. Also: The system is not connected to the internet - In this current iteration the ESP32 gets the time sent from the Raspberry Pi. This is a bad idea since it can cause the time to drift and have to be revised.

[5] https://github.com/martintara/kaffeknekt/blob/0673ad0f92ce84eff4f25eab455542ad74a3edc3/combined_test/src/main.cpp Copy of this file also on USB.

April 10, 2025

Unit test: T-ESP32-1.6 - Testing pressure sensor (Appx. A.4)

Code used for testing: GitHub (history version) [6]
We did not find any arduino example code for this sensor. We found the conversion formula in the NPI-19 datasheet [58]
We instructed Claude AI to make an example code.
Conversion formula:

$$\text{Output (Counts)} = \frac{P_{\text{applied}} - P_{\text{min}}}{P_{\text{max}} - P_{\text{min}}} \times (\text{Output}_{\text{max}} - \text{Output}_{\text{min}}) \tag{C.1.1}$$

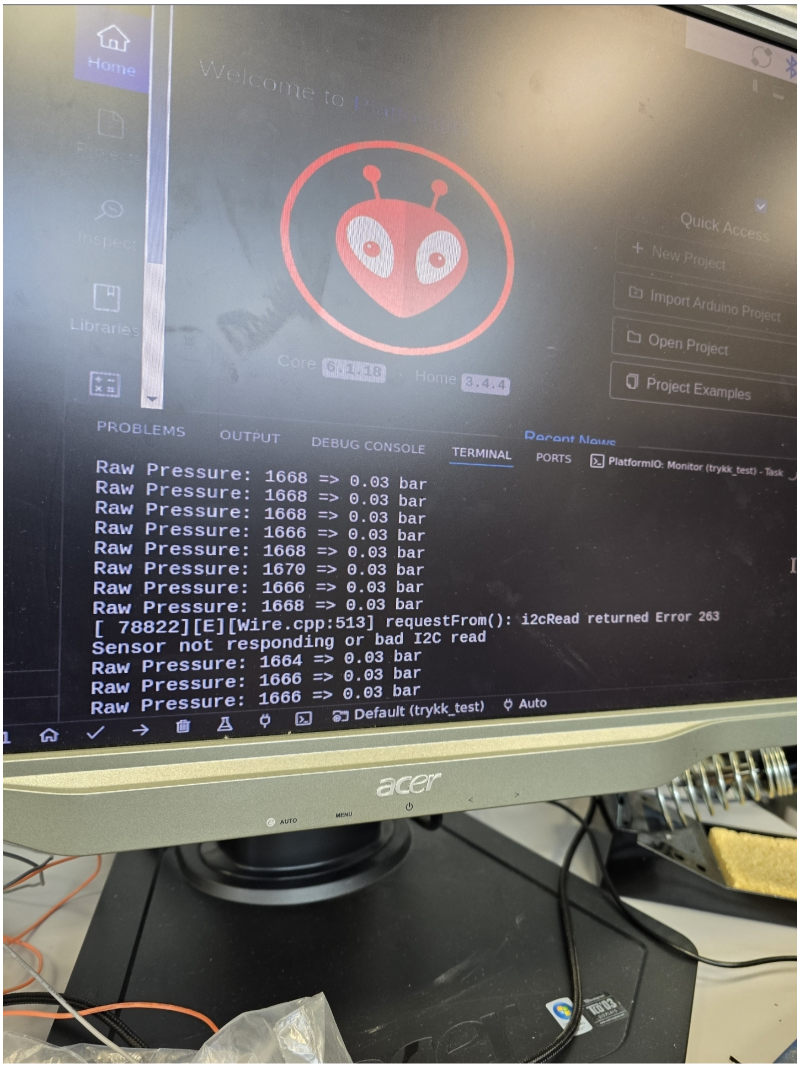


Figure C.1.8: Terminal readout.

Conclusion: Test passes. Readouts makes sense since the sensor is in open air.
Notes: We notice a sporadic I2C error once in a while.

[6] https://github.com/martintara/kaffeknekt/blob/c37503749b72bc410b1b505dc2fac483dea63eb0/trykk_test/src/main.cpp Copy of this file also on USB.

April 11, 2025

Unit test: T-ESP32-1.7 - AC current sensor 2nd test (Appx. A.4)

We concluded that the internal ADC of the ESP32 is not up to par. Will now test an ADC module with higher bitrate.

Example code provided from DFRobot [22].

We did not get the example code to work. We used Claude AI to translate the working Arduino example code to ESP32.

Code used for testing: GitHub [7]

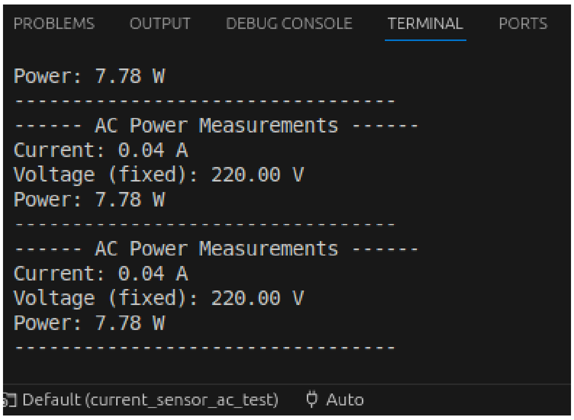


Figure C.1.9: Terminal readout.

Conclusion: Test fails. Readouts makes no sense: we are still hooked up to a water boiler.

[7] https://github.com/martintara/kaffeknekt/blob/martin/current_sensor_ac_test/src/main.cpp Copy of this file also on USB.

April 14, 2025

Unit test: T-ESP32-1.8 - RTC-module testing (Appx. A.4)

Testing using RTC module instead of getting the time from Raspberry Pi.

Example code found at DFRobot [19].
Code used for testing: GitHub [8] and: GitHub [9]

We used Claude AI to make a python script to test sending timestamps as JSON format.

Libraries imported from GitHub [10]

This defines the RTC interfaces such as uint8_t getMinutes(), uint8_t getSeconds() and void setTime(uint16_t year, uint8_t month, uint8_t date, uint8_t hour, uint8_t minute, uint8_t second) etc. GitHub [11] adds the functionality to serialize a JSON document.

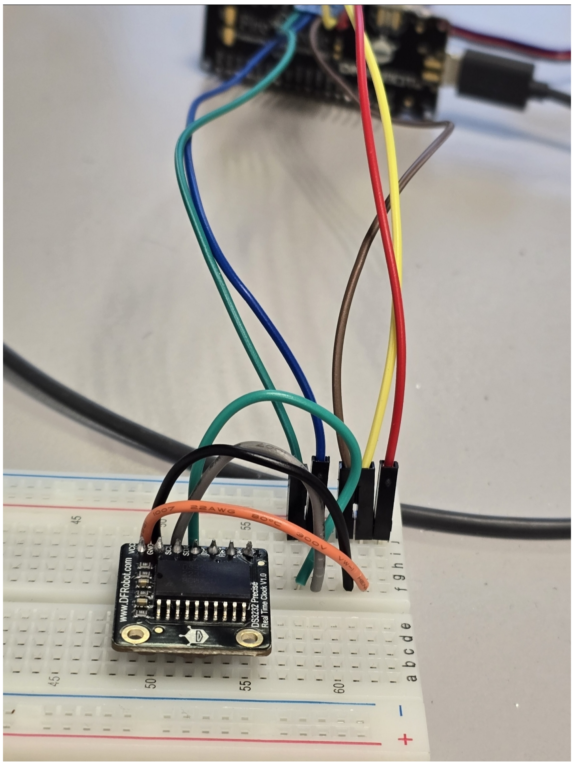


Figure C.1.10: RTC-module on testbench.

Conclusion: Test passes. Setting and getting the time works. Forgot screenshot.

[8] https://github.com/martintara/kaffeknekt/blob/martin/rtc_test/src/main.cpp Copy of this file also on USB.
[9] https://github.com/martintara/kaffeknekt/blob/martin/rtc_test/rtc_json.py Copy of this file also on USB.
[10] https://github.com/DFRobot/DFRobot_DS323X
[11] https://github.com/bblanchon/ArduinoJson

April 15, 2025

Unit test: T-ESP32-1.9 - AC current sensor 3rd test (Appx. A.4)
Setup: This time the current sensor is hooked up to a water boiler that we know is between 1000 and 1300 watts.

Picking up where we left off with the AC current sensor. After consulting with electrical, we try to use a better adc [22].
Code used for testing: GitHub [12]

Code is based on the example code from the DFrobot wiki [22].

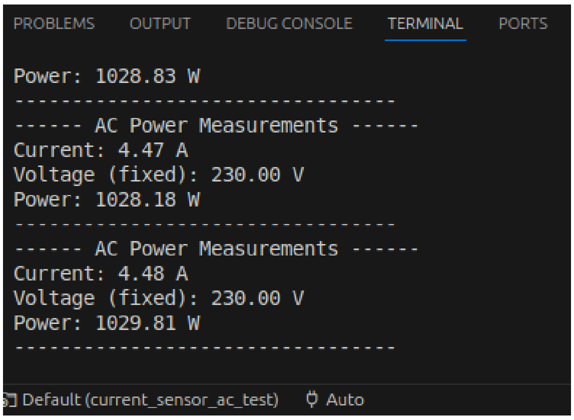


Figure C.1.11: Terminal readout - with load.

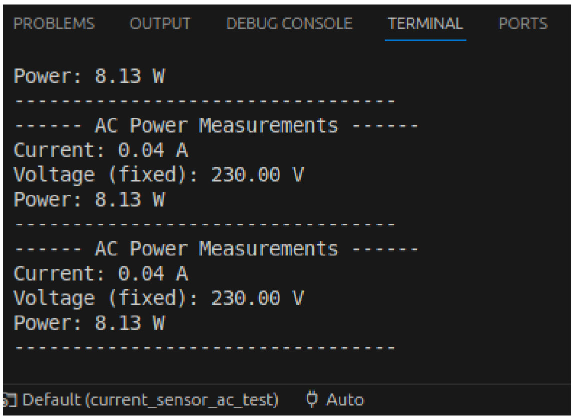


Figure C.1.12: Terminal readout - Without load.

Conclusion: Test passes.
Note: These readouts makes sense, but we want to confirm this with using ampere meter later.

[12] https://github.com/martintara/kaffeknekt/blob/martin/current_sensor_ac_test/src/main.cpp Copy of this file also on USB.

April 22, 2025

Moved the testbench down to the coffee machine in the workshop.

After installing the sensor and turning on the coffee machine, we quickly discovered a leakage and had to power off the machine.

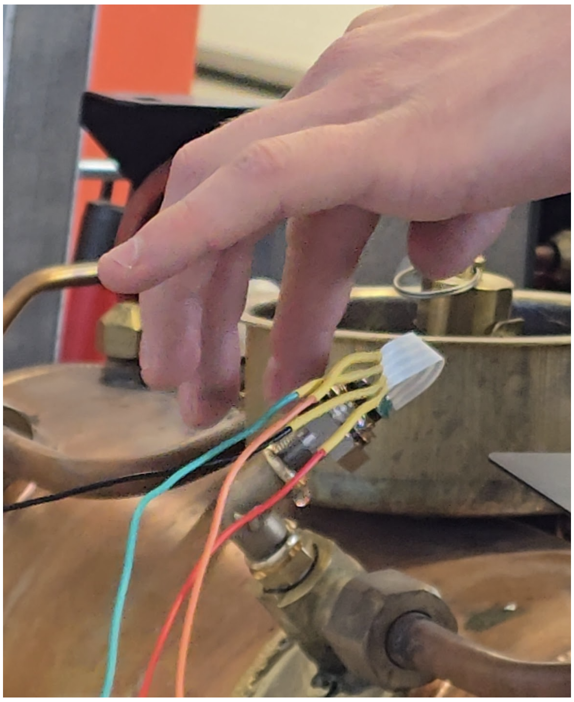


Figure C.1.13: Aborted test - Leakage.

Conclusion: Test aborted.

April 23, 2025

Unit test: T-ESP32-2.0 - Calibrating AC current sensor (Appx. A.4)

Picked up where we left off, where we had the idea to measure actual current through the water boiler using a ampere meter. After applying a calibration factor of 1.070 (4.76 / 4.45) (measured/code output) the system now senses the current accurately.

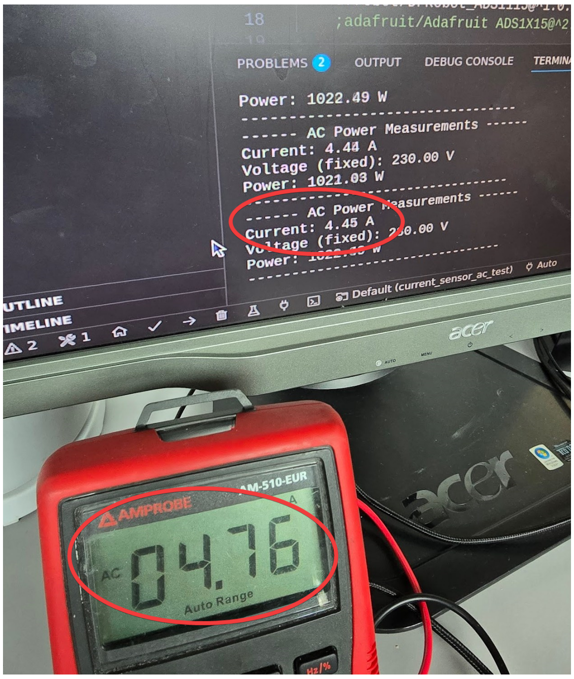


Figure C.1.14: Before calibration.

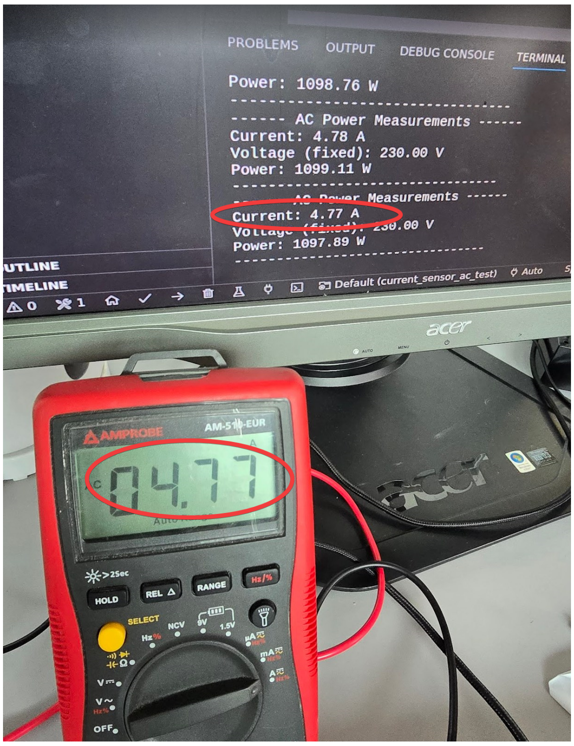


Figure C.1.15: After calibration.

Conclusion: Test passes. Current sensor now satisfies our needs (measuring current accurately)

Unit test: T-ESP32-2.1 - Testing MQTT (Appx. A.4)

In a meeting with Semcon we uncovered a new "want" and that was having wireless communication between Raspberry Pi and ESP32.
Code used for testing: GitHub [13]
Test code generated by Claude AI.

When trying to get MQTT to work the best solution seemed to be hosting a local wifi hotspot from the Raspberry Pi.

We got the communication to send messages internally sending mqtt messages through localhost.
Listener:

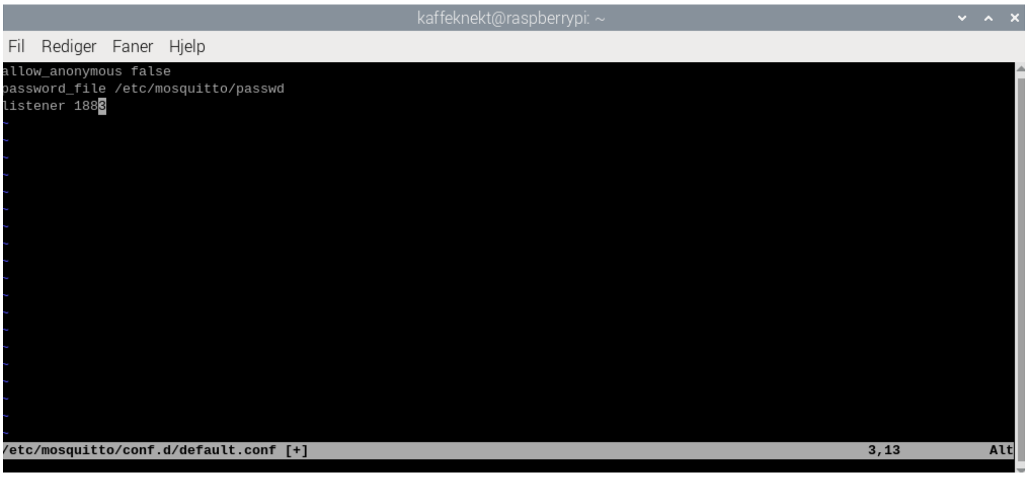


Figure C.1.16: Local listener.

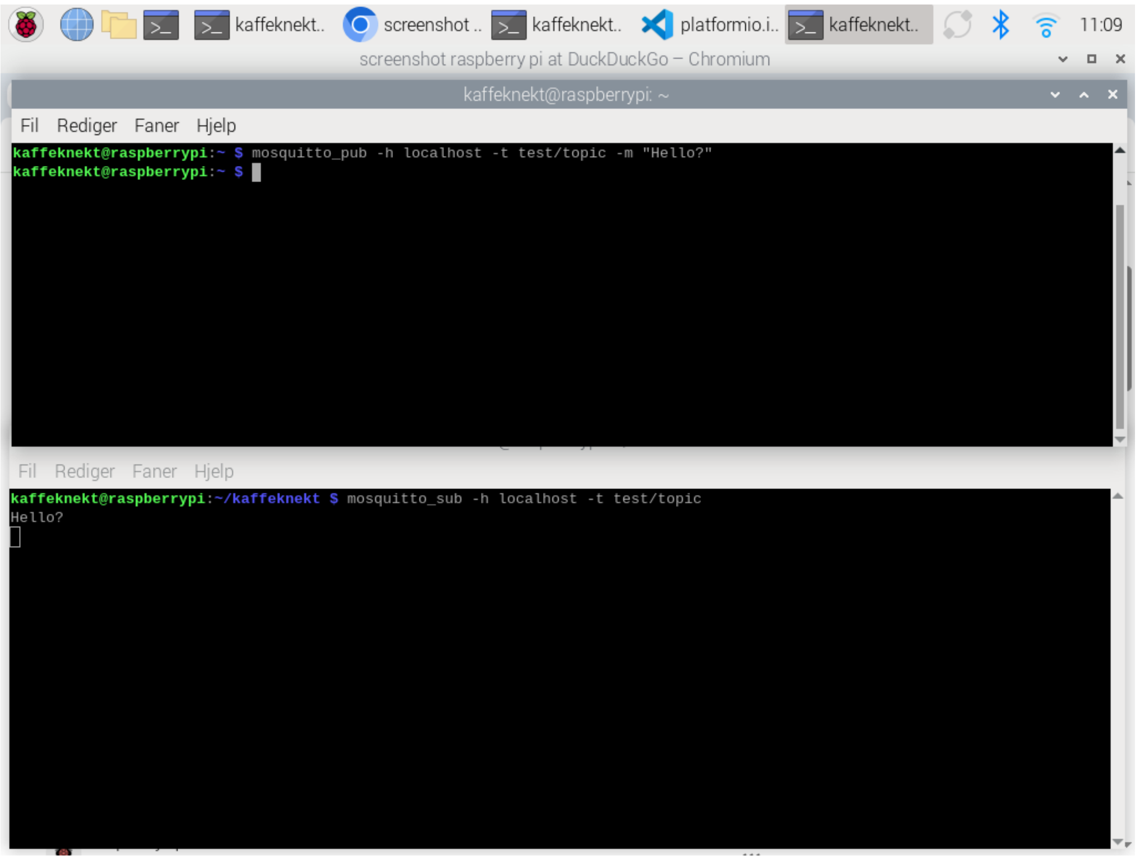


Figure C.1.17: Sending and receiving MQTT

[13] https://github.com/martintara/kaffeknekt/blob/martin/mqtt_test/src/main.cpp Copy of this file also on USB.

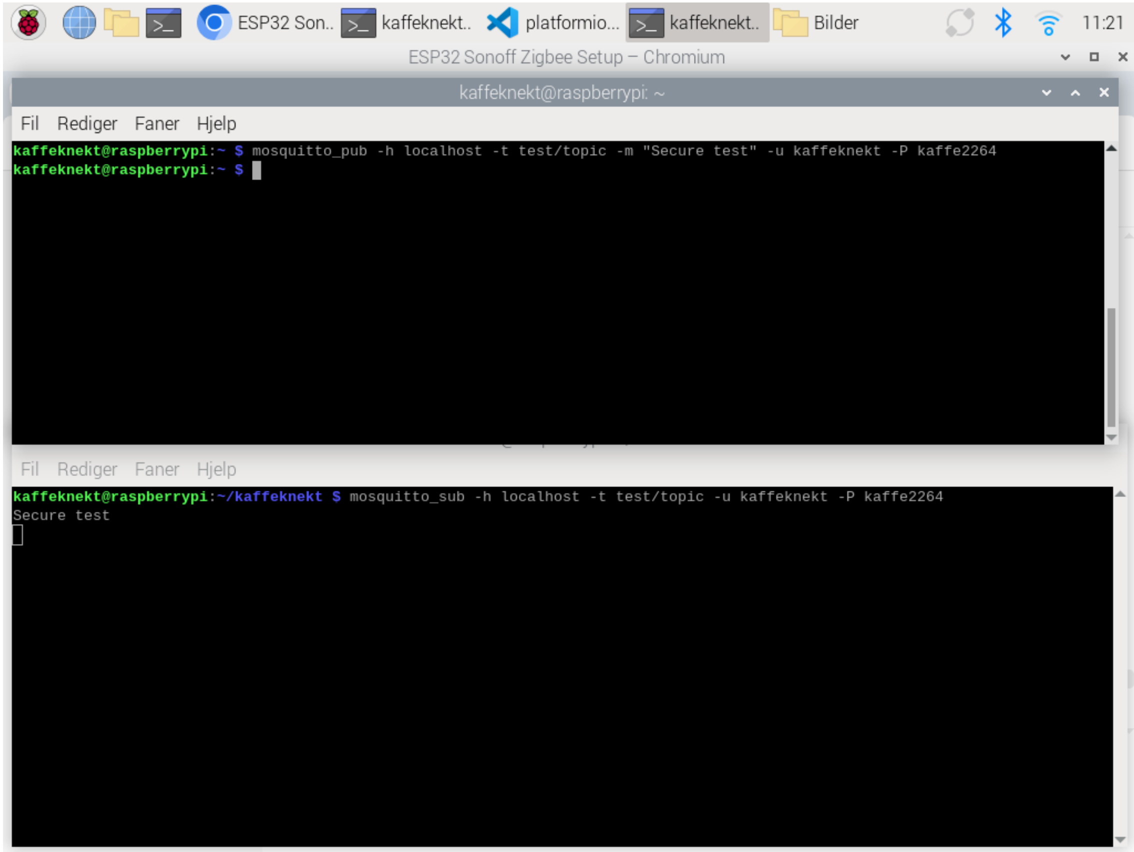


Figure C.1.18: Password protected

After a lot of failed tinkering with trying to get the ESP32 to connect with a local hope. We had a glimmer of hope after installing RaspAP (wifi hotspot software).

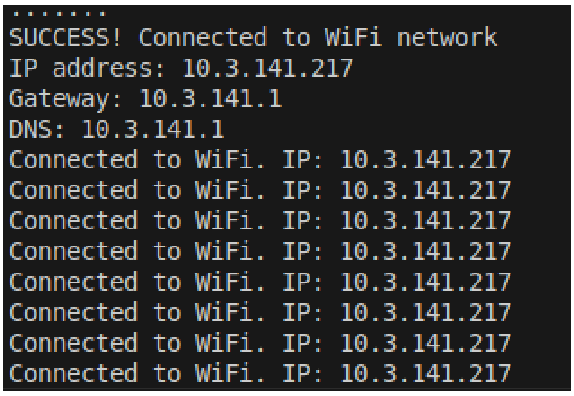


Figure C.1.19: MQTT Success

RaspAP bloated the Raspberry Pi. First of all we could not connect to the internet. (needed in our development workflow).
After rebooting the Raspberry Pi we discovered it took forever to boot. It is loading hundreds of packages.

Tinkering this much with the config files of the networking functionality of the Pi made it impossible to restore to default behavior.

Luckily we had had the foresight to make snapshots of the system, so we did a full system restore using Timeshift:
Conclusion: Test failed.

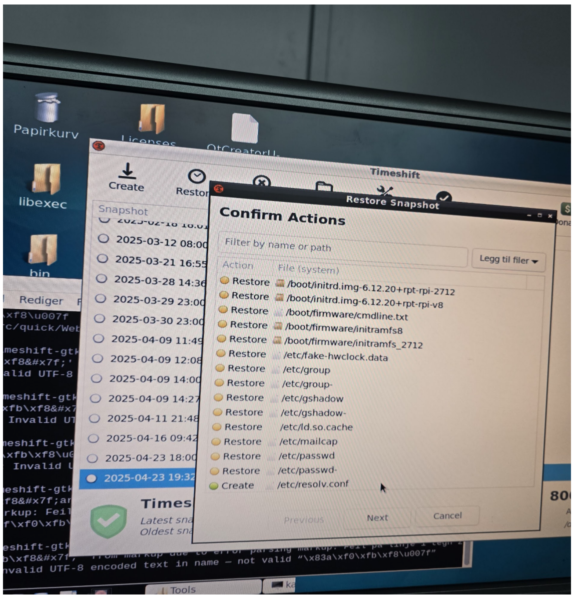


Figure C.1.20: System restoration using Timeshift

Conclusion: Test failed.
Notes: Putting MQTT on hold.

Unit test: T-ESP32-2.2 - Pressure sensor testing (Appx. A.4)

Picking up where I left off yesterday. Last session we had to abort because of leakage.
Datasheet: NPI-19 Series Digital Pressure Sensor I2C [58].
Code used for testing: GitHub [14]

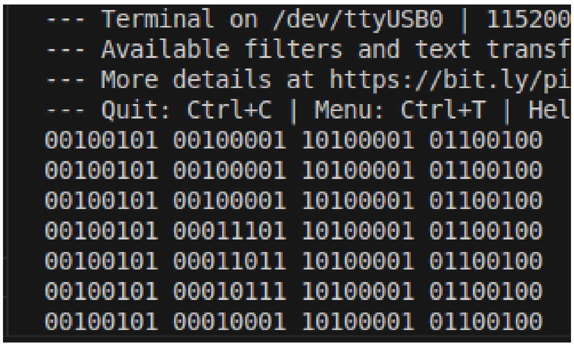


Figure C.1.21: Terminal readout: binary sensor data

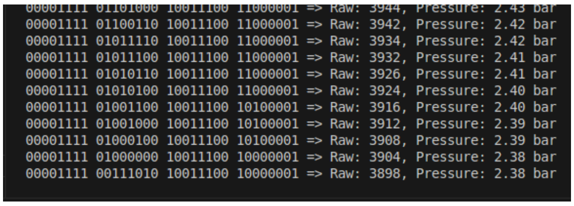


Figure C.1.22: Terminal readout: Pressure detected.

Conclusion: Test passes. Seems like we are getting accurate readings. It gets very close to zero when we turn off the coffee machine.

Notes: Our hope for finding 9 bars in the current position of the system faded after testing. The pressure builds gradually when system is idle, and it drops when starting brewing. It does not have the exact same range of values dependent on how much pressure has built up. But it usually gets to around 4 bar during brewing.

[14] https://github.com/martintara/kaffeknekt/blob/martin/trykk_test/src/main.cpp Copy of this file also on USB.

April 23, 2025

Had another attempt on setting up wifi hotspot. Messed up network configurations again. Had to do full system restore again. Putting MQTT on hold, taking too much time, it is not our highest priority.

Did some more pressure sensor testing, observing the system. When we observed the coffee machine in the search for finding a trigger it seems that we can differentiate the pressure during brewing and the pressure that gets built up when only refilling the tank.

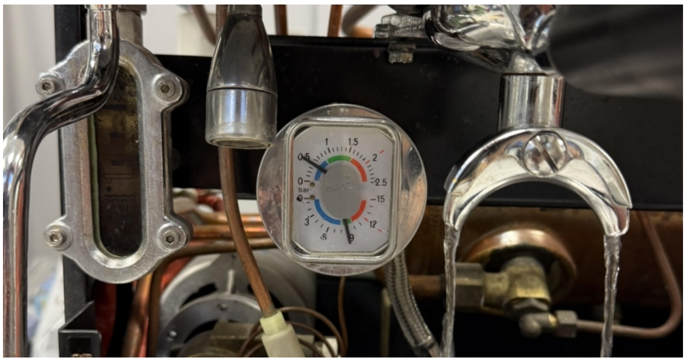


Figure C.1.23: Pressure when brewing.

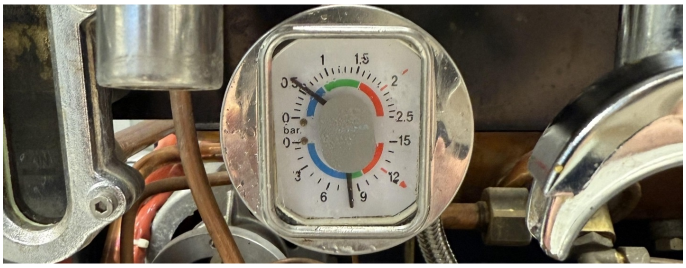


Figure C.1.24: Pressure when pump kicks in.

The two different pressure levels observed is 1. machine is brewing, 2. tank gets low and the pump starts pumping fresh water. We should be able to distinguish these levels and use the 8 bars as indicator for when the machine is currently brewing.
Though in theory this should work, we can not find the correlation with the pressure dropping in the boiler and the readouts we get from the sensor.

Conclusion: We have to look into other means on finding a trigger that indicates brewing.

April 27, 2025

Unit test: T-ESP32-2.3 - Bluetooth testing (Appx. A.4)

We have working code for each individual code now. Added AC current sensor to the combined testing code. While the use of AI has been helpful in understanding the sensors and helping creating testcode, the complexity in combining the sensors is now increasing to such extent that it starting to struggle. We have to take a step back and manually rebuild the code step by step and test for each step.

Since figuring out MQTT took too much time and we had to abort, we wanted to check if bluetooth can be a viable option. We made a quick test with generating code on ESP32 and a python script with claude ai to observe if I can establish the connection.

Code used for testing: GitHub [15] and GitHub [16]
Terminal readout:

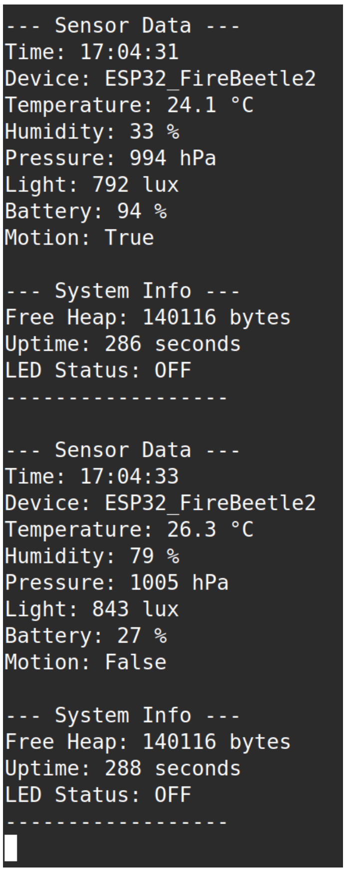


Figure C.1.25: Bluetooth testing.

Conclusion: Test passes. Established connection and it seems stable. (the sensor data is random gibberish but the purpose of the test was to check if i could send data reliably over bluetooth. Potential candidate for defining a wireless interface, but will first complete the implementation using usb connection.

[15] https://github.com/martintara/kaffeknekt/blob/martin/bluetooth_test/src/main.cppCopy of this file also on USB.

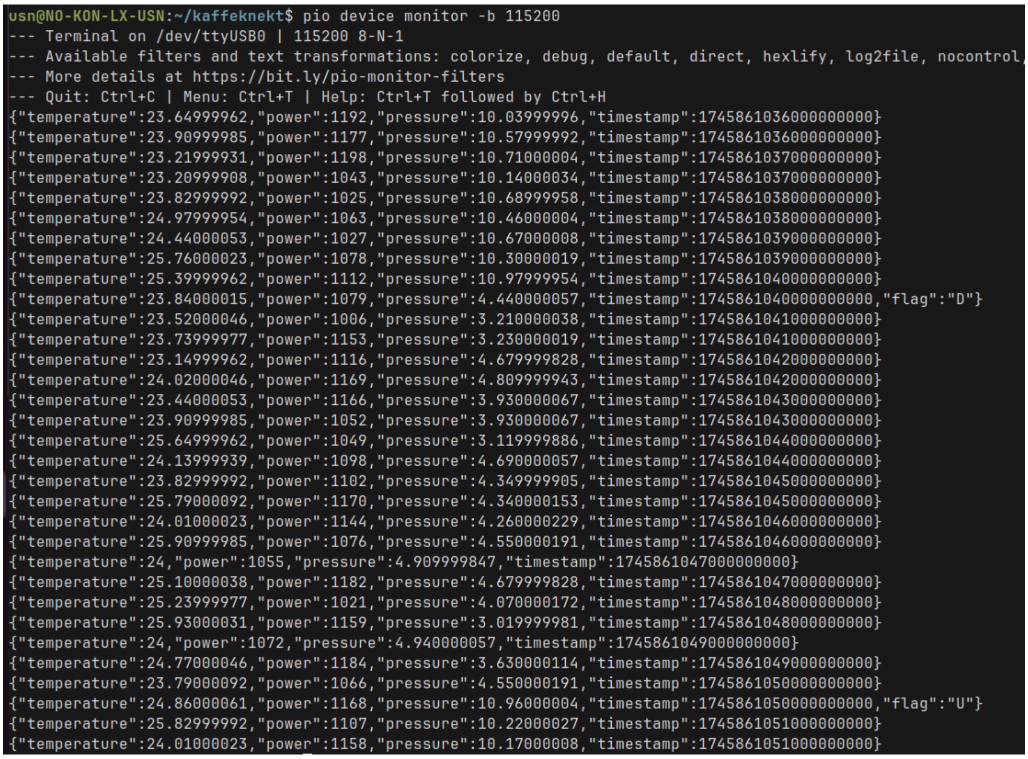
[16] https://github.com/martintara/kaffeknekt/blob/martin/bluetooth_test/esp32_bluetooth.py Copy of this file also on USB.

April 28, 2025

Started working on implementing a combined test but this time without the help of ai. We start off combining tasks one by one but something goes horribly wrong.
We get this code to work: GitHub (history version) [17]
We should break the tasks into different files. And observe when errors gets introduced.

Since we do not have functioning code yet, we made a program for ESP32 that simulates values close to what we have observed. Testing can now be commenced in other areas of development without being dependent on functional sensor code. Used Claude AI. A mistake we did was making the simulated temperature suddenly rise or drop. We meant to implement this behavior for pressure. Instructing claudeai to swap this functionality worked with grace. Code for random generator script: GitHub (history version) [18]

Conclusion: Have to read more on tasking and figure out a structured way to do this. Note: Random generator script simulates the wanted behavior from ESP32 in such manner that the rest of the team does not have to wait for functional sensor code to do their testing. Findings: When making the random generator script we thought of the wanted behavior. An idea of using a flag variable came up. This flag will indicate if the system is currently brewing coffee or not.



```
usn@NO-KON-LX-USN:~/kaffeknekt$ pio device monitor -b 115200
--- Terminal on /dev/ttyUSB0 | 115200 8-N-1
--- Available filters and text transformations: colorize, debug, default, direct, hexlify, log2file, nocontrol,
--- More details at https://bit.ly/pio-monitor-filters
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
{"temperature":23.64999962,"power":1192,"pressure":10.03999996,"timestamp":1745861036000000000}
{"temperature":23.90999985,"power":1177,"pressure":10.57999992,"timestamp":1745861036000000000}
{"temperature":23.21999931,"power":1198,"pressure":10.71000004,"timestamp":1745861037000000000}
{"temperature":23.20999908,"power":1043,"pressure":10.14000034,"timestamp":1745861037000000000}
{"temperature":23.82999992,"power":1025,"pressure":10.68999958,"timestamp":1745861038000000000}
{"temperature":24.97999954,"power":1063,"pressure":10.46000004,"timestamp":1745861038000000000}
{"temperature":24.44000053,"power":1027,"pressure":10.67000008,"timestamp":1745861039000000000}
{"temperature":25.76000023,"power":1078,"pressure":10.30000019,"timestamp":1745861039000000000}
{"temperature":25.39999962,"power":1112,"pressure":10.97999954,"timestamp":1745861040000000000}
{"temperature":23.84000015,"power":1079,"pressure":4.440000057,"timestamp":1745861040000000000,"flag":"D"}
{"temperature":23.52000046,"power":1006,"pressure":3.210000038,"timestamp":1745861041000000000}
{"temperature":23.73999977,"power":1153,"pressure":3.230000019,"timestamp":1745861041000000000}
{"temperature":23.14999962,"power":1116,"pressure":4.679999828,"timestamp":1745861042000000000}
{"temperature":24.02000046,"power":1169,"pressure":4.809999943,"timestamp":1745861042000000000}
{"temperature":23.44000053,"power":1166,"pressure":3.930000067,"timestamp":1745861043000000000}
{"temperature":23.90999985,"power":1052,"pressure":3.930000067,"timestamp":1745861043000000000}
{"temperature":25.64999962,"power":1049,"pressure":3.119999886,"timestamp":1745861044000000000}
{"temperature":24.13999939,"power":1098,"pressure":4.690000057,"timestamp":1745861044000000000}
{"temperature":23.82999992,"power":1102,"pressure":4.349999905,"timestamp":1745861045000000000}
{"temperature":25.79000092,"power":1170,"pressure":4.340000153,"timestamp":1745861045000000000}
{"temperature":24.01000023,"power":1144,"pressure":4.260000229,"timestamp":1745861046000000000}
{"temperature":25.90999985,"power":1076,"pressure":4.550000191,"timestamp":1745861046000000000}
{"temperature":24,"power":1055,"pressure":4.909999847,"timestamp":1745861047000000000}
{"temperature":25.10000038,"power":1182,"pressure":4.679999828,"timestamp":1745861047000000000}
{"temperature":25.23999977,"power":1021,"pressure":4.070000172,"timestamp":1745861048000000000}
{"temperature":25.93000031,"power":1159,"pressure":3.019999981,"timestamp":1745861048000000000}
{"temperature":24,"power":1072,"pressure":4.940000057,"timestamp":1745861049000000000}
{"temperature":24.77000046,"power":1184,"pressure":3.630000114,"timestamp":1745861049000000000}
{"temperature":23.79000092,"power":1066,"pressure":4.550000191,"timestamp":1745861050000000000}
{"temperature":24.86000061,"power":1168,"pressure":10.96000004,"timestamp":1745861050000000000,"flag":"U"}
{"temperature":25.82999992,"power":1107,"pressure":10.22000027,"timestamp":1745861051000000000}
{"temperature":24.01000023,"power":1158,"pressure":10.17000008,"timestamp":1745861051000000000}
```

Figure C.1.26: Script simulating sensor readouts.

[17] https://github.com/martintara/kaffeknekt/blob/4e5d57021a7406fd65095720412f9ef87f4749f0/combined_test2/src/main.cpp Copy of this file also on USB.
[18] https://github.com/martintara/kaffeknekt/blob/4e5d57021a7406fd65095720412f9ef87f4749f0/combined_test2/src/main.cpp Copy of this file also on USB.

May 2, 2025

Unit test: T-ESP32-2.4 - RTC interface testing (Appx. A.4)

ESP32 Code used for testing: GitHub [19]
Python script used for sending a manually set time (defined in variables): GitHub [20]
The two tasks are based on earlier tested code. One task for receiving data over serial. One task for sending data over serial.

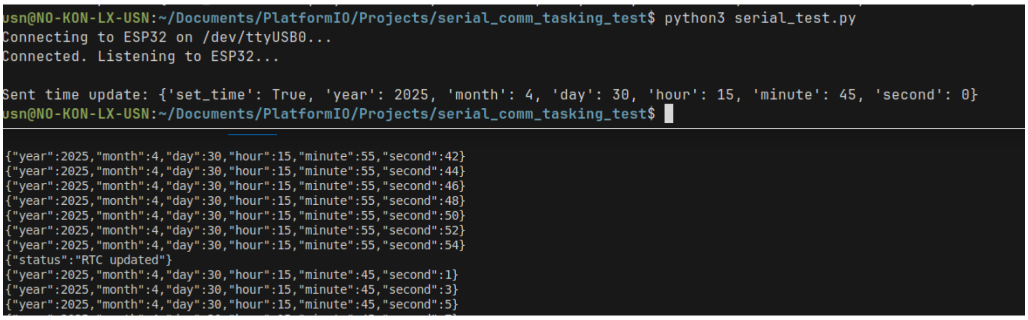


Figure C.1.27: Terminal readout.

Conclusion: Test passes. Date got sent and registered. ESP32 keeps sending data.

[19] https://github.com/martintara/kaffeknekt/blob/martin/serial_comm_tasking_test/src/main.cpp Copy of this file also on USB.
[20] https://github.com/martintara/kaffeknekt/blob/martin/serial_comm_tasking_test/serial_test.py Copy of this file also on USB.

Unit test: T-ESP32-2.5 - Leading zeros test (Appx. A.4)
Further testing: Was asked from GUI dev if leading 0's will work (as in month 01, hour 01 etc.) Testing with leading 0.
Code used for testing: GitHub [21]

```
usn@NO-KON-LX-USN:~/kaffeknekt/serial_comm_tasking_test$ python3 serial_test.py
  File "/home/usn/kaffeknekt/serial_comm_tasking_test/serial_test.py", line 12
    "month": 04,
             ^
SyntaxError: leading zeros in decimal integer literals are not permitted; use an 0o prefix for octal integers
usn@NO-KON-LX-USN:~/kaffeknekt/serial_comm_tasking_test$
```

Figure C.1.28: Crash using leading zero.

As suspected this crashed. An easy fix was using as_int [4].

```
int year = doc["year"].as<int>();
int month = doc["month"].as<int>();
int day = doc["day"].as<int>();
int hour = doc["hour"].as<int>();
int minute = doc["minute"].as<int>();
int second = doc["second"].as<int>();
```

Figure C.1.29: Using as int.

Conclusion: Test passes. Leading zero works now.
Note: No need to consider any more validation of the string since the values will be set from the GUI, not user input from keyboard. This was just for convenience for GUI dev.

[21] https://github.com/martintara/kaffeknekt/blob/martin/serial_comm_tasking_test/src/main.cpp Copy of this file also on USB.

May 5, 2025

Unit test: T-ESP32-2.6 - Combining sensors test (Appx. A.4)

Code used for testing: GitHub (history version) [22]

First we thought there was no contact with sensor. But that did not really make sense. We tried testing the pressure sensor with the standalone code which we have tested and confirmed to work. Sensor works with this code. We then added prinln statements to debug. It prints the raw data and conversion to bar, meaning we get sensor readouts.
Conclusion: Problem was the float value gets stored in an int and is rounded to 0.
Fix: Changed int to float

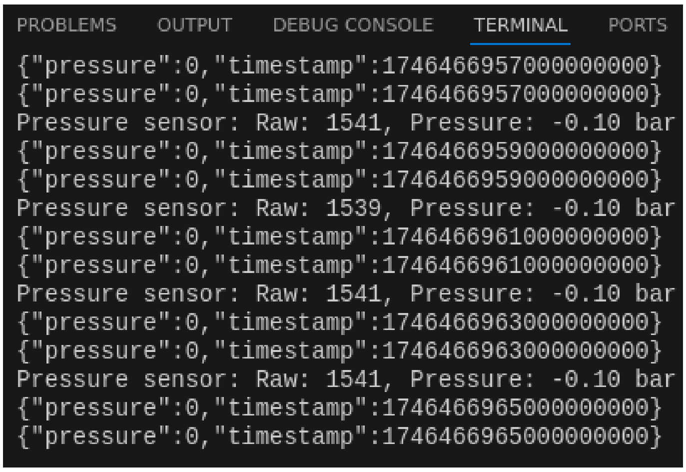


Figure C.1.30: Float issue.

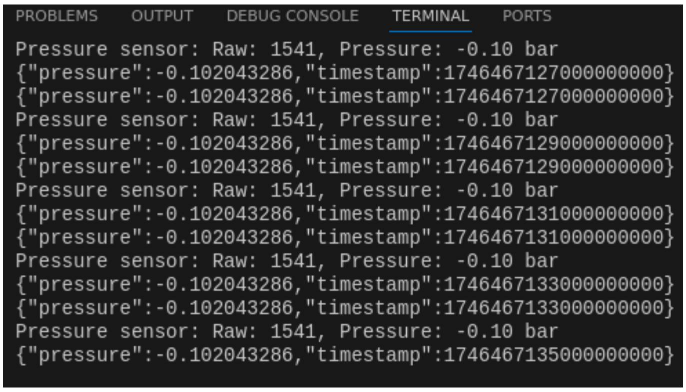


Figure C.1.31: Fixed by changing int to float.

Conclusion: Bug fixed. Test passes.

[22] https://github.com/martintara/kaffeknekt/tree/08f33bc8212c93c9b0d330fe3669bde35571627d/combined_test2
Copy of these files also on USB.

Unit test: T-ESP32-2.7 - Testing new temperature sensor (Appx. A.4)

Code used for testing: GitHub (history version) [23]

Connected sensor to ESP32. Getting no response.
Only instructions from Adafruit guide [1] is to download the test code from the Arduino IDE for usage on an Arduino. Cannot find code anywhere else.
Found our trusty Arduino Mega. Downloaded and installed arduino IDE and tried the example code: GitHub [24]
At first we did not get readouts here either. We had been instructed by electrical team that this was a 4wire sensor. Turned out the sensor we got is a 2wire configuration. We discovered this by reading on the packaging bag. Consulting the datasheet the soultion then is to short the two terminals on the chip. This made the arduino give proper readouts.

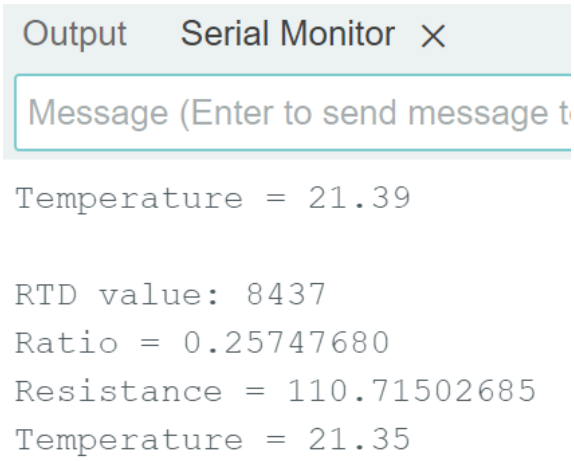


Figure C.1.32: Correct readouts on Arduino Mega.

After trying the example code on ESP32 we had high hopes but we did not get readouts.

We checked the wiring again on the ESP32. The mistake was discovered. We had plugged the chip select into pin 21, which at first glance was empty and unused (it is not specifically marked SDA either). After consulting the pinout diagram from the datasheet we discovered that pin 21 is found on both sides of the esp 32.

[23] https://github.com/martintara/kaffeknekt/blob/08f33bc8212c93c9b0d330fe3669bde35571627d/combined_test2/src/TaskTemperature.cpp Copy of this file also on USB.
[24] https://github.com/martintara/kaffeknekt/blob/martin/temp_test_arduino/max31865.ino Copy of this file also on USB.

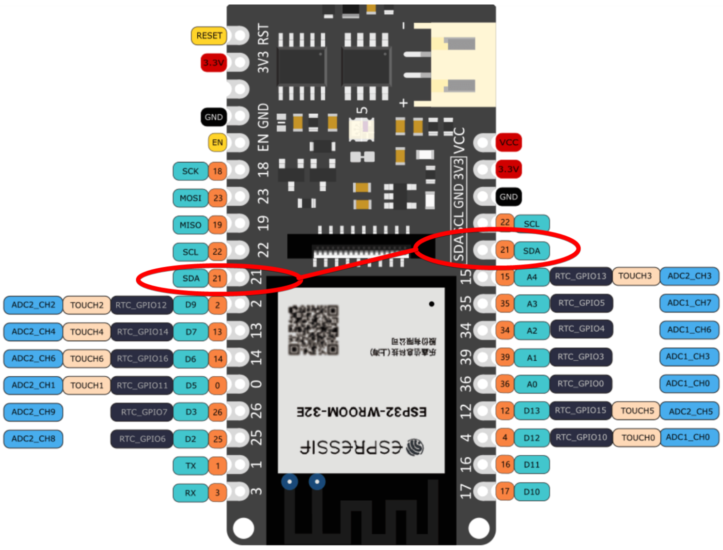
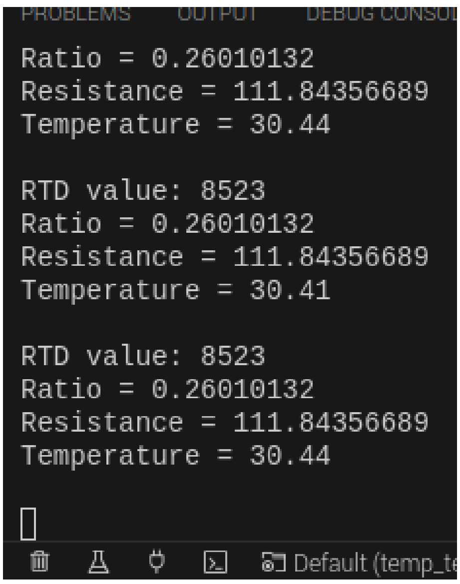


Figure C.1.33: Discovered SDA is on both sides of ESP32.

Actually many things happened here. We were not thorough enough when reading datasheets and we were not methodical enough when testing the sensor on ESP32. We were to eager to test the sensor combined with other sensors. We got confused when we did not get readouts and we could not isolate the error.

After changing the chip select to a regular gpio pin (pin 14 in our case) we got the sensor to work on ESP32.
The working testcode: GitHub (history version)[25]
Terminal readout:



```
Ratio = 0.26010132
Resistance = 111.84356689
Temperature = 30.44

RTD value: 8523
Ratio = 0.26010132
Resistance = 111.84356689
Temperature = 30.41

RTD value: 8523
Ratio = 0.26010132
Resistance = 111.84356689
Temperature = 30.44
```

Figure C.1.34: Temperature showing 30c when it is 24c room temperature.

This was room temperature. Since the sensor is linear we just offsetted the value with a constant.

[25]https://github.com/martintara/kaffeknekt/blob/08f33bc8212c93c9b0d330fe3669bde35571627d/combined_test2/src/TaskTemperature.cpp Copy of this file also on USB.

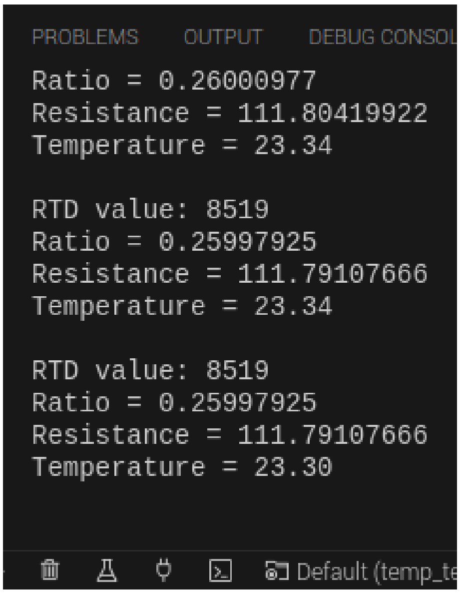


Figure C.1.35: Calibrating by offsetting the readout.

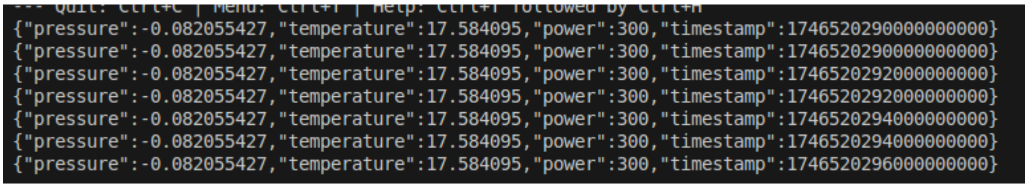
Conclusion: Test passes. This made the temperature readout align with the room temperature thermometer.

May 6, 2025

Unit test: T-ESP32-2.8 - Combined test with static data (Appx. A.4)

Implemented ACPower task with static dummy data.

Code used for testing: GitHub (history version) [26]



--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
{"pressure":-0.082055427,"temperature":17.584095,"power":300,"timestamp":1746520290000000000}
{"pressure":-0.082055427,"temperature":17.584095,"power":300,"timestamp":1746520290000000000}
{"pressure":-0.082055427,"temperature":17.584095,"power":300,"timestamp":1746520292000000000}
{"pressure":-0.082055427,"temperature":17.584095,"power":300,"timestamp":1746520292000000000}
{"pressure":-0.082055427,"temperature":17.584095,"power":300,"timestamp":1746520294000000000}
{"pressure":-0.082055427,"temperature":17.584095,"power":300,"timestamp":1746520294000000000}
{"pressure":-0.082055427,"temperature":17.584095,"power":300,"timestamp":1746520296000000000}

Figure C.1.36: ACPower task with static data.

Conclusion: Test passes. First test with static data successfully adds power as a field in the shared data.

[26] https://github.com/martintara/kaffeknekt/blob/ffa60c247f7527a502ec11ae527a3d49b11f0639/combined_test3/src/TaskACPower.cpp Copy of this file also on USB.

Unit test: T-ESP32-2.9 - Implementation of AC current sensor (Appx. A.4)

Attempt at implementing the ac current sensor to combined test.
Code used for testing: GitHub (history version) [27]
Terminal output:

```
Temperature sensor task starting...
{"pressure":-0.075743474,"temperature":0,"power":0,"timestamp":1746526241000000000}

assert failed: xQueueGenericSend queue.c:832 (pxQueue->pcHead != ((void *)0) || pxQueue->u.xSemaphore.xMutexHolder == ((void *)0) || pxQueue->u.xSemaphore.xMutexHolder =
= xTaskGetCurrentTaskHandle())


Backtrace: 0x400839fd:0x3ffaf5b0 0x40089819:0x3ffaf5d0 0x4008e931:0x3ffaf5f0 0x4008a01e:0x3ffaf720 0x400d36ff:0x3ffaf760 0x400d3941:0x3ffaf780 0x400d47c0:0x3ffaf7a0 0x40
0d47dd:0x3ffaf7c0 0x400d48f0:0x3ffaf7f0 0x400d18fd:0x3ffaf810
```

Figure C.1.37: Assert error. Unsure how to proceed.

Conclusion: Test failed. Used a lot of time. We suspect some kind of address conflicts since error only happens when we introduce a new sensor.
Notes: I get mutex errors. But the mutex I use only locks down the shared data. Errors only occur when i introduce the sensor to the code.

[27] https://github.com/martintara/kaffeknekt/tree/230badd950372938254fdfff359fd3c66bc4e35a/combined_test3/src Copy of these files also on USB.

May 7, 2025

Unit test: T-ESP32-3.0 - Implementation of static temperature data (Appx. A.4)

Code used for testing: GitHub (history version) [28]


```
{"temperature":300,"power":9.077880859,"timestamp":1746626052000000000,"flag":0}
{"temperature":300,"power":9.077880859,"timestamp":1746626052000000000,"flag":0}
{"temperature":300,"power":9.077880859,"timestamp":1746626052000000000,"flag":0}
{"temperature":300,"power":9.077880859,"timestamp":1746626052000000000,"flag":0}
{"temperature":300,"power":9.077880859,"timestamp":1746626052000000000,"flag":0}
{"temperature":300,"power":9.077880859,"timestamp":1746626052000000000,"flag":0}
{"temperature":300,"power":9.077880859,"timestamp":1746626052000000000,"flag":0}
{"temperature":300,"power":9.077880859,"timestamp":1746626052000000000,"flag":0}
{"temperature":300,"power":9.077880859,"timestamp":1746626052000000000,"flag":0}
{"temperature":300,"power":9.077880859,"timestamp":1746626052000000000,"flag":0}
```


Figure C.1.38: Temperature task with static data.

Conclusion: Test passes. Static data shows up as new field in the JSON structure.

Unit test: T-ESP32-3.1 - Implementation of temperature sensor

Using the currently functioning code for temperature sensor and implementing it as a task.


```
{"temperature":20.35871124,"power":9.077880859,"timestamp":1746626691000000000,"fl
{"temperature":20.35871124,"power":9.077880859,"timestamp":1746626691000000000,"fl
{"temperature":20.35871124,"power":9.077880859,"timestamp":1746626691000000000,"fl
{"temperature":20.35871124,"power":9.077880859,"timestamp":1746626691000000000,"fl
{"temperature":20.35871124,"power":9.077880859,"timestamp":1746626691000000000,"fl
{"temperature":20.35871124,"power":9.077880859,"timestamp":1746626691000000000,"fl
{"temperature":20.35871124,"power":9.077880859,"timestamp":1746626691000000000,"fl
{"temperature":20.35871124,"power":9.077880859,"timestamp":1746626691000000000,"fl
{"temperature":20.35871124,"power":9.077880859,"timestamp":1746626691000000000,"fl
{"temperature":20.35871124,"power":9.077880859,"timestamp":1746626691000000000,"fl
```


Figure C.1.39: Implementation of actual temperature sensor.

Conclusion: Test passes. Getting the the expected sensor readout in the JSON structure.

Unit test: T-ESP32-3.2 - Implementation of static pressure data (Appx. A.4)

Implemented pressure task with static data:


```
{"pressure":10,"temperature":16.97530746,"power":9.077880859,"timestamp":1746627777000000000,"flag":0}
{"pressure":10,"temperature":16.97530746,"power":9.077880859,"timestamp":1746627777000000000,"flag":0}
{"pressure":10,"temperature":16.97530746,"power":9.077880859,"timestamp":1746627777000000000,"flag":0}
{"pressure":10,"temperature":16.97530746,"power":9.077880859,"timestamp":1746627777000000000,"flag":0}
{"pressure":10,"temperature":16.97530746,"power":9.077880859,"timestamp":1746627777000000000,"flag":0}
{"pressure":10,"temperature":16.97530746,"power":9.077880859,"timestamp":1746627777000000000,"flag":0}
{"pressure":10,"temperature":16.97530746,"power":9.077880859,"timestamp":1746627777000000000,"flag":0}
{"pressure":10,"temperature":16.97530746,"power":9.077880859,"timestamp":1746627777000000000,"flag":0}
{"pressure":10,"temperature":16.97530746,"power":9.077880859,"timestamp":1746627777000000000,"flag":0}
{"pressure":10,"temperature":16.97530746,"power":9.077880859,"timestamp":1746627777000000000,"flag":0}
```


Figure C.1.40: Pressure task with static data.

Conclusion: Test passes. Static data shows up as new field in the JSON structure

Unit test: T-ESP32-3.3 - Implementation of pressure sensor (Appx. A.4)

Implemented pressure sensor instead of static data:

[28] https://github.com/martintara/kaffeknekt/tree/852e6ed460069c4ed0c86f58a004ff6a3bc70c5b/combined_test3
Copy of these files also on USB.

```
{"pressure":-0.065223545,"temperature":16.8739109,"power":0,"timestamp":1746627935000000000,"flag":0}
{"pressure":-0.065223545,"temperature":16.8739109,"power":0,"timestamp":1746627935000000000,"flag":0}
{"pressure":-0.065223545,"temperature":16.8739109,"power":0,"timestamp":1746627935000000000,"flag":0}
{"pressure":-0.065223545,"temperature":16.8739109,"power":0,"timestamp":1746627935000000000,"flag":0}
{"pressure":-0.065223545,"temperature":16.8739109,"power":0,"timestamp":1746627935000000000,"flag":0}

assert failed: xQueueGenericSend queue.c:832 (pxQueue->pcHead != ((void *)0) || pxQueue->u.xSemaphore.xMutexHolder
= xTaskGetCurrentTaskHandle())


Backtrace: 0x400839fd:0x3ffaf520 0x40089819:0x3ffaf540 0x4008e931:0x3ffaf560 0x4008a01e:0x3ffaf690 0x400d2c43:0x3f
```


Figure C.1.41: Assert fail again. Suspecting I2C conflict.

Conclusion: Test failed. We get the readouts but suddenly crashes.

Unit test: T-ESP32-3.4 - Implementation of pressure sensor take 2 (Appx. A.4)

Locking down the i2c functionality with mutex:


```
{"pressure":-0.077847458,"temperature":16.6372509,"power":9.077880859,"timestamp":1746628377000000000,"flag":0}
{"pressure":-0.077847458,"temperature":16.6372509,"power":9.077880859,"timestamp":1746628377000000000,"flag":0}
{"pressure":-0.077847458,"temperature":16.6372509,"power":9.077880859,"timestamp":1746628377000000000,"flag":0}
{"pressure":-0.077847458,"temperature":16.6372509,"power":9.077880859,"timestamp":1746628377000000000,"flag":0}
{"pressure":-0.077847458,"temperature":16.6372509,"power":9.077880859,"timestamp":1746628377000000000,"flag":0}
{"pressure":-0.077847458,"temperature":16.6372509,"power":9.077880859,"timestamp":1746628377000000000,"flag":0}
{"pressure":-0.077847458,"temperature":16.6372509,"power":9.077880859,"timestamp":1746628377000000000,"flag":0}
{"pressure":-0.077847458,"temperature":16.6372509,"power":9.077880859,"timestamp":1746628377000000000,"flag":0}
{"pressure":-0.077847458,"temperature":16.6372509,"power":9.077880859,"timestamp":1746628377000000000,"flag":0}
{"pressure":-0.077847458,"temperature":16.6372509,"power":9.077880859,"timestamp":1746628377000000000,"flag":0}
```


Figure C.1.42: Pressure task implemented correctly.

Conclusion: Test passes. No errors. Getting expected readouts.

Note: Still noticing timing issue. Multiple readings within same second, and sometimes it skips a beat.


```
{"power":9.077880859,"timestamp":1746606294000000000}
{"power":9.077880859,"timestamp":1746606294000000000}
{"power":9.077880859,"timestamp":1746606296000000000}
{"power":9.077880859,"timestamp":1746606296000000000}
{"power":9.077880859,"timestamp":1746606296000000000}
{"power":9.077880859,"timestamp":1746606297000000000}
{"power":9.077880859,"timestamp":1746606297000000000}
{"power":9.077880859,"timestamp":1746606297000000000}
{"power":9.077880859,"timestamp":1746606299000000000}
{"power":9.077880859,"timestamp":1746606299000000000}
{"power":9.077880859,"timestamp":1746606299000000000}
```


Figure C.1.43: Timer skipping a beat.

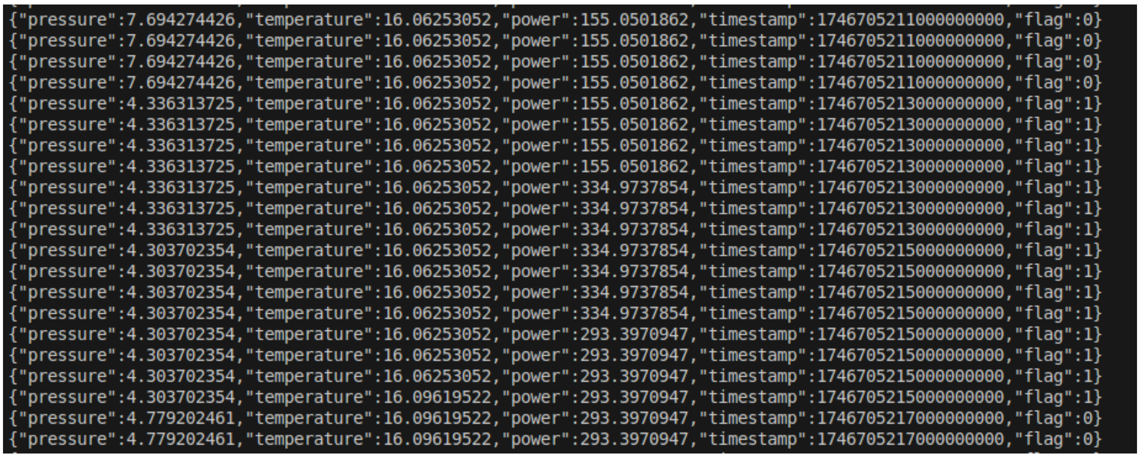
Note: While doing this much testing on sensors and occupying the machine. To not prevent other developers from performing tests, we modified the random generator script to align with the current wanted behavior (flagging 1 when brewing, and 0 when not): GitHub (history version) [29]

[29]https://github.com/martintara/kaffeknekt/blob/488a8bd77872f6e66f14222b7c72046b7fd3377a/random_generator/src/main.cpp Copy of this file also on USB.

May 8, 2025

Unit test: T-ESP32-3.5 - Testing of brew trigger (Appx. A.4)

After exploring currents running in the system we found a wire that runs a control current to the solenoid. Highly likely a candidate to use as indicator of current brewing status.
Code used for testing: GitHub (history version) [30]



{"pressure":7.694274426,"temperature":16.06253052,"power":155.0501862,"timestamp":1746705211000000000,"flag":0}
{"pressure":7.694274426,"temperature":16.06253052,"power":155.0501862,"timestamp":1746705211000000000,"flag":0}
{"pressure":7.694274426,"temperature":16.06253052,"power":155.0501862,"timestamp":1746705211000000000,"flag":0}
{"pressure":7.694274426,"temperature":16.06253052,"power":155.0501862,"timestamp":1746705211000000000,"flag":0}
{"pressure":4.336313725,"temperature":16.06253052,"power":155.0501862,"timestamp":1746705213000000000,"flag":1}
{"pressure":4.336313725,"temperature":16.06253052,"power":155.0501862,"timestamp":1746705213000000000,"flag":1}
{"pressure":4.336313725,"temperature":16.06253052,"power":155.0501862,"timestamp":1746705213000000000,"flag":1}
{"pressure":4.336313725,"temperature":16.06253052,"power":155.0501862,"timestamp":1746705213000000000,"flag":1}
{"pressure":4.336313725,"temperature":16.06253052,"power":334.9737854,"timestamp":1746705213000000000,"flag":1}
{"pressure":4.336313725,"temperature":16.06253052,"power":334.9737854,"timestamp":1746705213000000000,"flag":1}
{"pressure":4.336313725,"temperature":16.06253052,"power":334.9737854,"timestamp":1746705213000000000,"flag":1}
{"pressure":4.303702354,"temperature":16.06253052,"power":334.9737854,"timestamp":1746705215000000000,"flag":1}
{"pressure":4.303702354,"temperature":16.06253052,"power":334.9737854,"timestamp":1746705215000000000,"flag":1}
{"pressure":4.303702354,"temperature":16.06253052,"power":334.9737854,"timestamp":1746705215000000000,"flag":1}
{"pressure":4.303702354,"temperature":16.06253052,"power":334.9737854,"timestamp":1746705215000000000,"flag":1}
{"pressure":4.303702354,"temperature":16.06253052,"power":293.3970947,"timestamp":1746705215000000000,"flag":1}
{"pressure":4.303702354,"temperature":16.06253052,"power":293.3970947,"timestamp":1746705215000000000,"flag":1}
{"pressure":4.303702354,"temperature":16.06253052,"power":293.3970947,"timestamp":1746705215000000000,"flag":1}
{"pressure":4.303702354,"temperature":16.09619522,"power":293.3970947,"timestamp":1746705215000000000,"flag":1}
{"pressure":4.779202461,"temperature":16.09619522,"power":293.3970947,"timestamp":1746705217000000000,"flag":0}
{"pressure":4.779202461,"temperature":16.09619522,"power":293.3970947,"timestamp":1746705217000000000,"flag":0}

Figure C.1.44: Brewing flag changes when brewing.

Test passes. Flag variable changes to 1 when brewing and 0 when not brewing. Note: There is a slight delay. Could be improved with timing tasks. Maybe separating out the watt meters into different tasks, give priority to brew trigger current sensor. Currently not a requirement. Note: Timing still skips a beat, should look into calculate time task next.

[30] https://github.com/martintara/kaffeknekt/blob/ad28862f7e0f0e25f8ed769ed991be672808ddbd/combined_test3/src/TaskACPower.cpp Copy of this file also on USB.

May 9, 2025

Unit test: T-ESP32-3.6 - Testing new timer functionality (Appx. A.4)

After discussing the current implementation of how the ESP32 keeps track of time, we realized that this is not a good solution. The problem with current solution is that esp32 is continuously checking in with the RTC and asks what time it is. The smallest time unit the RTC provides is seconds. Herin lies the culprit. Not only is this bad since we cant uniquely identify sensor readings, but also because current implementation is unnecessarily using the I2C bus (and by doing so locking it down).

Code used for testing: GitHub (history version) [31]

Testing with getting the time from RTC on boot, and from this point keep track of the time by using millis() counter on the ESP32. Also added functionality of setting the time once every 3 hours to prevent the millis() counter to drift.


```
{"pressure":-0.06311956,"temperature":18.42974472,"power":9.077880859,"timestamp":1746781229987000000,"flag":0}
{"pressure":-0.06311956,"temperature":18.49747658,"power":9.077880859,"timestamp":1746781230389000000,"flag":0}
{"pressure":-0.06311956,"temperature":18.49747658,"power":9.077880859,"timestamp":1746781230590000000,"flag":0}
{"pressure":-0.06311956,"temperature":18.49747658,"power":9.077880859,"timestamp":1746781230992000000,"flag":0}
{"pressure":-0.06311956,"temperature":18.49747658,"power":9.077880859,"timestamp":1746781231193000000,"flag":0}
{"pressure":-0.06311956,"temperature":18.49747658,"power":9.077880859,"timestamp":1746781231595000000,"flag":0}
{"pressure":-0.06311956,"temperature":18.49747658,"power":9.077880859,"timestamp":1746781231796000000,"flag":0}
```


Figure C.1.45: Correct timer behavior.

Conclusion: Test passes. We now get milliseconds as the smallest unit of time. This now uniquely identify each line of sensor readouts.
Note: It also solved the problem of the observed behavior of lines skipping a beat. Most likely because the I2C bus does not get locked down as often.

Code is now fulfilling the wanted functionality.

[31] https://github.com/martintara/kaffeknekt/blob/5167ba9dd53da734a878a01ebe2e171918d20150/combined_test3/src/TaskCalculateTime.cpp Copy of this file also on USB.

Setting Clock and Socket Implementation

C.2.1 Clock Setting Code Formulation(Qt Side) **DAB** | *MT*

Defining the Interface
To facilitate the functionality for manual time setting from the application side, it was decided that it would use the same interface as the communication between the ESP32 and the Data Handling Script. That interface being utilizing the serial port for data transfer in a JSON format. As the ESP32 looks for a "set_time" with a value of "true" when the setup of the JSON where have this structure:

{'set_time': True,
'year': yyyy,
'month': MM,
'day': dd,
'hour': HH,
'minute': mm,
'second': ss}

The second thing to look at was what limitations Qt had in terms of delivering data over a serial port. Looking through the documentation on Qt official site[71] the QtSerialPort class seemed to have all the functionality needed, but its write() function only takes in "const char*" or QByteArray so the data would need to be converted to either. QByteArray was used as the ".toUtf8" is an easy way to convert QString to QByteArray.

Code Development Process
Looking at the formatting that the QTimeEdit class utilized "yyyy-MM-dd HH:mm:ss" when representing it as a 24h clock with the Gregorian calendar. A method using the split() function was first tried where the value was added to a QString and was then split using "space" before each of the splits were again split using "-" for year, month, day and ":" for hour, minutes, seconds allowing for the extraction of each individual value. This was then set into a new QString with the necessary structure. (Arguments were used so they could easily be added) Unfortunately this did not work. Although the ESP32 received the same format structure it registered it as not a valid JSON format. After looking at the structure again a final attempt was made as it seemed that a "newline" was also added at the end, but same result.
The next attempt was to use the QJsonDocument and QJsonObject classes in Qt to set the format up as a JSON object, insert it into a JSON document and then send that as a QByteArray to see if the problem lied in the structuring of the QString. That did the trick, and when testing against the ESP32 the data was accepted.

C.2.2 Domain Socket Code Formulation **DAB** | *MT*

Defining the Interface
After a meeting with our customer, the idea of looking into using sockets for the communication between the Qt desktop and the Influx database came up. This was as a way to avoid having the Qt desktop continuously query the database when looking for the "trigger" that tells the system the brewing process has started. Because the Influx had limited socket functionality that being only able to take in information over a socket connection, another way was needed.
Under this process another shortcoming of our system came to light and one more that was known beforehand, but as a new solution was being looked at, addressing it seemed relevant. Our known issue stemmed from our current solution for real-time visualization, which was to query the database for the information continuously, leading to potential delays and creating an unnecessary load on the system. This was initially done to reduce the number of interfaces needed in the system, but with the goal of optimizing performance, a way to access the data as early as possible was looked into.
Our second issue was regarding the amount of traffic induced over the serial port. This was raised as a potential issue if the amount of sensors increase to a point where multiple processes try to

access the data causing congestion. After a discussion it was decided that the data handling scrip which at that point only handled data from the ESP32 to the database should handle all data from the ESP32 to mitigate the traffic on the serial port. And even tho the socket solution did not work for our previous intentions here it was a perfect fit. The last decision was which type of socket where to be utilized and it was fairly straightforward that UNIX domain socket was the optimal choice as it is faster than Transmission Control Protocol (TCP) sockets, which we are looking for, and that our system is local so the need for a communication channel over internet is not needed.

Code Development Process (Client Side)
For the client, implemented on the Qt desktop, the socket solution was developed in a C++ environment due to prior experience within the team and as Qt uses C++ no conflict arose from this approach. With that in mind the general setup for a socket implementation on the client side was used (i.e. make socket and connect to it) while the options for how to handle the received data was deliberated more thoroughly as to keep in mind the scalability of the system. The methods considered where fixed length messages, fixed length header and delimiter to detect the length of each serial data. Fixed length was ruled out almost immediately as any additions to the system means that the value would need to be redefined while the other two functions in an expanding system. It was ultimately decided to use the delimiter as it reduced the complexity on the server side by just having to add a "newline" at the end of the data it catches from the ESP32. This naturally also means that the data will arrive in a JSON format which makes the extraction of the relevant data an easy process.
From there after receiving data over the socket and adding it to a string one would only need to find the position of the "newline", add the data before the position to a new string and delete the data + the "newline". (i. e. pos + 1). Once you have the data it can be parsed using a number of JSON libraries; in this case nlohmann/json was used, before adding the relevant data for real-time visualization and the flag to check if the brewing has started.
At first this was implemented all in a single main, but for better readability it was later split into on function that handles the opening of the socket and another that handles the retrieval of data.

GUI Doxygen Documentation

Kaffeknekt

Generated by Doxygen 1.13.2

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 Ui Namespace Reference

Qt namespace for UI classes generated from .ui files.

5.1.1 Detailed Description

Qt namespace for UI classes generated from .ui files.

Qt namespace containing UI classes generated from .ui files.

Chapter 6

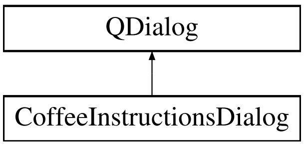
Class Documentation

6.1 CoffeeInstructionsDialog Class Reference

A dialog window that displays coffee brewing instructions.

`#include <coffeeinstructionsdialog.h>`

Inheritance diagram for CoffeeInstructionsDialog:



Public Member Functions

- CoffeeInstructionsDialog (QWidget ∗parent=nullptr)

    Constructor.
- ∼CoffeeInstructionsDialog ()

    Destructor.

Private Attributes

- Ui::CoffeeInstructionsDialog ∗ ui

    Pointer to the UI elements of the dialog.

6.1.1 Detailed Description

A dialog window that displays coffee brewing instructions.

This class represents a modal dialog window built using Qt, specifically for displaying instructions related to coffee preparation.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 CoffeeInstructionsDialog()

```
CoffeeInstructionsDialog::CoffeeInstructionsDialog (
            QWidget * parent = nullptr)  [explicit]
```

Constructor.

Parameters

parent	Pointer to the parent widget (default is nullptr).

Creates and initializes the CoffeeInstructionsDialog.

6.1.2.2 ∼CoffeeInstructionsDialog()

```
CoffeeInstructionsDialog::~CoffeeInstructionsDialog ()
```

Destructor.

Cleans up resources used by the dialog.

6.1.3 Member Data Documentation

6.1.3.1 ui

```
Ui::CoffeeInstructionsDialog* CoffeeInstructionsDialog::ui [private]
```

Pointer to the UI elements of the dialog.

This is generated automatically by Qt Designer (via .ui file).

The documentation for this class was generated from the following files:

- coffeeinstructionsdialog.h
- coffeeinstructionsdialog.cpp

6.2 DataFetcher Class Reference

Utility class for fetching sensor data from InfluxDB.

```
#include <datafetcher.h>
```

Public Member Functions

- **DataFetcher** ()

  Default constructor for DataFetcher.

Static Public Member Functions

- static QVector< DataPoint > fetchPressureWindow (qreal windowSeconds, const QString &influxUrl, const QString &token, const QString &bucket)

  Fetch pressure data points within a given time window.
- static QVector< DataPoint > fetchTempWindow (qreal windowSeconds, const QString &influxUrl, const QString &token, const QString &bucket)

  Fetch temperature data points within a given time window.

6.2.1 Detailed Description

Utility class for fetching sensor data from InfluxDB.

This class provides static methods for retrieving temperature and pressure data over a given time window from an InfluxDB source.

6.2.2 Member Function Documentation

6.2.2.1 fetchPressureWindow()

```
QVector< DataPoint > DataFetcher::fetchPressureWindow (
            qreal windowSeconds,
            const QString & influxUrl,
            const QString & token,
            const QString & bucket)  [static]
```

Fetch pressure data points within a given time window.

Parameters

windowSeconds	The size of the time window in seconds.
influxUrl	URL of the InfluxDB server.
token	Authentication token for accessing the InfluxDB.
bucket	Name of the bucket to query data from.

Returns

QVector<DataPoint> A vector of pressure data points.

6.2.2.2 fetchTempWindow()

```
QVector< DataPoint > DataFetcher::fetchTempWindow (
            qreal windowSeconds,
            const QString & influxUrl,
            const QString & token,
            const QString & bucket)  [static]
```

Fetch temperature data points within a given time window.

Parameters

windowSeconds	The size of the time window in seconds.
influxUrl	URL of the InfluxDB server.
token	Authentication token for accessing the InfluxDB.
bucket	Name of the bucket to query data from.

Returns

QVector<DataPoint> A vector of temperature data points.

The documentation for this class was generated from the following files:

- datafetcher.h
- datafetcher.cpp

6.3 DataPoint Struct Reference

Represents a single data sample with a timestamp and a value.

```
#include <graphview.h>
```

Public Attributes

- qreal **timestamp**

    Time of the data point (seconds since epoch or relative).
- qreal **value**

    Value of the data point (e.g., temperature or pressure).

6.3.1 Detailed Description

Represents a single data sample with a timestamp and a value.

The documentation for this struct was generated from the following file:

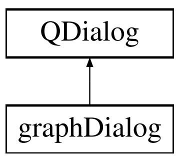
- graphview.h

6.4 graphDialog Class Reference

Dialog window for displaying real-time pressure and temperature graphs.

```
#include <graphdialog.h>
```

Inheritance diagram for graphDialog:



Public Slots

- void appendData (double pressure, double temperature)

    Appends pressure and temperature data to the internal buffer.
- void onDataReceived (double pressure, double temperature, const QString &flag)

    Handles data received from the WebSocketClient.

Signals

- void **dialogShown** ()

    Emitted when the dialog is shown.
- void **dialogHidden** ()

    Emitted when the dialog is hidden.
- void **flagsent** ()

    Emitted when a flag is sent.

Public Member Functions

- graphDialog (QWidget ∗parent=nullptr)

  Constructs a graphDialog window.
- ∼**graphDialog** ()

  Destroys the graphDialog instance.
- void setWindowSeconds (qreal seconds)

  Sets the time window (in seconds) to display on the graph.

Protected Member Functions

- void showEvent (QShowEvent ∗ev) override

  Handles the event when the dialog is shown.
- void hideEvent (QHideEvent ∗ev) override

  Handles the event when the dialog is hidden.

Private Attributes

- Ui::graphDialog ∗ **ui**

  Pointer to the UI components.
- GraphWidget ∗ **m_graph**

  Widget used to plot the data.
- WebSocketClient ∗ **m_wsClient**

  WebSocket client for real-time data.
- QVector< DataPoint > **m_pressure**

  Buffer for pressure data points.
- QVector< DataPoint > **m_temp**

  Buffer for temperature data points.
- qreal **m_windowSeconds** = 600.0

  Display window size in seconds.

6.4.1 Detailed Description

Dialog window for displaying real-time pressure and temperature graphs.

This class handles data visualization using a GraphWidget, and receives real-time updates via WebSocketClient. It also emits signals on show/hide events and when a specific flag is sent.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 graphDialog()

```
graphDialog::graphDialog (
            QWidget * parent = nullptr)  [explicit]
```

Constructs a graphDialog window.

Parameters

parent	The parent widget.

6.4.3 Member Function Documentation

6.4.3.1 appendData

```
void graphDialog::appendData (
            double pressure,
            double temperature)  [slot]
```

Appends pressure and temperature data to the internal buffer.

Parameters

pressure	The new pressure reading.
temperature	The new temperature reading.

6.4.3.2 hideEvent()

```
void graphDialog::hideEvent (
            QHideEvent * ev)  [override], [protected]
```

Handles the event when the dialog is hidden.

Parameters

ev	Pointer to the QHideEvent.

6.4.3.3 onDataReceived

```
void graphDialog::onDataReceived (
            double pressure,
            double temperature,
            const QString & flag)  [slot]
```

Handles data received from the WebSocketClient.

Parameters

pressure	The received pressure value.
temperature	The received temperature value.
flag	A string flag received with the data.

6.4.3.4 setWindowSeconds()

```
void graphDialog::setWindowSeconds (
            qreal seconds)  [inline]
```

Sets the time window (in seconds) to display on the graph.

Parameters

seconds	Length of the time window.

6.4.3.5 showEvent()

```
void graphDialog::showEvent (
            QShowEvent * ev)  [override], [protected]
```

Handles the event when the dialog is shown.

Parameters

ev	Pointer to the QShowEvent.

The documentation for this class was generated from the following files:

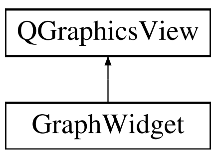
- graphdialog.h
- graphdialog.cpp

6.5 GraphWidget Class Reference

A custom graphics view for plotting pressure and temperature data in real-time.

```
#include <graphview.h>
```

Inheritance diagram for GraphWidget:



Public Member Functions

- GraphWidget (QWidget ∗parent=nullptr)
  Constructs a GraphWidget.
- ∼**GraphWidget** ()
  Destructor.
- void **start** ()
  Starts the periodic data refresh timer.
- void setWindowSeconds (qreal seconds)
  Sets the size of the time window (in seconds) for graph display.
- void appendPressurePoint (const DataPoint &p)
  Appends a new pressure data point.
- void appendTempPoint (const DataPoint &t)

Appends a new temperature data point.
- void **refresh** ()

  Redraws the entire graph: clears scene, draws axes and data series.
- void **clearData** ()

  Clears all stored data points and the graph display.
- void **drawAxes** ()

  Draws the axes on the graph.
- void drawSeries (const QVector< DataPoint > &series, const QColor &penColor, qreal yOffset)

  Draws a time series on the graph with a given color and Y offset.

Protected Member Functions

- void resizeEvent (QResizeEvent ∗event) override

  Handles resizing of the widget to adjust the graph accordingly.

Private Slots

- void **fetchAndRedraw** ()

  Called periodically to fetch new data and update the graph.

Private Attributes

- QGraphicsScene ∗ **m_scene**

  Scene for rendering the graph.
- QTimer ∗ **m_timer**

  Timer for triggering periodic updates.
- QVector< DataPoint > **m_pressure**

  Stored pressure data points.
- QVector< DataPoint > **m_temp**

  Stored temperature data points.
- qreal **m_windowSeconds** = 600.0

  Time window for visible data (default 10 minutes).

6.5.1 Detailed Description

A custom graphics view for plotting pressure and temperature data in real-time.

This widget displays two time-series graphs (pressure and temperature) and periodically updates to reflect new incoming data.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 GraphWidget()

```
GraphWidget::GraphWidget (
            QWidget * parent = nullptr)  [explicit]
```

Constructs a GraphWidget.

Parameters

parent	The parent widget.

6.5.3 Member Function Documentation

6.5.3.1 appendPressurePoint()

```
void GraphWidget::appendPressurePoint (
            const DataPoint & p)
```

Appends a new pressure data point.

Parameters

p	The pressure data point to append.

6.5.3.2 appendTempPoint()

```
void GraphWidget::appendTempPoint (
            const DataPoint & t)
```

Appends a new temperature data point.

Parameters

t	The temperature data point to append.

6.5.3.3 drawSeries()

```
void GraphWidget::drawSeries (
            const QVector< DataPoint > & series,
            const QColor & penColor,
            qreal yOffset)
```

Draws a time series on the graph with a given color and Y offset.

Parameters

series	The data series to draw.
penColor	The color of the graph line.
yOffset	Vertical offset for the series.

6.5.3.4 resizeEvent()

```
void GraphWidget::resizeEvent (
            QResizeEvent * event)  [override], [protected]
```

Handles resizing of the widget to adjust the graph accordingly.

Parameters

event	Resize event.

6.5.3.5 setWindowSeconds()

```
void GraphWidget::setWindowSeconds (
             qreal seconds)  [inline]
```

Sets the size of the time window (in seconds) for graph display.

Parameters

seconds	Length of the display window in seconds.

The documentation for this class was generated from the following files:

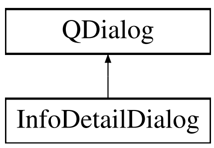
- graphview.h
- graphview.cpp

6.6 InfoDetailDialog Class Reference

A dialog that displays detailed information with navigation options.

```
#include <infodetaildialog.h>
```

Inheritance diagram for InfoDetailDialog:



Public Member Functions

- InfoDetailDialog (QWidget ∗parent=nullptr)

    Constructs the InfoDetailDialog.
- ∼**InfoDetailDialog** ()

    Destructor.

Private Slots

- void **on_btnBack_clicked** ()

    Slot triggered when the "Back" button is clicked.
- void **on_btnHome_clicked** ()

    Slot triggered when the "Home" button is clicked.

Private Attributes

- Ui::InfoDetailDialog ∗ **ui**

  Pointer to the UI components.

6.6.1 Detailed Description

A dialog that displays detailed information with navigation options.

This dialog provides a UI for displaying extended information and includes navigation buttons such as "Back" and "Home".

6.6.2 Constructor & Destructor Documentation

6.6.2.1 InfoDetailDialog()

```
InfoDetailDialog::InfoDetailDialog (
            QWidget * parent = nullptr)  [explicit]
```

Constructs the InfoDetailDialog.

Parameters

parent	The parent widget (optional).

The documentation for this class was generated from the following files:

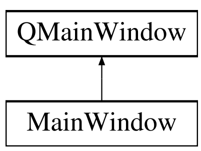
- infodetaildialog.h
- infodetaildialog.cpp

6.7 MainWindow Class Reference

The main application window for the Kaffeknekt dashboard.

```
#include <mainwindow.h>
```

Inheritance diagram for MainWindow:



Public Member Functions

- MainWindow (QWidget ∗parent=nullptr)

  Constructs the main window.
- ∼**MainWindow** ()

  Destructor.
- void **hideInfoFrame** ()

  Hides the informational side frame (if visible).
- void **on_flagsent** ()

  Slot to handle when a flag is sent (e.g., for a warning or state change).

Protected Member Functions

- void showEvent (QShowEvent ∗event) override

  Handles the event when the main window is shown.

Private Slots

- void **on_btnHamburger_clicked** ()

  Slot triggered when the hamburger menu button is clicked.
- void **on_btnHome_clicked** ()

  Slot triggered when the home button is clicked.
- void **on_btnSettings_clicked** ()

  Slot triggered when the settings button is clicked.
- void **on_btnInfo_clicked** ()

  Slot triggered when the info button is clicked.
- void **on_btnInstructions_clicked** ()

  Slot triggered when the instructions button is clicked.
- void **on_btnSensorAnalytics_clicked** ()

  Slot triggered when the sensor analytics button is clicked.
- void **on_btnStatistics_clicked** ()

  Slot triggered when the statistics button is clicked.
- void **on_btnHere_clicked** ()

  Slot triggered when the "Here" button inside info frame is clicked.
- void **on_btnTestWarning_clicked** ()

  Slot triggered when the test warning button is clicked. Used for testing the warning popup.

Private Attributes

- Ui::Kaffeknekt ∗ **ui**

  Pointer to UI components.
- GraphWidget ∗ **m_graph**

  Graph widget for data display.
- Statistics ∗ **m_statsDialog**

  Dialog for showing statistics.
- SensorAnalyticsDialog ∗ **m_saDialog**

  Dialog for real-time analytics.
- WebSocketClient ∗ **m_ws**

  WebSocket client for real-time data.

- bool **sideMenuVisible**

  Tracks visibility of side menu.
- int **m_cupCount** = 0

  Tracks the number of cups (usage metric).
- bool **m_warningShown** = false

  Whether a warning dialog has been shown.
- graphDialog ∗ **m_graphDialog**

  Dialog for graph display.

6.7.1 Detailed Description

The main application window for the Kaffeknekt dashboard.

This class controls the main GUI, manages views such as graphs, settings, and instructions, and connects to real-time data via WebSocket. It also handles various UI interactions.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 MainWindow()

```
MainWindow::MainWindow (
            QWidget * parent = nullptr)
```

Constructs the main window.

Parameters

parent	Optional parent widget.

6.7.3 Member Function Documentation

6.7.3.1 showEvent()

```
void MainWindow::showEvent (
            QShowEvent * event)  [override], [protected]
```

Handles the event when the main window is shown.

Parameters

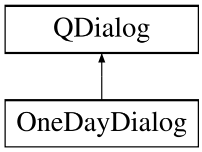
event	Pointer to the QShowEvent.

The documentation for this class was generated from the following files:

- mainwindow.h
- mainwindow.cpp
- warningdialog.cpp

6.8 OneDayDialog Class Reference

Inheritance diagram for OneDayDialog:



Public Member Functions

- **OneDayDialog** (QWidget ∗parent=nullptr)

Private Attributes

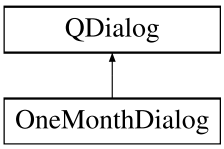
- Ui::OneDayDialog ∗ **ui**

The documentation for this class was generated from the following files:

- onedaydialog.h
- onedaydialog.cpp

6.9 OneMonthDialog Class Reference

Inheritance diagram for OneMonthDialog:



Public Member Functions

- **OneMonthDialog** (QWidget ∗parent=nullptr)

Private Attributes

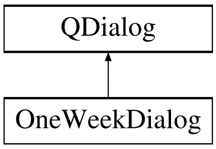
- Ui::OneMonthDialog ∗ **ui**

The documentation for this class was generated from the following files:

- onemonthdialog.h
- onemonthdialog.cpp

6.10 OneWeekDialog Class Reference

Inheritance diagram for OneWeekDialog:



Public Member Functions

- **OneWeekDialog** (QWidget ∗parent=nullptr)

Private Attributes

- Ui::OneWeekDialog ∗ **ui**

The documentation for this class was generated from the following files:

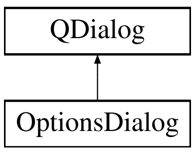
- oneweekdialog.h
- oneweekdialog.cpp

6.11 OptionsDialog Class Reference

Dialog window for displaying and modifying application options.

```
#include <optionsdialog.h>
```

Inheritance diagram for OptionsDialog:



Public Member Functions

- OptionsDialog (QWidget ∗parent=nullptr)

  Constructs the OptionsDialog.
- ∼**OptionsDialog** ()

  Destructor.

Private Attributes

- Ui::OptionsDialog ∗ **ui**

  Pointer to the UI components.

6.11.1 Detailed Description

Dialog window for displaying and modifying application options.

This class provides a simple Qt dialog interface for setting or displaying configuration options within the application.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 OptionsDialog()

```
OptionsDialog::OptionsDialog (
            QWidget * parent = nullptr)  [explicit]
```

Constructs the OptionsDialog.

Parameters

parent	The parent widget (optional).

The documentation for this class was generated from the following files:

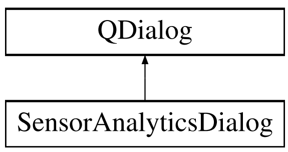
- optionsdialog.h
- optionsdialog.cpp

6.12 SensorAnalyticsDialog Class Reference

Dialog for displaying and selecting sensor analytics metrics.

`#include <sensoranalyticsdialog.h>`

Inheritance diagram for SensorAnalyticsDialog:



Signals

- void intervalSelected (qreal seconds)
  Emitted when the user selects a time interval.

Public Member Functions

- SensorAnalyticsDialog (QWidget *parent=nullptr)
  Constructs the SensorAnalyticsDialog.
- ∼**SensorAnalyticsDialog** ()
  Destructor.

Private Slots

- void **on_btnWaterTemp_clicked** ()
  Slot triggered when the water temperature button is clicked.
- void **on_btnWaterPressure_clicked** ()
  Slot triggered when the water pressure button is clicked.
- void **on_btnPowerConsumption_clicked** ()
  Slot triggered when the power consumption button is clicked.
- void **on_day_clicked** ()
  Slot triggered when the "day" interval is selected.
- void **on_week_clicked** ()
  Slot triggered when the "week" interval is selected.
- void **on_month_clicked** ()
  Slot triggered when the "month" interval is selected.

Private Attributes

- Ui::SensorAnalyticsDialog * **ui**
  Pointer to the UI components.

6.12.1 Detailed Description

Dialog for displaying and selecting sensor analytics metrics.

This dialog allows the user to view different sensor-related data (e.g., water temperature, pressure, power consumption) and to select the time interval for data analysis.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 SensorAnalyticsDialog()

```
SensorAnalyticsDialog::SensorAnalyticsDialog (
            QWidget * parent = nullptr)  [explicit]
```

Constructs the SensorAnalyticsDialog.

Parameters

parent	The parent widget (optional).

6.12.3 Member Function Documentation

6.12.3.1 intervalSelected

```
void SensorAnalyticsDialog::intervalSelected (
            qreal seconds)  [signal]
```

Emitted when the user selects a time interval.

Parameters

seconds	The selected interval in seconds.

The documentation for this class was generated from the following files:

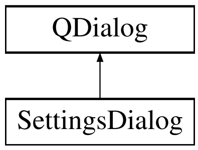
- sensoranalyticsdialog.h
- sensoranalyticsdialog.cpp

6.13 SettingsDialog Class Reference

Dialog window for configuring application settings.

`#include <settingsdialog.h>`

Inheritance diagram for SettingsDialog:



Public Member Functions

- SettingsDialog (QWidget *parent=nullptr)

  Constructs the SettingsDialog.
- ∼**SettingsDialog** ()

  Destructor.

Private Attributes

- Ui::SettingsDialog * **ui**

  Pointer to the UI components.

6.13.1 Detailed Description

Dialog window for configuring application settings.

This class provides a Qt-based dialog interface where users can view and modify application-specific settings.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 SettingsDialog()

```
SettingsDialog::SettingsDialog (
            QWidget * parent = nullptr)  [explicit]
```

Constructs the SettingsDialog.

Parameters

parent	The parent widget (optional).

The documentation for this class was generated from the following files:

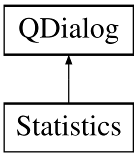
- settingsdialog.h
- settingsdialog.cpp

6.14 Statistics Class Reference

Dialog for displaying usage statistics.

```
#include <statistics.h>
```

Inheritance diagram for Statistics:



Public Member Functions

- Statistics (QWidget ∗parent=nullptr)

  Constructs the Statistics dialog.
- ∼**Statistics** ()

  Destructor.
- void setCupCount (int num)

  Sets the number of cups to display in the statistics.

Private Attributes

- Ui::Statistics ∗ **ui**

  Pointer to the UI components.
- int **m_cupCount** = 0

  Number of cups tracked for statistics.

6.14.1 Detailed Description

Dialog for displaying usage statistics.

This class provides a Qt dialog that displays statistical information, such as the number of cups made by the espresso machine.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Statistics()

```
Statistics::Statistics (
                QWidget * parent = nullptr)  [explicit]
```

Constructs the Statistics dialog.

Parameters

parent	The parent widget (optional).

6.14.3 Member Function Documentation

6.14.3.1 setCupCount()

```
void Statistics::setCupCount (
             int num)
```

Sets the number of cups to display in the statistics.

Parameters

num	The number of cups.

The documentation for this class was generated from the following files:

- statistics.h
- statistics.cpp

6.15 WarningDialog Class Reference

Dialog for displaying a temperature-related warning to the user.

```
#include <warningdialog.h>
```

6.15.1 Detailed Description

Dialog for displaying a temperature-related warning to the user.

This dialog notifies the user when a critical temperature threshold is reached. It provides options to ignore or acknowledge the warning.

The documentation for this class was generated from the following file:

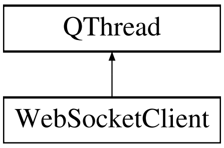
- warningdialog.h

6.16 WebSocketClient Class Reference

Threaded client for receiving real-time data via WebSocket.

`#include <websocketclient.h>`

Inheritance diagram for WebSocketClient:



Signals

- void dataReceived (double pressure, double temperature, const QString &flag)

  Emitted when new sensor data is received.

Public Member Functions

- WebSocketClient (QObject *parent=nullptr)

  Constructs the WebSocketClient.
- void **run** () override

  Runs the thread logic. Override this method to implement the WebSocket communication.

6.16.1 Detailed Description

Threaded client for receiving real-time data via WebSocket.

This class runs in a separate thread and is responsible for connecting to a WebSocket server to receive live pressure, temperature, and flag data.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 WebSocketClient()

```
WebSocketClient::WebSocketClient (
            QObject * parent = nullptr)  [explicit]
```

Constructs the WebSocketClient.

Parameters

parent	Optional parent QObject.

6.16.3 Member Function Documentation

6.16.3.1 dataReceived

```
void WebSocketClient::dataReceived (
            double pressure,
            double temperature,
            const QString & flag)  [signal]
```

Emitted when new sensor data is received.

Parameters

pressure	The pressure value.
temperature	The temperature value.
flag	A string-based flag or status indicator.

The documentation for this class was generated from the following files:

- websocketclient.h
- websocketclient.cpp

Chapter 7

File Documentation

7.1 coffeeinstructionsdialog.h

```
#ifndef COFFEEINSTRUCTIONSDIALOG_H
#define COFFEEINSTRUCTIONSDIALOG_H

#include <QDialog>

namespace Ui {
class CoffeeInstructionsDialog;
}

class CoffeeInstructionsDialog : public QDialog
{
    Q_OBJECT

public:
    explicit CoffeeInstructionsDialog(QWidget *parent = nullptr);

    ~CoffeeInstructionsDialog();

private:
    Ui::CoffeeInstructionsDialog *ui;
};

#endif // COFFEEINSTRUCTIONSDIALOG_H
```

7.2 datafetcher.h

```
#ifndef DATAFETCHER_H
#define DATAFETCHER_H
#include <QDebug>
#include <QObject>
#include <QVector>
#include "graphview.h"    // for DataPoint

class DataFetcher
{
public:
    DataFetcher();

    static QVector<DataPoint> fetchPressureWindow(qreal windowSeconds,
                                                  const QString& influxUrl,
                                                  const QString& token,
                                                  const QString& bucket);

    static QVector<DataPoint> fetchTempWindow(qreal windowSeconds,
                                              const QString& influxUrl,
                                              const QString& token,
                                              const QString& bucket);
};

#endif // DATAFETCHER_H
```

7.3 graphdialog.h

```
//graphdialog.h
#ifndef GRAPHDIALOG_H
#define GRAPHDIALOG_H

#include <QDialog>
#include <QSizePolicy>

#include "graphview.h"
#include "websocketclient.h"

namespace Ui { class graphDialog; }

class graphDialog : public QDialog {
    Q_OBJECT

public:
    explicit graphDialog(QWidget *parent = nullptr);

    ~graphDialog();

    void setWindowSeconds(qreal seconds) { m_windowSeconds = seconds; }

signals:
    void dialogShown();

    void dialogHidden();

    void flagsent();

public slots:
    void appendData(double pressure, double temperature);

    void onDataReceived(double pressure,
                        double temperature,
                        const QString& flag);

private:
    Ui::graphDialog *ui;
    GraphWidget     *m_graph;
    WebSocketClient *m_wsClient;

    QVector<DataPoint> m_pressure;
    QVector<DataPoint> m_temp;
    qreal              m_windowSeconds = 600.0;

protected:
    void showEvent(QShowEvent* ev) override;

    void hideEvent(QHideEvent* ev) override;
};

#endif // GRAPHDIALOG_H
```

7.4 graphview.h

```
#ifndef GRAPHWIDGET_H
#define GRAPHWIDGET_H

#include <QGraphicsView>
#include <QGraphicsScene>
#include <QTimer>

#include <QResizeEvent>
#include <QGraphicsSimpleTextItem>

struct DataPoint {
    qreal timestamp;
    qreal value;
};

class GraphWidget : public QGraphicsView {
    Q_OBJECT
public:
    explicit GraphWidget(QWidget* parent = nullptr);

    ~GraphWidget();

    void start();

    void setWindowSeconds(qreal seconds) { m_windowSeconds = seconds; }
```

```
    void appendPressurePoint(const DataPoint& p);

    void appendTempPoint(const DataPoint& t);

    void refresh();

    void clearData();

    void drawAxes();

    void drawSeries(const QVector<DataPoint>& series,
                    const QColor& penColor,
                    qreal yOffset);

private slots:
    void fetchAndRedraw();

protected:
    void resizeEvent(QResizeEvent* event) override;

private:
    QGraphicsScene* m_scene;
    QTimer*         m_timer;

    QVector<DataPoint> m_pressure;
    QVector<DataPoint> m_temp;

    qreal m_windowSeconds = 600.0;
};

#endif // GRAPHWIDGET_H
```

7.5 infodetaildialog.h

```
#ifndef INFODETAILDIALOG_H
#define INFODETAILDIALOG_H

#include <QDialog>

namespace Ui {
class InfoDetailDialog;
}

class InfoDetailDialog : public QDialog
{
    Q_OBJECT

public:
    explicit InfoDetailDialog(QWidget *parent = nullptr);

    ~InfoDetailDialog();

private slots:
    void on_btnBack_clicked();

    void on_btnHome_clicked();

private:
    Ui::InfoDetailDialog *ui;
};

#endif // INFODETAILDIALOG_H
```

7.6 mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "sensoranalyticsdialog.h"
#include "coffeeinstructionsdialog.h"
#include "graphview.h"
#include "warningdialog.h"
#include "websocketclient.h"
#include "graphdialog.h"
#include <QMainWindow>
#include <QTimer>
#include <QGraphicsScene>
```

```
#include <QGraphicsView>
#include "statistics.h"
#include "ui_mainwindow.h"
#include "infodetaildialog.h"
#include "datafetcher.h"
#include <QFrame>
#include <QPushButton>

QT_BEGIN_NAMESPACE
namespace Ui {
class Kaffeknekt;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);

    ~MainWindow();

    void hideInfoFrame();

    void on_flagsent();

private slots:
    // Frame 1 (main menu)
    void on_btnHamburger_clicked();

    void on_btnHome_clicked();

    void on_btnSettings_clicked();

    void on_btnInfo_clicked();

    // Frame 2 (submenu)
    void on_btnInstructions_clicked();

    void on_btnSensorAnalytics_clicked();

    void on_btnStatistics_clicked();

    void on_btnHere_clicked();

    void on_btnTestWarning_clicked();

protected:
    void showEvent(QShowEvent *event) override;

private:
    Ui::Kaffeknekt *ui;
    GraphWidget    *m_graph;
    Statistics          *m_statsDialog;
    SensorAnalyticsDialog *m_saDialog;
    WebSocketClient       *m_ws;
    bool sideMenuVisible;
    int    m_cupCount         = 0;
    bool   m_warningShown      = false;
    graphDialog     *m_graphDialog;
};

#endif // MAINWINDOW_H
```

7.7 moc_predefs.h

```
#define __DBL_MIN_EXP__ (-1021)
#define __LDBL_MANT_DIG__ 113
#define __cpp_attributes 200809L
#define __cpp_nontype_template_parameter_auto 201606L
#define __UINT_LEAST16_MAX__ 0xffff
#define __ARM_SIZEOF_WCHAR_T 4
#define __ATOMIC_ACQUIRE 2
#define __FLT128_MAX_10_EXP__ 4932
#define __FLT_MIN__ 1.17549435082228750796873653722224568e-38F
#define __GCC_IEC_559_COMPLEX 2
#define __cpp_aggregate_nsdmi 201304L
#define __UINT_LEAST8_TYPE__ unsigned char
#define __INTMAX_C(c) c ## L
#define __CHAR_BIT__ 8
#define __UINT8_MAX__ 0xff
#define __USER_LABEL_PREFIX__
```

```
#define __WINT_MAX__ 0xffffffffU
#define __cpp_static_assert 201411L
#define __WCHAR_MAX__ 0xffffffffU
#define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_2 1
#define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_4 1
#define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_8 1
#define __GCC_ATOMIC_CHAR_LOCK_FREE 2
#define __GCC_IEC_559 2
#define __FLT32X_DECIMAL_DIG__ 17
#define __FLT_EVAL_METHOD__ 0
#define __cpp_binary_literals 201304L
#define __FLT64_DECIMAL_DIG__ 17
#define __cpp_noexcept_function_type 201510L
#define __GCC_ATOMIC_CHAR32_T_LOCK_FREE 2
#define __cpp_variadic_templates 200704L
#define __UINT_FAST32_TYPE__ long unsigned int
#define __UINT_FAST64_MAX__ 0xffffffffffffffffUL
#define __SIG_ATOMIC_TYPE__ int
#define __DBL_MIN_10_EXP__ (-307)
#define __FINITE_MATH_ONLY__ 0
#define __cpp_variable_templates 201304L
#define __FLT32X_MAX_EXP__ 1024
#define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_1 1
#define __GNUC_PATCHLEVEL__ 0
#define __FLT32_HAS_DENORM__ 1
#define __UINT_FAST8_MAX__ 0xff
#define __cpp_rvalue_reference 200610L
#define __cpp_nested_namespace_definitions 201411L
#define __INT8_C(c) c
#define __INT_LEAST8_WIDTH__ 8
#define __cpp_variadic_using 201611L
#define __UINT_LEAST64_MAX__ 0xffffffffffffffffUL
#define __INT_LEAST8_MAX__ 0x7f
#define __cpp_capture_star_this 201603L
#define __SHRT_MAX__ 0x7fff
#define __STDC_ISO_10646__ 201706L
#define __LDBL_MAX__ 1.18973149535723176508575932662800702e+4932L
#define __ARM_FEATURE_IDIV 1
#define __FLT64X_MAX_10_EXP__ 4932
#define __cpp_if_constexpr 201606L
#define __FLT64_NORM_MAX__ 1.79769313486231570814527423731704357e+308F64
#define __LDBL_IS_IEC_60559__ 2
#define __ARM_FP 14
#define __FLT64X_IS_IEC_60559__ 2
#define __FLT64X_HAS_QUIET_NAN__ 1
#define __WINT_TYPE__ unsigned int
#define __UINT_LEAST8_MAX__ 0xff
#define __FLT128_DENORM_MIN__ 6.47517511943802511092443895822764655e-4966F128
#define __UINTMAX_TYPE__ long unsigned int
#define __cpp_nsdmi 200809L
#define __linux 1
#define __CHAR_UNSIGNED__ 1
#define __UINT32_MAX__ 0xffffffffU
#define __GXX_EXPERIMENTAL_CXX0X__ 1
#define __DBL_DENORM_MIN__ double(4.94065645841246544176568792868221372e-324L)
#define __AARCH64_CMODEL_SMALL__ 1
#define __LDBL_MAX_EXP__ 16384
#define __INT_FAST32_WIDTH__ 64
#define __FLT128_MIN_EXP__ (-16381)
#define __FLT128_MIN_10_EXP__ (-4931)
#define __FLT32X_IS_IEC_60559__ 2
#define __INT_LEAST16_WIDTH__ 16
#define __SCHAR_MAX__ 0x7f
#define __FLT128_MANT_DIG__ 113
#define __DBL_MAX__ double(1.79769313486231570814527423731704357e+308L)
#define __FLT32X_DIG__ 15
#define __WCHAR_MIN__ 0U
#define __INT64_C(c) c ## L
#define __GCC_ATOMIC_POINTER_LOCK_FREE 2
#define __FLT_MAX__ 3.40282346638528859811704183484516925e+38F
#define __SIZEOF_INT__ 4
#define __FLT32X_MANT_DIG__ 53
#define __GCC_ATOMIC_CHAR16_T_LOCK_FREE 2
#define __cpp_aligned_new 201606L
#define __FLT32_MAX_10_EXP__ 38
#define __FLT64X_EPSILON__ 1.92592994438723585305597794258492732e-34F64x
#define __STDC_HOSTED__ 1
#define __cpp_decltype_auto 201304L
#define __DBL_DIG__ 15
#define __FLT32_DIG__ 6
#define __FLT_EPSILON__ 1.19209289550781250000000000000000000e-7F
#define __GXX_WEAK__ 1
#define __SHRT_WIDTH__ 16
#define __FLT32_IS_IEC_60559__ 2
#define __LDBL_MIN__ 3.36210314311209350626267781732175260e-4932L
#define __DBL_IS_IEC_60559__ 2
#define __FLT16_HAS_QUIET_NAN__ 1
```

```
#define __cpp_threadsafe_static_init 200806L
#define __ARM_SIZEOF_MINIMAL_ENUM 4
#define __cpp_enumerator_attributes 201411L
#define __FLT64X_DENORM_MIN__ 6.47517511943802511092443895822764655e-4966F64x
#define __FP_FAST_FMA 1
#define __FLT32X_HAS_INFINITY__ 1
#define __INT32_MAX__ 0x7fffffff
#define __FLT16_DIG__ 3
#define __INT_WIDTH__ 32
#define __SIZEOF_LONG__ 8
#define __STDC_IEC_559__ 1
#define __UINT16_C(c) c
#define __DECIMAL_DIG__ 36
#define __FLT64_EPSILON__ 2.22044604925031308084726333618164062e-16F64
#define __gnu_linux__ 1
#define __INT16_MAX__ 0x7fff
#define __FLT64_MIN_EXP__ (-1021)
#define __FLT64X_MIN_10_EXP__ (-4931)
#define __LDBL_HAS_QUIET_NAN__ 1
#define __FLT16_MIN_EXP__ (-13)
#define __FLT64_MANT_DIG__ 53
#define __FLT64X_MANT_DIG__ 113
#define __GNUC__ 12
#define __pie__ 2
#define __GXX_RTTI 1
#define __FLT16_DECIMAL_DIG__ 5
#define __FLT_HAS_DENORM__ 1
#define __SIZEOF_LONG_DOUBLE__ 16
#define __cpp_rtti 199711L
#define __STDC_UTF_16__ 1
#define __FLT64_MAX_10_EXP__ 308
#define __FLT16_MAX_10_EXP__ 4
#define __FLT32_HAS_INFINITY__ 1
#define __cpp_raw_strings 200710L
#define __INT_FAST32_MAX__ 0x7fffffffffffffffL
#define __DBL_HAS_INFINITY__ 1
#define __INT64_MAX__ 0x7fffffffffffffffL
#define __HAVE_SPECULATION_SAFE_VALUE 1
#define __cpp_fold_expressions 201603L
#define __INTPTR_WIDTH__ 64
#define __FLT64X_HAS_INFINITY__ 1
#define __cpp_delegating_constructors 200604L
#define __FLT32X_HAS_DENORM__ 1
#define __INT_FAST16_TYPE__ long int
#define __cpp_template_auto 201606L
#define __LDBL_HAS_DENORM__ 1
#define __cplusplus 201703L
#define __cpp_ref_qualifiers 200710L
#define __DEPRECATED 1
#define __cpp_rvalue_references 200610L
#define __DBL_MAX_EXP__ 1024
#define __WCHAR_WIDTH__ 32
#define __FLT64_MAX__ 1.79769313486231570814527423731704357e+308F64
#define __FLT32_MAX__ 3.40282346638528859811704183484516925e+38F32
#define __GCC_ATOMIC_LONG_LOCK_FREE 2
#define __FLT16_MANT_DIG__ 11
#define __FLT32_HAS_QUIET_NAN__ 1
#define __LONG_LONG_MAX__ 0x7fffffffffffffffLL
#define __SIZEOF_SIZE_T__ 8
#define __ARM_ALIGN_MAX_PWR 28
#define __SIZEOF_WINT_T__ 4
#define __LONG_LONG_WIDTH__ 64
#define __cpp_initializer_lists 200806L
#define __WCHAR_UNSIGNED__ 1
#define __FLT32_MAX_EXP__ 128
#define __cpp_hex_float 201603L
#define __ARM_FP16_FORMAT_IEEE 1
#define __FP_FAST_FMAF32x 1
#define __FLT128_HAS_INFINITY__ 1
#define __FLT_MIN_EXP__ (-125)
#define __PIE__ 2
#define __GCC_HAVE_DWARF2_CFI_ASM 1
#define __cpp_lambdas 200907L
#define __FLT32X_MIN_EXP__ (-1021)
#define __INT_FAST64_TYPE__ long int
#define __ARM_FP16_ARGS 1
#define __DBL_DECIMAL_DIG__ 17
#define __FP_FAST_FMAF 1
#define __FLT128_NORM_MAX__ 1.18973149535723176508575932662800702e+4932F128
#define __FLT64_DENORM_MIN__ 4.94065645841246544176568792868221372e-324F64
#define __DBL_MIN__ double(2.22507385850720138309023271733240406e-308L)
#define __ARM_FEATURE_CLZ 1
#define __FLT16_DENORM_MIN__ 5.96046447753906250000000000000000000e-8F16
#define __unix__ 1
#define __FLT64X_NORM_MAX__ 1.18973149535723176508575932662800702e+4932F64x
#define __SIZEOF_POINTER__ 8
#define __LP64__ 1
```

```
#define __DBL_HAS_QUIET_NAN__ 1
#define __FLT_EVAL_METHOD_C99__ 0
#define __FLT32X_EPSILON__ 2.22044604925031308084726333618164062e-16F32x
#define __LDBL_DECIMAL_DIG__ 36
#define __aarch64__ 1
#define __FLT64_MIN_10_EXP__ (-307)
#define __INT_FAST64_WIDTH__ 64
#define __FLT64X_DECIMAL_DIG__ 36
#define __REGISTER_PREFIX__
#define __UINT16_MAX__ 0xffff
#define __INTMAX_WIDTH__ 64
#define __GXX_ABI_VERSION 1017
#define __AARCH64EL__ 1
#define __LDBL_HAS_INFINITY__ 1
#define __UINT8_TYPE__ unsigned char
#define __FLT_DIG__ 6
#define __NO_INLINE__ 1
#define __DEC_EVAL_METHOD__ 2
#define __FLT_MANT_DIG__ 24
#define __FLT16_MIN_10_EXP__ (-4)
#define __VERSION__ "12.2.0"
#define __UINT64_C(c) c ## UL
#define __cpp_unicode_characters 201411L
#define _STDC_PREDEF_H 1
#define __INT_LEAST32_MAX__ 0x7fffffff
#define __GCC_ATOMIC_INT_LOCK_FREE 2
#define __FLT128_MAX_EXP__ 16384
#define __FLT32_MANT_DIG__ 24
#define __FLOAT_WORD_ORDER__ __ORDER_LITTLE_ENDIAN__
#define __FLT16_MAX_EXP__ 16
#define __BIGGEST_ALIGNMENT__ 16
#define __STDC_IEC_60559_COMPLEX__ 201404L
#define __INT32_C(c) c
#define __cpp_aggregate_bases 201603L
#define __FLT128_HAS_DENORM__ 1
#define __FLT128_DIG__ 33
#define __SCHAR_WIDTH__ 8
#define __ORDER_PDP_ENDIAN__ 3412
#define __ARM_64BIT_STATE 1
#define __INT_FAST32_TYPE__ long int
#define __FLT128_MIN__ 3.36210314311209350626267781732175260e-4932F128
#define __UINT_LEAST16_TYPE__ short unsigned int
#define __SIZE_TYPE__ long unsigned int
#define __UINT64_MAX__ 0xffffffffffffffffUL
#define __FLT_IS_IEC_60559__ 2
#define __GNUC_WIDE_EXECUTION_CHARSET_NAME "UTF-32LE"
#define __FLT64X_DIG__ 33
#define __ARM_FEATURE_FMA 1
#define __INT8_TYPE__ signed char
#define __GNUG__ 12
#define __cpp_digit_separators 201309L
#define __ELF__ 1
#define __GCC_ASM_FLAG_OUTPUTS__ 1
#define __GCC_ATOMIC_TEST_AND_SET_TRUEVAL 1
#define __FLT_RADIX__ 2
#define __INT_LEAST16_TYPE__ short int
#define __ARM_ARCH_PROFILE 65
#define __LDBL_EPSILON__ 1.92592994438723585305597794258492732e-34L
#define __UINTMAX_C(c) c ## UL
#define __GLIBCXX_BITSIZE_INT_N_0 128
#define __ARM_PCS_AAPCS64 1
#define __SIG_ATOMIC_MAX__ 0x7fffffff
#define __INT_LEAST64_WIDTH__ 64
#define __GCC_ATOMIC_WCHAR_T_LOCK_FREE 2
#define __STDC_IEC_60559_BFP__ 201404L
#define __SIZEOF_PTRDIFF_T__ 8
#define __ATOMIC_RELAXED 0
#define __FLT_EVAL_METHOD_TS_18661_3__ 0
#define unix 1
#define __cpp_guaranteed_copy_elision 201606L
#define __LDBL_DIG__ 33
#define __FLT64_IS_IEC_60559__ 2
#define __FLT16_IS_IEC_60559__ 2
#define __INT_FAST16_MAX__ 0x7fffffffffffffffL
#define __GCC_CONSTRUCTIVE_SIZE 64
#define __FLT64_DIG__ 15
#define __UINT_FAST32_MAX__ 0xffffffffffffffffUL
#define __UINT_LEAST64_TYPE__ long unsigned int
#define __FLT16_EPSILON__ 9.76562500000000000000000000000000000e-4F16
#define __FLT_HAS_QUIET_NAN__ 1
#define __FLT_MAX_10_EXP__ 38
#define __LONG_MAX__ 0x7fffffffffffffffL
#define __FLT64X_HAS_DENORM__ 1
#define __FLT_HAS_INFINITY__ 1
#define __GNUC_EXECUTION_CHARSET_NAME "UTF-8"
#define __unix 1
#define __cpp_unicode_literals 200710L
```

```
#define __DBL_HAS_DENORM__ 1
#define __UINT_FAST16_TYPE__ long unsigned int
#define __FLT32X_HAS_QUIET_NAN__ 1
#define __CHAR16_TYPE__ short unsigned int
#define __PRAGMA_REDEFINE_EXTNAME 1
#define __SIZE_WIDTH__ 64
#define __INT_LEAST16_MAX__ 0x7fff
#define __FLT16_NORM_MAX__ 6.55040000000000000000000000000000000e+4F16
#define __FLT32_DENORM_MIN__ 1.40129846432481707092372958328991613e-45F32
#define __INT_LEAST64_TYPE__ long int
#define __INT16_TYPE__ short int
#define __INT_LEAST8_TYPE__ signed char
#define __FLT16_MAX__ 6.55040000000000000000000000000000000e+4F16
#define __cpp_structured_bindings 201606L
#define __INT_FAST8_MAX__ 0x7f
#define __ARM_ARCH 8
#define __FLT128_MAX__ 1.18973149535723176508575932662800702e+4932F128
#define __INTPTR_MAX__ 0x7fffffffffffffffL
#define __cpp_sized_deallocation 201309L
#define linux 1
#define __FLT64_HAS_QUIET_NAN__ 1
#define __INTMAX_TYPE__ long int
#define __FLT64X_MIN_EXP__ (-16381)
#define __FLT32_MIN_10_EXP__ (-37)
#define __PTRDIFF_WIDTH__ 64
#define __FLT64_HAS_INFINITY__ 1
#define __FLT64X_MAX__ 1.18973149535723176508575932662800702e+4932F64x
#define __FLT16_HAS_INFINITY__ 1
#define __STDCPP_DEFAULT_NEW_ALIGNMENT__ 16
#define __SIG_ATOMIC_MIN__ (-__SIG_ATOMIC_MAX__ - 1)
#define __cpp_nontype_template_args 201411L
#define __PTRDIFF_MAX__ 0x7fffffffffffffffL
#define __cpp_return_type_deduction 201304L
#define __INTPTR_TYPE__ long int
#define __UINT16_TYPE__ short unsigned int
#define __WCHAR_TYPE__ unsigned int
#define __cpp_range_based_for 201603L
#define __pic__ 2
#define __UINTPTR_MAX__ 0xffffffffffffffffUL
#define __ARM_ARCH_8A 1
#define __ARM_FEATURE_UNALIGNED 1
#define __cpp_decltype 200707L
#define __FLT32_DECIMAL_DIG__ 9
#define __INT_FAST64_MAX__ 0x7fffffffffffffffL
#define __FLT_NORM_MAX__ 3.40282346638528859811704183484516925e+38F
#define __FLT64X_MAX_EXP__ 16384
#define __UINT_FAST64_TYPE__ long unsigned int
#define __cpp_inline_variables 201606L
#define __INT_MAX__ 0x7fffffff
#define __STDCPP_THREADS__ 1
#define __INT64_TYPE__ long int
#define __FLT_MAX_EXP__ 128
#define __ORDER_BIG_ENDIAN__ 4321
#define __DBL_MANT_DIG__ 53
#define __cpp_inheriting_constructors 201511L
#define __INT_LEAST64_MAX__ 0x7fffffffffffffffL
#define __FP_FAST_FMAF32 1
#define __UINT_LEAST32_TYPE__ unsigned int
#define __SIZEOF_SHORT__ 2
#define __FLT32_NORM_MAX__ 3.40282346638528859811704183484516925e+38F32
#define __LDBL_MIN_EXP__ (-16381)
#define __GCC_ATOMIC_BOOL_LOCK_FREE 2
#define __SIG_ATOMIC_WIDTH__ 32
#define __WINT_WIDTH__ 32
#define __FP_FAST_FMAF64 1
#define __FLT32X_MAX_10_EXP__ 308
#define __SIZEOF_INT128__ 16
#define __FLT16_MIN__ 6.10351562500000000000000000000000000e-5F16
#define __LDBL_MAX_10_EXP__ 4932
#define __DBL_EPSILON__ double(2.22044604925031308084726333618164062e-16L)
#define __FLT32_MIN_EXP__ (-125)
#define _LP64 1
#define __UINT8_C(c) c
#define __FLT64_MAX_EXP__ 1024
#define __INT_LEAST32_TYPE__ int
#define __SIZEOF_WCHAR_T__ 4
#define __ARM_NEON 1
#define __FLT128_HAS_QUIET_NAN__ 1
#define __INTMAX_MAX__ 0x7fffffffffffffffL
#define __UINT_FAST8_TYPE__ unsigned char
#define __INT_FAST8_TYPE__ signed char
#define __cpp_namespace_attributes 201411L
#define __FLT64X_MIN__ 3.36210314311209350626267781732175260e-4932F64x
#define __GNUC_STDC_INLINE__ 1
#define __FLT64_HAS_DENORM__ 1
#define __FLT32_EPSILON__ 1.19209289550781250000000000000000000e-7F32
#define __FLT16_HAS_DENORM__ 1
```

```
#define __STDC_UTF_32__ 1
#define __INT_FAST8_WIDTH__ 8
#define __FLT32X_MAX__ 1.79769313486231570814527423731704357e+308F32x
#define __cpp_alias_templates 200704L
#define __DBL_NORM_MAX__ double(1.79769313486231570814527423731704357e+308L)
#define __BYTE_ORDER__ __ORDER_LITTLE_ENDIAN__
#define __ARM_ALIGN_MAX_STACK_PWR 16
#define __LDBL_DENORM_MIN__ 6.47517511943802511092443895822764655e-4966L
#define __GCC_DESTRUCTIVE_SIZE 256
#define __cpp_runtime_arrays 198712L
#define __UINT64_TYPE__ long unsigned int
#define __UINT32_C(c) c ## U
#define __FLT32X_MIN__ 2.22507385850720138309023271733240406e-308F32x
#define __WINT_MIN__ 0U
#define __FLT128_IS_IEC_60559__ 2
#define __INT8_MAX__ 0x7f
#define __LONG_WIDTH__ 64
#define __PIC__ 2
#define __FLT32X_NORM_MAX__ 1.79769313486231570814527423731704357e+308F32x
#define __CHAR32_TYPE__ unsigned int
#define __cpp_constexpr 201603L
#define __cpp_deduction_guides 201703L
#define __ARM_FEATURE_NUMERIC_MAXMIN 1
#define __INT32_TYPE__ int
#define __SIZEOF_DOUBLE__ 8
#define __cpp_exceptions 199711L
#define __FLT64_MIN__ 2.22507385850720138309023271733240406e-308F64
#define __FLT_DENORM_MIN__ 1.40129846432481707092372958328991613e-45F
#define __INT_LEAST32_WIDTH__ 32
#define __SIZEOF_FLOAT__ 4
#define __ATOMIC_CONSUME 1
#define __GNUC_MINOR__ 2
#define __GLIBCXX_TYPE_INT_N_0 __int128
#define __INT_FAST16_WIDTH__ 64
#define __UINTMAX_MAX__ 0xffffffffffffffffUL
#define __FLT32X_DENORM_MIN__ 4.94065645841246544176568792868221372e-324F32x
#define __cpp_template_template_args 201611L
#define __DBL_MAX_10_EXP__ 308
#define __INT16_C(c) c
#define __ARM_ARCH_ISA_A64 1
#define __STDC__ 1
#define __PTRDIFF_TYPE__ long int
#define __FLT32_MIN__ 1.17549435082228750796873653722224568e-38F32
#define __ATOMIC_SEQ_CST 5
#define __EXCEPTIONS 1
#define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_16 1
#define __UINT32_TYPE__ unsigned int
#define __FLT32X_MIN_10_EXP__ (-307)
#define __UINTPTR_TYPE__ long unsigned int
#define __linux__ 1
#define __LDBL_MIN_10_EXP__ (-4931)
#define __cpp_generic_lambdas 201304L
#define __FLT128_EPSILON__ 1.92592994438723585305597794258492732e-34F128
#define __SIZEOF_LONG_LONG__ 8
#define __cpp_user_defined_literals 200809L
#define __FLT128_DECIMAL_DIG__ 36
#define __GCC_ATOMIC_LLONG_LOCK_FREE 2
#define __FLT_DECIMAL_DIG__ 9
#define __UINT_FAST16_MAX__ 0xffffffffffffffffUL
#define __STDC_IEC_559_COMPLEX__ 1
#define __LDBL_NORM_MAX__ 1.18973149535723176508575932662800702e+4932L
#define __FLT_MIN_10_EXP__ (-37)
#define __GCC_ATOMIC_SHORT_LOCK_FREE 2
#define __ORDER_LITTLE_ENDIAN__ 1234
#define __SIZE_MAX__ 0xffffffffffffffffUL
#define _GNU_SOURCE 1
#define __UINT_LEAST32_MAX__ 0xffffffffU
#define __cpp_init_captures 201304L
#define __ATOMIC_ACQ_REL 4
#define __ATOMIC_RELEASE 3
```

7.8 onedaydialog.h

```
#ifndef ONEDAYDIALOG_H
#define ONEDAYDIALOG_H

#include <QDialog>

namespace Ui {
class OneDayDialog;
}

class OneDayDialog : public QDialog
```

```
{
    Q_OBJECT

public:
    explicit OneDayDialog(QWidget *parent = nullptr);
    ~OneDayDialog();

private:
    Ui::OneDayDialog *ui;
};

#endif // ONEDAYDIALOG_H
```

7.9 onemonthdialog.h

```
#ifndef ONEMONTHDIALOG_H
#define ONEMONTHDIALOG_H

#include <QDialog>

namespace Ui {
class OneMonthDialog;
}

class OneMonthDialog : public QDialog
{
    Q_OBJECT

public:
    explicit OneMonthDialog(QWidget *parent = nullptr);
    ~OneMonthDialog();

private:
    Ui::OneMonthDialog *ui;
};

#endif // ONEMONTHDIALOG_H
```

7.10 oneweekdialog.h

```
#ifndef ONEWEEKDIALOG_H
#define ONEWEEKDIALOG_H

#include <QDialog>

namespace Ui {
class OneWeekDialog;
}

class OneWeekDialog : public QDialog
{
    Q_OBJECT

public:
    explicit OneWeekDialog(QWidget *parent = nullptr);
    ~OneWeekDialog();

private:
    Ui::OneWeekDialog *ui;
};

#endif // ONEWEEKDIALOG_H
```

7.11 optionsdialog.h

```
#ifndef OPTIONSDIALOG_H
#define OPTIONSDIALOG_H

#include <QDialog>

namespace Ui {
class OptionsDialog;
}
```

```
class OptionsDialog : public QDialog
{
    Q_OBJECT

public:
    explicit OptionsDialog(QWidget *parent = nullptr);

    ~OptionsDialog();

private:
    Ui::OptionsDialog *ui;
};

#endif // OPTIONSDIALOG_H
```

7.12 sensoranalyticsdialog.h

```
#ifndef SENSORANALYTICSDIALOG_H
#define SENSORANALYTICSDIALOG_H

#include <QDialog>
#include <QDebug>

namespace Ui {
class SensorAnalyticsDialog;
}

class SensorAnalyticsDialog : public QDialog
{
    Q_OBJECT

public:
    explicit SensorAnalyticsDialog(QWidget *parent = nullptr);

    ~SensorAnalyticsDialog();

private slots:
    void on_btnWaterTemp_clicked();

    void on_btnWaterPressure_clicked();

    void on_btnPowerConsumption_clicked();

    void on_day_clicked();

    void on_week_clicked();

    void on_month_clicked();

signals:
    void intervalSelected(qreal seconds);

private:
    Ui::SensorAnalyticsDialog *ui;
};

#endif // SENSORANALYTICSDIALOG_H
```

7.13 settingsdialog.h

```
#ifndef SETTINGSDIALOG_H
#define SETTINGSDIALOG_H

#include <QDialog>

namespace Ui {
class SettingsDialog;
}

class SettingsDialog : public QDialog
{
    Q_OBJECT

public:
    explicit SettingsDialog(QWidget *parent = nullptr);

    ~SettingsDialog();
```

```
private:
    Ui::SettingsDialog *ui;
};
#endif // SETTINGSDIALOG_H
```

7.14 statistics.h

```
#ifndef STATISTICS_H
#define STATISTICS_H
#include <QDialog>
namespace Ui {
class Statistics;
}
class Statistics : public QDialog
{
    Q_OBJECT
public:
    explicit Statistics(QWidget *parent = nullptr);
    ~Statistics();
    void setCupCount(int num);
private:
    Ui::Statistics *ui;
    int             m_cupCount = 0;
};
#endif // STATISTICS_H
```

7.15 warningdialog.h

```
#ifndef WARNINGDIALOG_H
#define WARNINGDIALOG_H
#include <QDialog>
namespace Ui { class WarningDialog; }
class WarningDialog : public QDialog {
    Q_OBJECT
    public:
```

7.16 websocketclient.h

```
class WebSocketClient : public QThread {
    Q_OBJECT
public:
    explicit WebSocketClient(QObject* parent = nullptr);
    void run() override;
signals:
    void dataReceived(double pressure, double temperature, const QString& flag);
};
```

Index

[32]

[32]Special thanks to Aditi Deshpande for her valuable guidance and clear explanation on how to generate Doxygen files.

Database Source Files

AK |

```python
"""ESP32 to InfluxDB and Socket Connection data bridge

This script reads JSON-formatted sensor data from an ESP32 over a serial
connection, parses and formats the data, writes it to InfluxDB, and transmits
it to a local client over a UNIX domain socket.

Requirements:
    - InfluxDB client library
    - A running InfluxDB instance with a configured bucket
    - ESP32 sending JSON-encoded data via serial
"""

import serial
import json
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS
from datetime import datetime
import socket
import os


# InfluxDB settings
URL = 'http://localhost:8086'
ORG = 'Kaffeknekt'
TOKEN = 'ETV_6VBhkfF7HzNGfOjN6F7nTvX0ye_tblcGObcB1OVJDLYxQXUWpt8NU84PJmrn6R6IV921X2eWLLJDg1wgdQ=='
BUCKET = 'sensor_data'

#Registers "address information" to connect to InfluxDB
client = InfluxDBClient(url=URL, token=TOKEN, org=ORG)

#Grants Write ability to the API
API = client.write_api(write_options=SYNCHRONOUS)

#Predetermined connection settings for Esp32
serial_port = '/dev/ttyUSB0'
baud_rate = 115200

def main():

    """Main loop to read, transform, and transfer sensor data from Esp32.

    Opens a serial connection to the Esp32 and a UNIX socket
    for a connection to a client. Continuously reads JSON-formatted data
    from the Esp32, converts timestamps, writes data to InfluxDB, and
    forwards the data to a connected client.

    Steps:
        - Sets up a UNIX socket with filepath and waits for client
        - Reads serial input line-by-line
        - Converts JSON strings into Python dictionaries
        - Extracts and formats timestamps
        - Structures points with serial data and writes to InfluxDB
        - Sends the same data through the socket connection to a connected client

    Raises:
        KeyboardInterrupt: If execution is manually stopped
        JSONDecodeError: If serial input cannot be parsed, i.e because of bad format
        KeyError: If certain fields are missing from data
        SerialException: If there's an error with the serial port, i.e no connection
    """

    #Define socket path
    path = '/tmp/socket'

    #unlinking 'path' to potentially existing filepath
    if os.path.exists(path):
        os.unlink(path)

    #Creating a UNIX socket for local client
    server = socket.socket(
        socket.AF_UNIX, #Declares a local socket
        socket.SOCK_STREAM #Ensures secure connection-based communication between server and client
    )

    #Assigns path to the socket
    server.bind(path)

    #Sets the server to listen mode for 1 client to connect
    server.listen(1)

    try:
        #Opens port to esp32 temporarily
        with serial.Serial(serial_port, baud_rate, timeout=1) as ser:
```

```
            print("Esp connected")

            print("Waiting for client connection")

            #Waits for client to connect
            connection, cli_addr = server.accept()
            print("Connected:", cli_addr)

            while True:
                #reads line, converts byte to string, and cleans excess characters like \n
                line = ser.readline().decode('utf-8').strip()

                if line: #checks if line exists
                    try:
                        #Convert json string to python dictionary
                        data = json.loads(line)
                        print(data)

                        #Turns nanoseconds into seconds and removes microseconds
                        stamp = datetime.fromtimestamp(int(data["timestamp"]) / 1e9).replace(microsecond=0)
                        #convert to ISO format
                        iso = stamp.isoformat()
                        #remove T
                        date_part, time_part = iso.split("T")
                        #save as string
                        readable_time = f"{date_part} {time_part}"

                        #Initializing Point structurization
                        point = Point("Esp32Metrics") \
                        .field("readable_time", readable_time)


                        #Goes through and adds every item in the dictionary to the Point with defined value types
                        for field, value in data.items():
                            if field != "timestamp" and field != "flag":
                                point = point.field(field, float(value))

                            elif field == "flag":
                                point = point.field(field, str(value))

                            elif field == "timestamp":
                                point = point.time(int(value))

                        print(point)

                        #Tells the API to write the current Point to the bucket with nanoseconds presicion
                        API.write(bucket=BUCKET, org=ORG, record=point, write_precision='ns')
                        print("Data written to Influx")

                        #Adding readable time to the json string
                        data["readable_time"] = readable_time

                        #Converting python dictionary back to json string and adding newline for the recieving code to detect
                        socketstring = json.dumps(data) + '\n'

                        print(socketstring)

                        #Encodes the current JSON string and sends to the client
                        connection.send(socketstring.encode('utf-8'))

                    except json.JSONDecodeError:
                        print("Invalid JSON:", line)

                    except KeyError as e:
                        print(f"{e} is missing.")

    except KeyboardInterrupt:
        print("Stopped by user.")

    except serial.SerialException as e:
        print(f"Serial error: {e}")

    finally:

        #In the case of session termination, closes the connection in a secure manner
        connection.close()

        #Unlinks the current filepath from socket upon connection close
        os.unlink(path)

#Calls the main() function every time te script is being run
if __name__ == "__main__":
    main()
```

```
import "experimental"

option task = {name: "actually_actually_event_log", every: 22s}

kaffe =
    from(bucket: "sensor_data")
        |> range(start: -1m)
        |> filter(
            fn: (r) => r._measurement == "Esp32Metrics",
        //and r._field == "flaag" and r._value == "1",
        )
        |> pivot(rowKey: ["_time"], columnKey: ["_field"], valueColumn: "_value")
        |> filter(fn: (r) => r.flag == "1")
        |> drop(columns: ["flag"])
        //|> group(columns: ["_time"])
        |> experimental.unpivot()
        |> to(bucket: "event_data", org: "Kaffeknekt")
```

```
//A script for efficient querying that drops unnecessary values

from(bucket: "sensor_data")
	|> range(start: -2h)
	|> filter(fn: (r) =>
		r._measurement == "Esp32Metrics"
		)
	|> pivot(
		rowKey: ["_time"],
		columnKey: ["_field"],
		valueColumn: "_value"
		)
	|> drop(columns: [
		"_start",
		"_stop",
		"measurement",
		"readable_time"
		])
```

```
//A script that calculates daily power consumption

import "date"

from(bucket: "sensor_data")
  |> range(start: date.truncate(t: now(), unit: 1d))
  |> filter(fn: (r) => r._measurement == "Esp32Metrics" and
    r._field == "power"
    )
  |> sum()
  |> map(fn: (r) => ({
    _time: now(),
    _measurement: "dailyPower",
    _value: r._value,
    _field: "dailyPower",
    )})
```

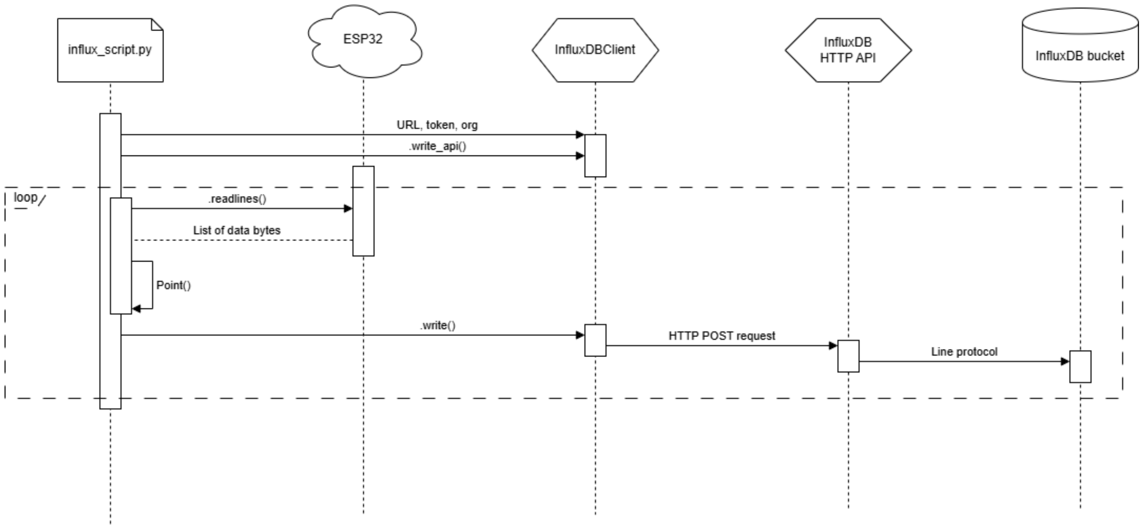


Figure C.4.1: InfluxDB Sequence Diagram

C.4.1 Data handling Sequence diagram **AK |**

This sequence diagram visualizes the communication between the python data handling script and the InfluxDB server as well as the Esp32.

- The sequence begins with the script establishing a connection to the InfluxDB server by creating a client with InfluxDBClient, which acts as an interface to InfluxDB.
- Using write_api(), we tell InfluxDBClient that we want to use a writing tool to write data.
- Inside a While loop, input data is recieved line-by-line, by reading data with readlines().
- The data inside the line is extracted, processed and used to build a Point() to write to InfluxDB
- With write(), the readied Point is presented to InfluxDBClient, which then sends a POST request to the InfluxDB API.
  Given that the identification and authorization info are correct, and the structured Point is valid (i.e correct value types and no mismatch with already existing fields), the API writes the data to the destination via line protocol.

GUI and ESP32 interface source files

Generated by Doxygen 1.13.2

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

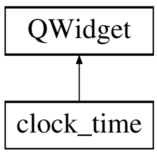
Chapter 4

Class Documentation

4.1 clock_time Class Reference

```
#include <clock_time.h>
```

Inheritance diagram for clock_time:



Public Member Functions

- clock_time (QWidget *parent=nullptr)

  Constructor for the clock_time class. Constructor that initializes the layout for the clock_time widget. (Placeholder version)

4.1.1 Detailed Description

Definition at line 8 of file clock_time.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 clock_time()

```
clock_time::clock_time (
            QWidget * parent = nullptr) [explicit]
```

Constructor for the clock_time class. Constructor that initializes the layout for the clock_time widget. (Placeholder version)

Definition at line 13 of file clock_time.cpp.

```
    : QWidget(parent)
{
    QVBoxLayout *layout = new QVBoxLayout(this);
    dateTimeEdit = new QDateTimeEdit(QDateTime::currentDateTime(), this);
    dateTimeEdit->setDisplayFormat("yyyy-MM-dd HH:mm:ss");
    QPushButton *setButton = new QPushButton("Set Time", this);
    layout->addWidget(dateTimeEdit);
    layout->addWidget(setButton);
    connect(setButton, &QPushButton::clicked, this, &clock_time::setTime);
}
```

The documentation for this class was generated from the following files:

- src/clock_Time/clock_time.h
- src/clock_Time/clock_time.cpp

Chapter 5

File Documentation

5.1 src/clock_Time/clock_main.cpp File Reference

```
#include "clock_time.h"
#include <QApplication>
```

Functions

- int main (int argc, char ∗argv[ ])

5.1.1 Function Documentation

5.1.1.1 main()

```
int main (
            int argc,
            char ∗ argv[ ])
```

Definition at line 5 of file clock_main.cpp.

```
00006 {
00007     QApplication a(argc, argv);
00008     clock_time w;
00009     w.show();
00010     return a.exec();
00011 }
```

5.2 clock_main.cpp

Go to the documentation of this file.

```
00001 #include "clock_time.h"
00002
00003 #include <QApplication>
00004
00005 int main(int argc, char *argv[])
00006 {
00007     QApplication a(argc, argv);
00008     clock_time w;
00009     w.show();
00010     return a.exec();
00011 }
```

5.3 src/clock_Time/clock_time.cpp File Reference

```
#include "clock_time.h"
#include <QVBoxLayout>
#include <QTimeEdit>
#include <QPushButton>
#include <QDebug>
#include <QString>
#include <QSerialPort>
#include <QJsonObject>
#include <QJsonDocument>
```

5.4 clock_time.cpp

Go to the documentation of this file.

```
#include "clock_time.h"
#include <QVBoxLayout>
#include <QTimeEdit>
#include <QPushButton>
#include <QDebug>
#include <QString>
#include <QSerialPort>
#include <QJsonObject>
#include <QJsonDocument>

clock_time::clock_time(QWidget *parent)
    : QWidget(parent)
{
    QVBoxLayout *layout = new QVBoxLayout(this);
    dateTimeEdit = new QDateTimeEdit(QDateTime::currentDateTime(), this);
    dateTimeEdit->setDisplayFormat("yyyy-MM-dd HH:mm:ss");
    QPushButton *setButton = new QPushButton("Set Time", this);
    layout->addWidget(dateTimeEdit);
    layout->addWidget(setButton);
    connect(setButton, &QPushButton::clicked, this, &clock_time::setTime);
}

void clock_time::setTime(){
    QDateTime dateTime = dateTimeEdit->dateTime();

    //Adds time values as a QJsonObject in correct format
    QJsonObject timeObject;
    timeObject["set_time"] = true;
    timeObject["year"] = dateTime.toString("yyyy");
    timeObject["month"] = dateTime.toString("MM");
    timeObject["day"] = dateTime.toString("dd");
    timeObject["hour"] = dateTime.toString("HH");
    timeObject["minute"] = dateTime.toString("mm");
    timeObject["second"] = dateTime.toString("ss");

    //Adds the QJsonObject to a QJsonDocument and converts it to a string
    QJsonDocument doc(timeObject);
    QString jsonString = doc.toJson(QJsonDocument::Compact);
    jsonString.append("\n");

    //Create a QSerialPort and defines the settings
    QSerialPort serial;
    serial.setPortName("/dev/ttyUSB0");
    serial.setBaudRate(QSerialPort::Baud115200);
    serial.setDataBits(QSerialPort::Data8);
    serial.setParity(QSerialPort::NoParity);
    serial.setFlowControl(QSerialPort::NoFlowControl);
    serial.setStopBits(QSerialPort::OneStop);

    //Opens the serial port and check if it opended correctly
    if(!serial.open(QIODevice::WriteOnly)){
        qDebug() « "Error: Port Not Open";
        return;
    }

    //Turns the QString int QByteArray and sends it to the ESP32
    QByteArray bytemessage = jsonString.toUtf8();
    serial.write(bytemessage);
    serial.flush(); //Is used to force the data to be sendt immediately
    serial.close();
}
```

5.5 src/clock_Time/clock_time.h File Reference

```
#include <QDateTimeEdit>
#include <QWidget>
```

Classes

- class clock_time

5.6 clock_time.h

Go to the documentation of this file.

```
#ifndef CLOCK_TIME_H
#define CLOCK_TIME_H

#include <QDateTimeEdit>
#include <QWidget>


class clock_time : public QWidget
{
    Q_OBJECT

public:
explicit clock_time(QWidget *parent = nullptr);

private slots:
void setTime();

private:
    QDateTimeEdit *dateTimeEdit;
};
#endif // CLOCK_TIME_H
```

5.7 src/SocketData/main.cpp File Reference

```
#include <QCoreApplication>
#include <QDebug>
#include <iostream>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <cstring>
#include <string>
#include <nlohmann/json.hpp>
```

Macros

- #define SOCKET_PATH "/tmp/socket"

  Path for the UNIX domain socket. This path is used to create the socket and connect to it.

Functions

- int createSocket ()

    Creates a UNXI socket and connects it to the data handling server. Creates a UNIX domain socket, connects to the data handling server, and handles errors.
- void getData (int socket)

    Reads data from the connected socket. Reads data from the connected socket, adds the specified values to variables.
- int main ()

    UNIX domain socket client. Connects to a UNIX domain socket using createSocket() and receives data using getData().

5.7.1 Macro Definition Documentation

5.7.1.1 SOCKET_PATH

```
#define SOCKET_PATH "/tmp/socket"
```

Path for the UNIX domain socket. This path is used to create the socket and connect to it.

Definition at line 13 of file main.cpp.

5.7.2 Function Documentation

5.7.2.1 createSocket()

```
int createSocket ()
```

Creates a UNXI socket and connects it to the data handling server. Creates a UNIX domain socket, connects to the data handling server, and handles errors.

Definition at line 17 of file main.cpp.

```
00017                    {
00018     int qtSocket;
00019     struct sockaddr_un addr;
00020     qtSocket = socket(AF_UNIX, SOCK_STREAM, 0);
00021     if (qtSocket < 0){
00022         perror("Socket Error");
00023         return -1;
00024     }
00025     addr.sun_family = AF_UNIX;
00026     std::strcpy(addr.sun_path, SOCKET_PATH);
00027     if (connect(qtSocket,(struct sockaddr*)&addr, sizeof(addr)) == -1){
00028         perror("Connection Error");
00029         close(qtSocket);
00030         return -1;
00031     }
00032     return qtSocket;
00033 }
```

5.7.2.2 getData()

```
void getData (
            int socket)
```

Reads data from the connected socket. Reads data from the connected socket, adds the specified values to variables.

Definition at line 37 of file main.cpp.

```
00037                    {
00038     char buffer[1024];
00039     std::string rest;
00040     bool status = true;
00041     while(status){
00042         ssize_t dataReceived = recv(socket, buffer, sizeof(buffer), 0);
00043         if (dataReceived <= 0) break;
00044         rest.append(buffer, dataReceived);
00045 
00046         size_t pos;
00047         while ((pos = rest.find('\n')) != std::string::npos) {
00048             std::string data = rest.substr(0, pos);
00049             rest.erase(0, pos + 1);
00050             if (nlohmann::json::accept(data)) {
00051                 nlohmann::json jsonData = nlohmann::json::parse(data);
00052                 std::string flag = jsonData["flag"];
00053                 float pressure = jsonData["pressure"];
00054                 float temperature = jsonData["temperature"];
00055                 std::cout « "Received JSON: " « jsonData.dump(4) « std::endl; //Prints the received JSON (is not needed)
00056             } else {
00057                 std::cout « "Invalid JSON data: " « data « std::endl;
00058                 status = false;
00059                 break;
00060             }
00061 
00062         }
00063 
00064     }
00065 }
```

5.7.2.3 main()

```
int main ()
```

UNIX domain socket client. Connects to a UNIX domain socket using createSocket() and receives data using getData().

Definition at line 69 of file main.cpp.

```
00069           {
00070     int newSocket = createSocket();
00071     if (newSocket == -1){
00072         std::cout « "Error: Failed to create socket" « std::endl;
00073         return -1;
00074     }
00075     getData(newSocket);
00076     close(newSocket);
00077     return 0;
00078 }
```

5.8 main.cpp

Go to the documentation of this file.

```
#include <QCoreApplication>
#include <QDebug>
#include <iostream>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
```

```
#include <cstring>
#include <string>
#include <nlohmann/json.hpp>
#define SOCKET_PATH "/tmp/socket"
int createSocket(){
    int qtSocket;
    struct sockaddr_un addr;
    qtSocket = socket(AF_UNIX, SOCK_STREAM, 0);
    if (qtSocket < 0){
        perror("Socket Error");
        return -1;
    }
    addr.sun_family = AF_UNIX;
    std::strcpy(addr.sun_path, SOCKET_PATH);
    if (connect(qtSocket,(struct sockaddr*)&addr, sizeof(addr)) == -1){
        perror("Connection Error");
        close(qtSocket);
        return -1;
    }
    return qtSocket;
}
void getData(int socket){
    char buffer[1024];
    std::string rest;
    bool status = true;
    while(status){
        ssize_t dataReceived = recv(socket, buffer, sizeof(buffer), 0);
        if (dataReceived <= 0) break;
        rest.append(buffer, dataReceived);
        size_t pos;
        while ((pos = rest.find('\n')) != std::string::npos) {
            std::string data = rest.substr(0, pos);
            rest.erase(0, pos + 1);
            if (nlohmann::json::accept(data)) {
                nlohmann::json jsonData = nlohmann::json::parse(data);
                std::string flag = jsonData["flag"];
                float pressure = jsonData["pressure"];
                float temperature = jsonData["temperature"];
                std::cout « "Received JSON: " « jsonData.dump(4) « std::endl; //Prints the received JSON (is not needed)
            } else {
                std::cout « "Invalid JSON data: " « data « std::endl;
                status = false;
                break;
            }
        }
    }
}
int main(){
    int newSocket = createSocket();
    if (newSocket == -1){
        std::cout « "Error: Failed to create socket" « std::endl;
        return -1;
    }
    getData(newSocket);
    close(newSocket);
    return 0;
}
```

First App Example GUI

Coffee_machine_main

Release 1.0.0

Sokaina Cherkane

May 19, 2025

CONTENTS:

`main.get_resource_path`(*rel_path*)

Returnerer den absolutte filstien til en ressursfil.

Dette gjør det mulig å bruke ressurser (som bilder) både under utvikling og når programmet er pakket med f.eks. PyInstaller.

Parameters
`rel_path` (`str`) – Relativ sti til ressursfilen.

Returns
Absolutt sti til ressursfilen.

Return type
str

`class main.SimpleClock`(**args*, ***kwargs*)

Bases: `QLCDNumber`

A simple digital clock using QLCDNumber. Updates every second and hides the colon every other second.

`update_time`()

`class main.SimpleCoffeeMaker`

Bases: `object`

Simulates a basic coffee machine with ingredient stock and simple recipes.

`get_stock_report`()

Returns a report of current ingredient levels.

`get_recipe`(*kind*)

Returns the recipe of the specified coffee type.

Parameters
`kind` (`str`) – Type of coffee.

Returns
Formatted recipe or error message.

Return type
str

`can_make`(*kind*)

Checks if there are enough ingredients to make the coffee.

Parameters
`kind` (`str`) – Coffee type.

Returns
(bool, str) whether it can be made, and which ingredient is lacking.

Return type
tuple

`use_ingredients`(*kind*)

Deducts the required ingredients from stock.

Parameters
`kind` (`str`) – Coffee type.

`refill`()

Refills the stock to full capacity.

class main.**ReportWin**(**args*, ***kwargs*)

Bases: `QMessageBox`

Simple dialog to show a text report using QMessageBox.

class main.**CoffeeWindow**(**args*, ***kwargs*)

Bases: `QMainWindow`

Main application window for the coffee machine GUI. Provides buttons for making coffee, displaying a report, and refilling.

setup_ui()

Sets up the GUI layout and widgets.

start_coffee(*kind*)

Begins the coffee brewing process for the given coffee type.

Parameters

kind (*str*) – The type of coffee to brew (e.g., "espresso", "cappuccino").

If ingredients are insufficient, a warning is shown. Disables all controls and starts progress and sensor timers.

update_progress()

Updates the progress bar during brewing.

If brewing is complete, stops timers and finalizes the process.

update_sensor()

Simulates sensor updates with random temperature and pressure values.

Updates the sensor label with current simulated readings.

finish_coffee()

Finalizes the brewing process.

Deducts ingredients, shows a completion message, re-enables controls, and resets progress bar and title.

stop_process()

Stops the brewing process if in progress.

Resets progress bar, title, and re-enables controls.

disable_all()

Stops the brewing process if in progress.

Resets progress bar, title, and re-enables controls.

enable_all()

Enables all control buttons after brewing is done or stopped.

show_report()

Shows a message box with the espresso recipe and current ingredient stock.

refill_maker()

Refills the machine's ingredient stock to full.

Shows a message to confirm refill.

INDEX

Database log

AK |

11.03
Developed and tested a logic that sends registered temperature to InfluxDB. This has verified the functionality of the database (T-FUNC-1.1)
GitHub [33]

08.04
Developed a script that connects to an Esp32, takes in data in the form of json, decodes it, and transforms it into a python dictionary that can be worked with. It reads every key-value pair in the dictionary, such as temperature, power and timestamp, and writes to a bucket using Point(). This happens in a While loop, so it takes every incoming line as long as the script is running.
Used 'influx v1 shell' command to open an SQL shell to query data and verify that the data is being stored. There is a _time column that shows the time in UNIX format, which is not human readable
Tried to overwrite the timestamp with a human readable version using ISO format, but InfluxDB didn't accept it.

09.04
After assuming that the UNIX timestamp cannot be overwritten with a different format in the python script, I have decided that instead of overwriting the UNIX timestamps, it would be just as fine to save the human readable time in its own column in the bucket. Now we can read the exact time in influx v1 shell
This GitHub commit was made 1 day after the script in its entirety was developed. The readable_time part was added this day [34]

11.04
I have developed a flux query script that can be used in the terminal. Using keep(), it only shows the important columns such as time, temperature and power, as the other ones don't hold meaningful information, including readable_time, since it turns out that flux queries return _time in readable format.
GitHub [35]

14.04
I made the python script scalable so it reads the entire json line no matter how big it is, instead of fixed power, temperature and pressure parameters. It goes in a loop for every key-value pair in the dictionary, and as long as the key is not "timestamp", it writes to the influx point as a field, otherwise time.
GitHub [36]

15.04
I have worked on the sequence diagram, but more importantly researched on how the different components in the diagram actually work, such as write_api(), that tells the API that we want to write data from the script.

16.04
Finished the sequence diagram. Tried to query data with flux script before taking the easter holiday, but it didn't show up. I suspect this is caused by how InfluxDB handles null values, because when I query with the command 'influx v1 shell' and use the SQL interface, it shows everything.
GitHub [37]

22.04
I have tried querying newly generated data with the flux script in several ways: with pivot(), without pivot(), filtering for specific data, filtering for only measurement, setting range starting from

[33]https://github.com/martintara/kaffeknekt/commit/3e8558951af4c9ab9482d0b9150f61c6819d26f1
[34]https://github.com/martintara/kaffeknekt/commit/0673ad0f92ce84eff4f25eab455542ad74a3edc3
[35]https://github.com/martintara/kaffeknekt/commit/248c6d7b95b83c97c532458e4b49998afdc31a69
[36]https://github.com/martintara/kaffeknekt/commit/ddacbd76275529893b04bc9d34f046cd80f3909a
[37]https://github.com/martintara/kaffeknekt/commit/942c37e3d868c44b07b9831c96d285226ae4c110

100 years ago, dropping fields to simplify the query, querying in a specific range knowing those rows don't include null values. None worked. I triple checked the field names, the data-writing python script, the names in the query script, making sure they all match.

23.04
I have started developing a flux script that filters for power ¿ 0 and returns the timestamp of the first row after filtering. Then variables are assigned where one holds a timestamp with 5 seconds subtracted from the returned timestamp, and another that holds 30 seconds added. Finally, data will be selected with these modified timestamps so we will end up filtering event data and send it to a different bucket.

24.04
I have finished the event logging flux script. These are theoretical values, and this script was developed so I have a baseline to go off after we have determined how we will approach this task. This also needs to be tested, as there are little to no ways to check the syntax etc. GitHub [38]

25.04
I have researched on different useful functions and how to implement them. I discovered a function called StateTracking() that can track the duration of a row from a given value until a different value is detected. In addition, I have learned more about other functions such as map(), which allows you to modify a table of rows after filtering data.

28.04
Came across a small problem where if incoming parameter data were whole numbers, they would be detected as integers, and influx sets the field type to whatever type the first value was, namely a float. I had to specify that the value in the point was going to be a float.
GitHub [39]
Later I wrote a script that will automatically detect null values in a bucket and replace them with 0.
GitHub [40]

29.04
A new variable "flag" has been added to the incoming json lines, that will mark drastic change of pressure (for now, we need better machine activation cues) with U for up and D for down. Discovered a function events.duration() that can track the state of rows after filtering. We can use this to calculate the duration of machine activity and query data from registered flag = U, to its duration.
GitHub [41]

30.04
Rewritten the event logging script so one query (kaffe) will filter for flag = U -¿ D events with the help of events.duration() and extract their time, another query (monitor) will just query all available data, and use join.left() towards the "kaffe" query to end up with data rendered down to events where the machine has been used. Needs testing.
GitHub [42]

02.05
Had trouble querying with flux scripts, presumably because of the null problem whenever U or D are not written to flag. I have investigated how InfluxDB handles null values and combining tables of rows that contain null values.

05.05
Added logic that tries to write flag to influx if it exists, else it writes flag with "1", instead of looking for flag in the for loop.

[38] https://github.com/martintara/kaffeknekt/commit/27f283f22c97efafeb46edcfdb0e3a56bb48e883
[39] https://github.com/martintara/kaffeknekt/commit/e6b34a9f56de3c91dc8726de4a1096bde28944f6
[40] https://github.com/martintara/kaffeknekt/commit/1c1329c8a4a9e1c46b105fb582ec5e408165eec8
[41] https://github.com/martintara/kaffeknekt/commit/3eb2ab3c2e01e596e2b8467721305c2c8a756967
[42] https://github.com/martintara/kaffeknekt/commit/2392ac7592c01a8b3ece4d66fe5c270e484e6fca

GitHub [43]

Added clock logic that adds milliseconds to timestamps, so that influx lines with the same timestamps don't overwrite each other. It also subtracts 2 hours from the timestamps as a temporary solution to the random data generator script sending future timestamps.
GitHub [44]

Changed join condition to time column instead of measurement column because I realized the time is not precise enough for the value matching to be a problem.
06.05
Added tag set implementation, which was quickly taken out as I deemed it unnecessary since flag does the same thing: mark machine activity. Added functionality that detected latest incoming flag value and filled subsequent rows with that value.

07.05
I tried to fix the event script that worked with events.duration(), that was supposedly going to detect when the flag was going to give "high" value, and count until it detected a "low" value, which was not the case at all. It simply returned the duration between value input timestamps. After realizing that, I tried using the duration to add to the query range to use as a stop line, so the script would correctly capture the scenario where the machine was active.

08.05
To avoid having to constantly query for new live data, we have agreed to integrate websocket functionality to the software architecture. My script will be the server side that waits for connection, then sends data in real-time.

GitHub [45]
After that, I worked with the flux script, and decided to drop the time range manipulation and decided to just join data based on flag="1".

GitHub [46]

I had trouble moving the finalized data to the other bucket, because of how pivot() handles the records, and buckets require input data to be in its original form. So I decided to also drop the join part of the script, and resorted to a simple unpivot() function, which did the trick. Although it is stated to be an experimental function, so with that I am carrying a small risk that it might be replaced with a different, more robust function in the future.

GitHub [47]

12.05
I have developed a simple flux script that calculates the sum of power usage every day. using date.truncate(), I can truncate the time to 12:00 pm every day, allowing the script to update the present day's sum of power usage in real-time.

[43] https://github.com/martintara/kaffeknekt/commit/e4f93896700dc21c736b4a7a264d6e588b02577d
[44] https://github.com/martintara/kaffeknekt/commit/69dab71794d00a93b7452a09d30221107c47a45a
[45] https://github.com/martintara/kaffeknekt/commit/52723cd677384abe46274065ab37a11bcb2e59cf
[46] https://github.com/martintara/kaffeknekt/commit/edaad350b042cc485894fc3cc2b18ee390c57608
[47] https://github.com/martintara/kaffeknekt/commit/2fd470a39ef772d31fa2f96483ab9f1843c591f9

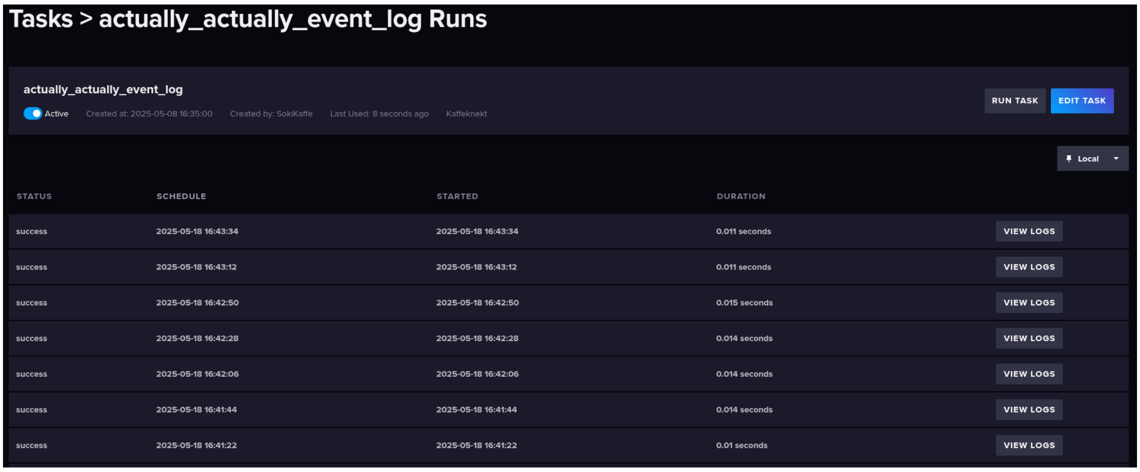
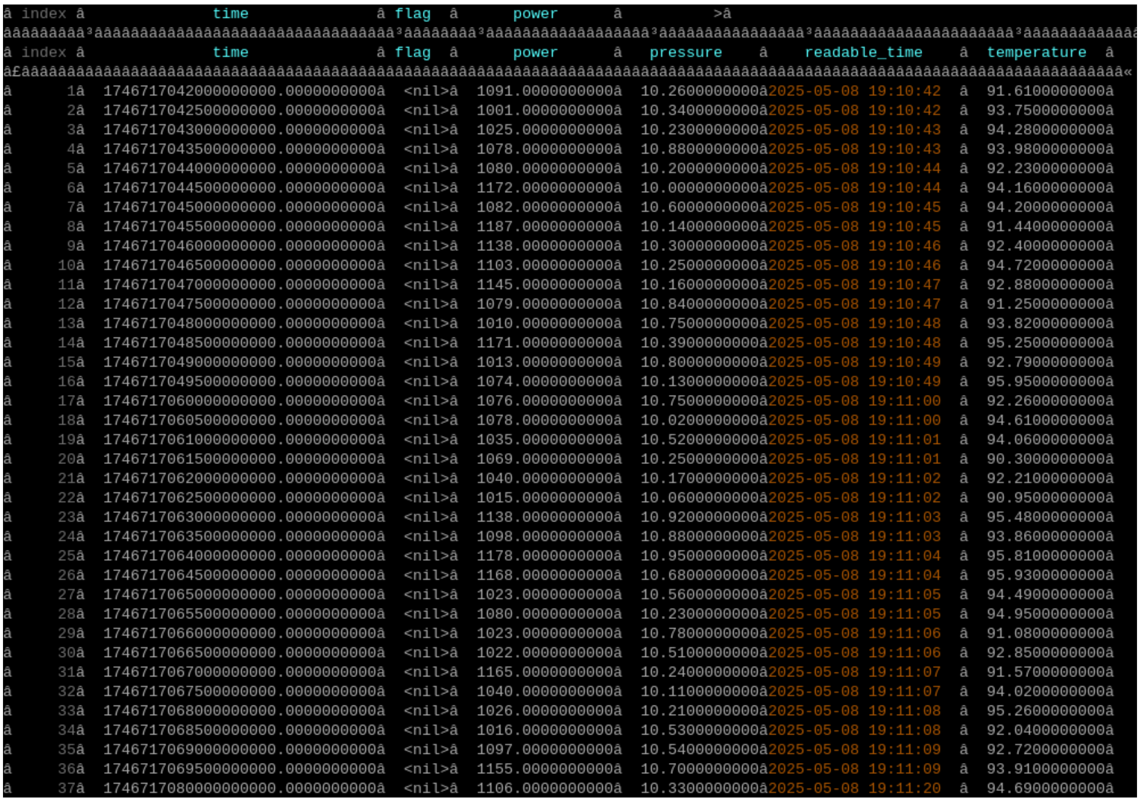


Figure C.6.1: Task run was a success!



index	time	flag	power	pressure	readable_time	temperature
1	1746717042000000000.0000000000	<nil>	1091.0000000000	10.2600000000	2025-05-08 19:10:42	91.6100000000
2	1746717042500000000.0000000000	<nil>	1001.0000000000	10.3400000000	2025-05-08 19:10:42	93.7500000000
3	1746717043000000000.0000000000	<nil>	1025.0000000000	10.2300000000	2025-05-08 19:10:43	94.2800000000
4	1746717043500000000.0000000000	<nil>	1078.0000000000	10.8800000000	2025-05-08 19:10:43	93.9800000000
5	1746717044000000000.0000000000	<nil>	1080.0000000000	10.2000000000	2025-05-08 19:10:44	92.2300000000
6	1746717044500000000.0000000000	<nil>	1172.0000000000	10.0000000000	2025-05-08 19:10:44	94.1600000000
7	1746717045000000000.0000000000	<nil>	1082.0000000000	10.6000000000	2025-05-08 19:10:45	94.2000000000
8	1746717045500000000.0000000000	<nil>	1187.0000000000	10.1400000000	2025-05-08 19:10:45	91.4400000000
9	1746717046000000000.0000000000	<nil>	1138.0000000000	10.3000000000	2025-05-08 19:10:46	92.4000000000
10	1746717046500000000.0000000000	<nil>	1103.0000000000	10.2500000000	2025-05-08 19:10:46	94.7200000000
11	1746717047000000000.0000000000	<nil>	1145.0000000000	10.1600000000	2025-05-08 19:10:47	92.8800000000
12	1746717047500000000.0000000000	<nil>	1079.0000000000	10.8400000000	2025-05-08 19:10:47	91.2500000000
13	1746717048000000000.0000000000	<nil>	1010.0000000000	10.7500000000	2025-05-08 19:10:48	93.8200000000
14	1746717048500000000.0000000000	<nil>	1171.0000000000	10.3900000000	2025-05-08 19:10:48	95.2500000000
15	1746717049000000000.0000000000	<nil>	1013.0000000000	10.8000000000	2025-05-08 19:10:49	92.7900000000
16	1746717049500000000.0000000000	<nil>	1074.0000000000	10.1300000000	2025-05-08 19:10:49	95.9500000000
17	1746717060000000000.0000000000	<nil>	1076.0000000000	10.7500000000	2025-05-08 19:11:00	92.2600000000
18	1746717060500000000.0000000000	<nil>	1078.0000000000	10.0200000000	2025-05-08 19:11:00	94.6100000000
19	1746717061000000000.0000000000	<nil>	1035.0000000000	10.5200000000	2025-05-08 19:11:01	94.0600000000
20	1746717061500000000.0000000000	<nil>	1069.0000000000	10.2500000000	2025-05-08 19:11:01	90.3000000000
21	1746717062000000000.0000000000	<nil>	1040.0000000000	10.1700000000	2025-05-08 19:11:02	92.2100000000
22	1746717062500000000.0000000000	<nil>	1015.0000000000	10.0600000000	2025-05-08 19:11:02	90.9500000000
23	1746717063000000000.0000000000	<nil>	1138.0000000000	10.9200000000	2025-05-08 19:11:03	95.4800000000
24	1746717063500000000.0000000000	<nil>	1098.0000000000	10.8800000000	2025-05-08 19:11:03	93.8600000000
25	1746717064000000000.0000000000	<nil>	1178.0000000000	10.9500000000	2025-05-08 19:11:04	95.8100000000
26	1746717064500000000.0000000000	<nil>	1168.0000000000	10.6800000000	2025-05-08 19:11:04	95.9300000000
27	1746717065000000000.0000000000	<nil>	1023.0000000000	10.5600000000	2025-05-08 19:11:05	94.4900000000
28	1746717065500000000.0000000000	<nil>	1080.0000000000	10.2300000000	2025-05-08 19:11:05	94.9500000000
29	1746717066000000000.0000000000	<nil>	1023.0000000000	10.7800000000	2025-05-08 19:11:06	91.0800000000
30	1746717066500000000.0000000000	<nil>	1022.0000000000	10.5100000000	2025-05-08 19:11:06	92.8500000000
31	1746717067000000000.0000000000	<nil>	1165.0000000000	10.2400000000	2025-05-08 19:11:07	91.5700000000
32	1746717067500000000.0000000000	<nil>	1040.0000000000	10.1100000000	2025-05-08 19:11:07	94.0200000000
33	1746717068000000000.0000000000	<nil>	1026.0000000000	10.2100000000	2025-05-08 19:11:08	95.2600000000
34	1746717068500000000.0000000000	<nil>	1016.0000000000	10.5300000000	2025-05-08 19:11:08	92.0400000000
35	1746717069000000000.0000000000	<nil>	1097.0000000000	10.5400000000	2025-05-08 19:11:09	92.7200000000
36	1746717069500000000.0000000000	<nil>	1155.0000000000	10.7000000000	2025-05-08 19:11:09	93.9100000000
37	1746717080000000000.0000000000	<nil>	1106.0000000000	10.3300000000	2025-05-08 19:11:20	94.6900000000

Figure C.6.2: Event sorted data

GitHub [48]
I have assigned retention period to the two buckets: 3 days for "sensor_data", and 6 months for "event_data" using commands from this page:
[44]

[48] https://github.com/martintara/kaffeknekt/commit/bcc9d285fe44c20a75baab348fc1cc2a2495f90a