Development of a Platform for Energy and Carbon Footprint Evaluation in Software Processes

Bachelor's Thesis in Computer Science and Engineering

Oliver Odhe, Albin Essman, Vilgot Teiffel, Fei Björnham, Herman Forsberg

DEPARTMENT OF Computer Science and Engineering

Chalmers University of Technology

Gothenburg, Sweden 2025

www.chalmers.se

On the cover: Illustration generated using ChatGPT (OpenAI)

Thesis for The Degree of Bachelor

Development of a Platform for Energy and Carbon Footprint Evaluation in Software Processes

Oliver Odhe, Albin Essman, Vilgot Teiffel
Fei Björnham, Herman Forsberg

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2025

Development of a Platform for Energy and Carbon Footprint Evaluation in Software Processes

Oliver Odhe, Albin Essman, Vilgot Teiffel
Fei Björnham, Herman Forsberg

Department of Computer Science and Engineering
Division of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg,
Sweden

Printed by Chalmers Digitaltryck,
Gothenburg, Sweden 2025.

Abstract

The global increase of Information and Communication Technologies (ICT) has led to a significant increase in energy consumption. Consequently, there is a growing demand to highlight and manage the environmental impact caused by the ICT sector. This project aims to bridge the gap between software development and sustainable computing by creating a computing platform capable of tracking the energy usage of running software processes in real time and evaluating their operational costs and carbon footprint. The approach to accomplish this goal began with selecting the Raspberry Pi 5 (RPi 5) as the hardware platform which allows for connection via Secure Shell (SSH) to our back-end. Additionally an Application Programming Interface (API) was integrated to retrieve real-time data of carbon footprint, forming an accurate foundation for calculating the conversion from energy to a carbon footprint.

Keywords: software sustainability, carbon footprint, energy consumption, real-time monitoring, Raspberry Pi 5 (RPi 5)

Sammandrag

Den globala ökningen av informations- och kommunikationsteknologi (IKT) har lett till en motsvarande ökning i energiförbrukning världen över. Det är därför viktigt att belysa och implementera åtgärder för att minska IKT sektorns miljöpåverkan. Syftet med projektet är att förenkla kopplingen mellan mjukvaruutveckling och hållbarhet genom att skapa en beräkningsplattform kapabel till att mäta en mjukvaras energikonsumtion i realtid, beräkna den totala operativa kostnaden samt beräkna det totala koldioxidavtrycket. För att utveckla detta system används en Raspberry Pi 5 som hårdvara vilket möjliggör anslutning till *back-end* systemet via *Secure Shell (SSH)*. Ett *Application Programming Interface (API)* integreras för att hämta data av koldioxidavtryck och elpriser. Datan används sedan för att beräkna den totala operativa kostnaden samt det totala koldioxidavtrycket.

Nyckelord: hållbar mjukvara, koldioxidavtryck, energikonsumtion, realtidsövervakning, Raspberry Pi 5 (RPi 5)

Acknowledgment

We would like to express our deepest gratitude to our supervisor Mateo Vázquez Maceiras for his invaluable guidance, support and encouragement through our bachelor thesis.

Oliver Odhe, Albin Essman, Vilgot Teiffel, Fei Björnham, Herman Forsberg

Contents

1

Introduction

The total energy consumption in the world has grown exponentially since the industrial revolution. In 2023, the primary energy consumption (energy required to meet society's needs before any transformation) globally reached 180 000 TWh [1]. Out of this, 20 000 TWh was energy in the form of electricity [2]. Although the transition from fossil fuels to renewable and sustainable energy sources is in progress, coal is still the dominant source of energy globally [1].

In a rapidly digitalizing world, the demand for Information and Communication Technology (ICT) services continues to grow. ICT's share of global energy consumption is expected to increase from 10% in 2018 to 20% by 2030 [3]. Thus, understanding and mitigating the environmental impact of software and computing has become crucial. Consequently, various practices have evolved to address this issue, but due to limited knowledge, many of these lack widespread integration [4]. Developers, researchers and organisations have limited visibility into how their applications contribute to energy consumption, making it difficult to implement optimisations. There is a substantial need for tools capable of analysing, quantifying and improving the energy efficiency of software applications in order to facilitate green practices within the field of ICT.

This project intends to enable developers, researchers and organizations to measure the real-time power consumption and carbon emissions of their software processes. By developing an open-source platform that simplifies energy usage and carbon emissions evaluation, the aim is to provide a solution that is accessible and practical to those seeking to implement green practices and optimise software performance.

1.1 Background

1.1.1 Energy Consumption Trends of ICT

In 2018, ICT accounted for 10% of the world's total energy consumption and it can be separated into three main categories: consumer devices, networking and data centres. By 2021, these three categories were split fairly evenly in terms of energy consumption. However, the expectation is that the electricity demands of data centres and networking will grow faster than the electricity needs of consumer devices. The complexity of systems is steadily growing with new technologies such as ChatGPT, and consequently the demand of electricity will grow [3].

The expected steady growth in energy usage is temporarily delayed. This is mainly because improvements in both system complexity and efficiency, to some extent, counteract each other, resulting in a lower rate of increase in total energy usage within ICT. However, this trend is unlikely to continue for long, as opportunities for further efficiency gains are becoming increasingly limited. Consequently, common efficiency improvements, such as shutting down and replacing older facilities with newer alternatives, may no longer be sufficient to counterbalance the ongoing rise in energy demand [2].

By 2030, the energy consumption of ICT is expected to account for around 20% of the global power consumption [3]. Therefore, it becomes important to adopt and develop strategies to minimise the energy usage of ICT.

1.1.2 Green Software Engineering

To address the growing power consumption of ICT, various practices have emerged to overcome the challenge of minimising power consumption and emissions. These practices fall under the umbrella term, Green Software Engineering (GSE), which focuses on sustainability throughout the entire software life cycle. Green coding, a subset of GSE, is specifically relevant for this project as it focuses on making code environmentally friendly [5].

Implementing green coding is crucial for minimising the environmental impact of ICT. In the past, the industry has mostly focused on trying to minimise the impact of hardware. However, the energy consumption of hardware is dictated by its software, as it directly controls how the hardware operates. Software developers tend to rely on the availability of newer, more powerful and increasingly affordable hardware in the near future, which leads to a decreased effort to optimise the software for efficiency. This phenomenon is called "Software Bloat", and it often causes the hardware life cycle to be shortened. It increases energy consumption and carbon emissions, thus having a negative effect on sustainability [4] [6]. To minimise "Software Bloat" it is necessary to have the principles of green coding in mind when developing software. Green coding is concerned with optimising energy-efficiency in software and it does this by reducing the complexity of the software, optimising algorithms for energy consumption

and by choosing a programming language based on how much energy it consumes [5].

The practices of GSE and green coding are being poorly adopted in the ICT sector. According to D. Junger et al. [4] the field of GSE is growing and the research has made significant progress over the last few years. However, Junger establishes that the ICT sector is not adopting the research of GSE into their development process. He concludes that implementing techniques from GSE and Green coding into the curriculum for the future generation of IT experts is crucial. He writes that this step is necessary to bring light to sustainability in the industry and creating a workforce that can leverage the techniques to decrease power consumption [4].

There is an increasing demand for tools that can provide visibility into the environmental impact of software development, this to push sustainability within the ICT sector into the spotlight. Real-time tracking of power consumption, carbon emissions, and electricity costs can support developers in making more sustainability-conscious decisions. By surfacing these metrics, such tools can play a critical role in promoting green software practices across the industry.

1.1.3 Related Work

An example of a tool that tries to reduce the carbon footprint of ICT is Amazons service Energy Use Optimisation. It is a closed service provided by Amazon Web Services (AWS) which is the largest computing provider in the world. They provide a wide range of services that can be combined in order to optimise systems, manage data and/or create an application [7]. Energy Use Optimisation is one such service which present energy use and carbon footprint. It collects data in real time from connected devices and sensors. Data is then preprocessed and used in a machine learning process. The machine leaning model is built and trained to predict the best strategy [8].

Unlike AWS, our platform will be open source and self-contained, eliminating the need to combine multiple tools, which enhances usability. Our platform will present real-time data on energy usage and carbon footprint in order to raise awareness of software's environmental impact. In contrast from their approach, this project will not incorporate machine learning to provide suggestions, we solely focus on presenting the measured data.

1.1.4 Sustainability and Ethics

In the Brundtland Report of 1987 [9], sustainability was defined as "meeting the needs of the present without compromising the ability of future generations to meet their own needs". Today, it is common to divide sustainability into three dimensions: social, economic and environmental.

Sustainability is important in all fields, including the ICT sector. GSE and Green coding is mostly concerned with the environmental aspects of sustainability. Accomplishing this through shifting de-

velopers to focus on minimizing emissions and power usage by improving code and prolonging the life cycle of software. However, social aspects should not be discounted in the pursuit of sustainable and environmental improvement within the ICT sector. R. Verdecchia et al. [10] argue that an effort into improving environmental consciousness of both software developers and consumers will increase demand for sustainable products, hence driving up the importance of sustainable solutions. Other social factors that Verdecchia believes will contribute to sustainability in the ICT sector is design for reuse, sustainable ICT skills training and adopting strategies to bring awareness to the issues [10].

As the project aims to provide real-time insights into energy consumption and pricing, it also supports sustainability challenges by promoting efficient energy usage. This aligns with the United Nations Sustainable Development Goal 7 (SDG 7), which intends to "ensure access to affordable, reliable, sustainable and modern energy for all". Specifically, this project supports SDG 7.3, with the objective of doubling the global rate of improvement in energy efficiency by 2030 [11]. By supporting SDG 7, this project contributes to global sustainability efforts, and highlights the ethical responsibility to act in the interest of future generations.

By aligning with SDG 7, this project not only addresses energy consumption concerns but also acknowledges the broader ethical aspects of contributing to global sustainability. It addresses intergenerational ethics as it emphasizes the ethical obligation to act in the interest of future generations by counteracting resource depletion. Furthermore, it addresses environmental ethics by aligning with the ethical duty to prevent harm to the global environment.

1.2 Purpose

This project aims to bridge the gap between software development and sustainable computing by creating a computing platform capable of tracking the power usage of running software processes in real-time and evaluating the operational costs associated with those processes. The platform will combine real-time power consumption with an Application Programming Interface (API) that provides data on carbon emissions from electricity production depending on the region in which the electricity was produced.

Through this platform, users can evaluate and analyse the environmental impact of their software processes. By providing detailed, real-time insights, the project will equip developers, researchers and organisations with the necessary data to further align with the practices of GSE and sustainable computing. Additionally, the platform will be open-source, making it widely available for anyone to use and modify.

1.3 Scope

This section presents the functional specification of the platform to be developed as well as presenting the limitations of the defined scope. The scope and functional specifications has been designed to address the lack of an open source platform for monitoring and evaluating the energy consumption and carbon footprint of software processes in real-time. The platform is divided into three key components: data collection, data management and data visualization. The back-end handles data processing, storage and communication with the hardware, while the front-end displays information and enables interaction between the user and the system. Functionally, the system should be capable of measuring energy consumption in real-time, calculating corresponding emissions and presenting the results in an interpretable format.

1.3.1 Objectives

The primary objectives of this development-focused project are:

- **O1** Measure real-time energy consumption of software processes and collect region-specific carbon intensity and electricity price data from external APIs.
- **O2** Combine energy consumption, electricity price and carbon intensity data to generate key environmental impact metrics for a software process.
- **O3** Develop an interactive interface that presents collected energy consumption data and key environmental impact metrics in real time.

1.3.2 Limitations

While the platform successfully fulfils its core objectives, it does not include predictive capabilities or provide suggestions for sustainable improvements, as these were considered beyond the scope of the project. For instance, machine learning will not be implemented as it requires a large dataset for training and testing the model. Due to limited resources it is not feasible to track historical data or build such a dataset.

A further constraint stems from the project's dependence on publicly available APIs. As a result, the accuracy and reliability of the displayed data on the platform is dependent on the quality of the APIs. Solely APIs covering the EU were identified for this project. However, the platform can be adapted to suit any geographical area as long as API data is available in the desired area.

When measuring carbon emissions, it is typically divided into operational emissions, i.e., to run the software, and embodied emission, i.e., to produce the hardware. To calculate embodied emissions, it would be preferred to use a Life Cycle Assessment (LCA). A LCA is an extensive analysis that requires

information about the development stage and end-of-life stage. This can be difficult to obtain and is specific for each hardware device. Thus, it will not be a part of this project, only operational emissions of the hardware and the running software will be included.

Finally, it is important to note that the power consumption measured is closely tied to the specific functionality and capabilities of the hardware platform. Different devices support different instruction sets, power management features and levels of hardware acceleration. All of which influence how software runs and how much energy it consumes. As a result, power measurements are inherently platform-dependent and comparisons between different hardware setups may not yield meaningful conclusions without accounting for these architectural differences.

2

Theory

This chapter aims to give the reader insight and a better understanding of the different concepts that need to be addressed in order to build the platform. The carbon footprint, energy consumption, carbon intensity and electricity prices are reviewed.

2.1 Carbon Footprint

The term carbon footprint is widely used across diverse fields. However, there is some confusion and misunderstanding regarding its meaning and what it actually measures. The most accepted definition is that a carbon footprint quantifies the total, direct and indirect, greenhouse gas emissions associated with a product, service or activity. These emissions are expressed in terms of carbon dioxide equivalents (CO_2e), which account for the global warming potential of different greenhouse gases, enabling comparison among all emissions [12].

The ICT sector is estimated to account for 2% of the world's carbon emissions [13]. This can be compared to the share of global CO_2 emissions from aviation which during 2024 summed up to 2,5% [14]. Therefore, addressing the ICT sector's CO_2 impact is just as crucial as reducing emissions from air travel, in order to contribute to and fulfil agreements such as the Paris Agreement [13].

In the ICT sector, emissions can originate from manufacturing, energy consumption and end-of-life disposal. The most common source of emissions for software-related emissions are energy usage during operation. Existing approaches tend to consider only the energy use of the software itself, forgetting to consider the carbon intensity of the power mix. They are also relying on additional software modules to handle functions which in total might increase the energy consumption. Finally, many of these energy tracking and carbon emission calculators are not open source, which limits their accessibility and broader use [13].

Currently, several metrics of various quality are used to calculate carbon emissions. In 2023, an attempt to standardize the calculation of carbon emissions was made through the development of the Software

Carbon Intensity (SCI) metric. SCI can account for both embodied and operational emissions as well as explicitly analyse one of them. This level of robustness is unique for the metrics available today and in 2023 SCI was submitted to the International Organization for Standardization [15].

2.2 Energy Consumption

The environmental impact of software is primarily determined on its energy consumption [13]. Therefore, to assess this impact, it is necessary to measure the amount of energy used while the software is running. Three main approaches can be employed to accomplish this: hardware-based, software-based, and hybrid methods [16].

Hardware-based methods typically include a physical measurement device such as power meters, on-board multimeters or integrated measurement units. This method offers high precision, but is less scalable and does not isolate software energy usage. In contrast, software-based methods are more scalable but may suffer from reduced accuracy as it relies on estimations [16]. These estimations can be based on CPU performance, processor frequency or power models trained on prior hardware measurements [17]. The third approach is a hybrid approach that combines hardware monitoring and software analysis intending to achieve both high accuracy and enable scalability [18].

One hardware-based approach is to use a Raspberry Pi 5 (RPi 5) which have a Power Management Integrated Circuit (PMIC). The PMIC handles power distribution and monitoring, including measuring power consumption in real-time. The RPi 5 is an affordable, compact, open source computer [19]. Due to its versatility and accessibility, it is used across a wide range of fields, e.g. nanotechnology, astronomy, image processing, healthcare and energy management. It serves as a tool for both researchers and developers, and is also widely used in education to introduce programming to students. One of the key advantages of the RPi 5 is its low power consumption [20]. Therefore, when running software on the RPi 5, the total power usage can be approximated as the power consumption of the software itself.

2.3 Carbon Intensity

A software's environmental impact cannot be assessed solely based on its total energy consumption, since different energy sources have different associated emissions. Therefore, carbon intensity provides a more accurate measure of environmental impact [13]. As shown in equation 2.1, both the carbon emissions and the energy consumed must be known in order to calculate the carbon intensity.

$$\text{Carbon Intensity [g } CO_2 \text{ e/kWh]} = \frac{\text{Carbon emissions [g } CO_2 \text{ e]}}{\text{Energy consumed [kWh]}} \tag{2.1}$$

Here, the energy consumed refers to the energy used by the software. As mentioned earlier, software energy consumption can be measured in various ways, for example, by using a RPi 5. To estimate the carbon emissions, the energy mix (i.e., the share of different energy sources used) and their respective emission factors must be identified. Emission factors for various energy sources can be found in the IPCC *Special Report on Renewable Energy Sources and Climate Change Mitigation* (2011) [21]. The mix of energy sources for a country is usually provided by the local Transmission System Operators (TSOs). Another alternative is to use other global sources, for instance, Electricity Maps [22].

Electricity Maps is a company that provides carbon data. Their mission is data-driven decarbonization, meaning that data is analysed and used to implement methods that reduce the need for fossil fuels. The data is compiled from open government sources, transmission and distribution system operators and other companies that directly manage electricity. The data can be added by anyone and is then assessed by Electricity Maps according to their regulations to ensure quality of the data. In the free version of Electricity Maps, only one zone is available, while all data requires a paid subscription [22].

The data provided includes carbon intensity of a given electricity zone. Carbon intensity refers to the quantity of carbon dioxide released into the atmosphere, typically measured in grams per kilowatt hour of electricity consumed, i.e. the carbon footprint in gCO_2e (gram carbon dioxide equivalent). This metric is calculated based on the mix of power plants in the zone, taking into account both operational and life cycle emissions of the power plants. Transferred electricity from other zones are also included, thereby reflecting the origin of the consumed electricity. The emission value for each energy source are based on data from the IPCC's Fifth Assessment Report (2014) [22].

All data are real-time and updated every hour. If some data points are missing, for instance due to delays, approximations are used instead. The method of approximation varies depending on the nature of the missing data. A common method for real-time data is to use machine learning models to predict the mix of power plants and therefore the total carbon emission [22].

2.4 Electricity Price

To calculate the total cost of the electricity consumed by the software, electricity price data from different regions is required. Electricity prices can be measured in different ways, such as consumer prices or bidding prices (i.e., the trading price on the electricity market). In this case, it is reasonable to use the consumer electricity price, as it reflects what developers, researchers and organizations actually pay. However, consumer prices can be difficult to obtain, as they are often provided by local utilities and vary by location. If instead using the bidding price, this data can be accessed through the ENTSO-E Transparency Platform [23].

ENTSO-E Transparency platform is a company providing electricity prices in Europe [23]. The Eu-

ropean Network of Transmission System Operators for Electricity (ENTSO-E) is an association of European transmission system operators (TSOs) from 36 different countries in Europe. Their mission is to ensure the security and functionality in the European interconnected electricity market as well as enable the integration of renewable energy sources. All their data is accessible by requesting a security token [24].

The ENTSO-E Transparency platform provides historical, real-time and day-ahead electricity prices. In this project only real-time electricity prices, given by TSOs, from all available zones are used. The price is given in EUR/MWh (except for Ukraine where the unit is UAH/MWh) and any available zone can be selected. The electricity price zones is called Bidding zones according to the Energy Identification Codes (EICs) developed by ENTSO-E. ENTSO-E defines a bidding zone as "the largest geographical area within which market participants can exchange energy without capacity allocation". Thus, depending on the electricity market, a zone can either be a country or a part of the country [24].

3

Method

This chapter outlines our approach and the techniques that were used to build the platform. It provides an explanation of the methods used, the tools and technologies chosen, and the reasoning behind their selection. figure 3.1 shows a brief overview of our system architecture and operational strategy.

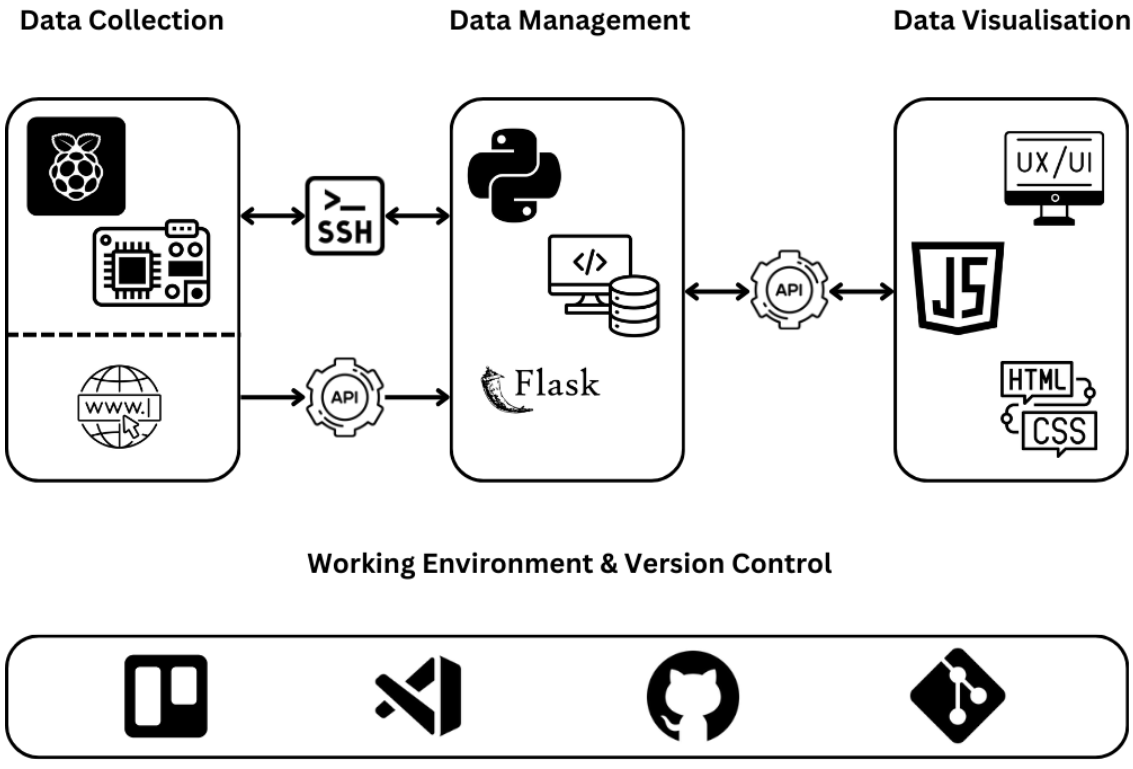

Figure 3.1: System Overview

3.1 System Architecture

Our system architecture leverages a diverse tech stack, where each component serves a specific role in fulfilling our objectives. When selecting components, we prioritised efficiency and alignment with the team's existing expertise, enabling a focus on implementation rather than extensive learning curves. By opting for familiar technologies, we aimed to streamline development and deliver a more complete platform within the project timeline.

3.1.1 Components

Our system architecture is composed of three main components: data collection, data management and data visualisation, each serving a direct role in achieving the project objectives. Specifically, the data collection component fulfils Objective **O1**, the data management component fulfils Objective **O2** and data visualisation component fulfils Objective **O3**.

The data collection component consists of two main subcomponents: a RPI 5, which serves as the hardware platform for power measurement, and the internet, which provides access to external APIs for environmental impact metrics. The RPi 5 utilises a custom shell script that wraps around the targeted script to measure power consumption in real time (for the full implementation of the script, see Appendix A). For external data, ENTSO-E was chosen to supply electricity price information, while Electricity Maps provides real-time carbon intensity data based on regional power sources. The decision to use two separate APIs was motivated by a couple of different factors. Firstly both APIs had a free version that were sufficient for the requirements of our platform. Secondly, no API could provide all the data required, hence two separate endpoints were necessary. Lastly both of these endpoints provide data that is regularly updated to reflect the current information, which facilitates the system to provide the most relevant and precise information possible. As this component includes measuring real-time power consumption data and gathering of external API data, it fulfils Objective **O1**.

The data management component processes the raw power consumption data collected by the RPI 5 and integrates it with external API data to generate key metrics. This component utilises Flask, a lightweight Python back-end framework, which supports data processing and API integration. The back-end collects power data from the RPi 5 and computes relevant metrics such as average power and total energy used. Concurrently, data from ENTSO-E and Electricity Maps APIs is fetched to provide electricity prices and carbon metrics, respectively. By combining the power data from the RPI 5 with the pricing and carbon data from the external APIs, we fulfil Objective **O2**.

The data visualisation component presents the processed data to the user through an interactive front-end. The interface is developed using HTML, CSS and vanilla JavaScript and is structured around three main elements: a customisable graph, a metrics board and a interactive terminal that replicates

the functionality of the RPi 5 terminal. The customisable graph can be configured to either visualise real-time power consumption or carbon emission of a software process, allowing users to track trends over time. The metrics board displays other key measurements related to the software process, such as total cost, average power and carbon intensity. Furthermore, the terminal is used to run scripts and monitor ongoing measurements. This component fulfils Objective **O3** by offering a clear, user-friendly interface for visualising energy data and environmental impact metrics.

3.1.2 Data Flow and Communication

We designed the system's communication architecture to centre around the data management component, which serves as the central hub for all data exchange. All data flows through the Flask back-end, ensuring consistent processing and integration across components. The system incorporates three primary communication pathways: between the RPi 5 and the back-end, between the back-end and the front-end, and from external APIs to the back-end.

The first and most critical communication pathway is between the RPi 5 and the back-end. The RPi 5 communicates exclusively with the back-end using Secure Shell (SSH). This communication is bidirectional: during measurements, power and output data are sent from the RPi 5 to the back-end for processing, while control commands can be issued from the back-end to the RPi 5 for remote management.

The second communication pathway is between the back-end and the front-end. Data processed by the back-end is transmitted to the front-end using a combination of Flask App Routing, Socket.IO and JavaScript Fetch API. Flask App Routing handles HTTP requests for initial data loads and static content, while Socket.IO establishes as persistent connection for real-time data streaming. Furthermore, JavaScript Fetch API is employed to request specific data updates or send user interactions to the back-end.

The third communication pathway involves external APIs. The back-end retrieves electricity price data from ENTSO-E and carbon intensity data from Electricity Maps via HTTP GET requests. This external data is combined with local power measurements to generate real-time environmental metrics.

3.2 Technical Implementation

The purpose of this section is to provide a detailed description of how we implemented each component of the system architecture, including the specific tools, libraries and programming methods used. While the previous section outlined the structure and data flow between components, this section focuses on the practical aspects of building and integrating those components.

3.2.1 Data Collection

As for the hardware data collection, the primary task was to enable the RPi 5 to transmit power data generated by the custom shell script to the back-end for further processing. As we opted to use SSH for communication, the first step involved activating SSH in the RPi 5 and ensuring it was configured to be on the same Local Area Network (LAN) as the host computer. This setup allowed the host computer to remotely access the RPi 5 and execute commands through a secure channel.

The second step centred around facilitating data exchange between the host computer and RPi 5. To accomplish this, we utilised Python's `paramiko` library to allow the host to execute commands and receive output data via Python. The SSH connection is initiated upon instantiation, with the provided hostname (IP address), username and password as input parameters.

Regarding the API data collection, we performed extensive preprocessing of the raw data to convert it into a structured format for further usage. As we utilised two separate external APIs with different data formats, two unique solutions were required for a successful implementation.

Data from ENTSO-E was delivered as an XML formatted file containing nested data structures. To extract the relevant pricing information, we used the Python library `BeautifulSoup`. The extracted data was then structured into a data frame using the `pandas` library, enabling the back-end to incorporate it for further processing.

In contrast, data from Electricity Maps was provided in JSON format, allowing for direct parsing using `pandas`. The carbon intensity values were extracted and organised into a data frame without requiring any prior parsing steps.

3.2.2 Data Management

To manage multiple data sources and parallel processes, we implemented a threading system. One thread maintains the SSH connection with the RPi 5 to continuously gather power data during measurements, while another thread processes the incoming data, computes metrics and transmits results to the front-end. This communication is handled by routing initial data (e.g. configure parameters such as region and measurement interval) through Flask's HTTP endpoints, while real-time updates are pushed via the Socket.IO thread, allowing the interface to reflect live system performance without interrupting user interaction.

3.2.3 Data Visualisation

The initial front-end design was developed using Figma (see figure B.1 in Appendix B) to sketch out the layout and functionality of the key components. We aimed to replicate this design as early as possible using Bootstrap, integrating its component library to structure a responsive and consistent

user interface. As the project progressed and new features were added, the front-end was regularly updated to reflect these changes and align with usability feedback.

The interface was iteratively refined during the development process, including the following.

- Integrating an interactive terminal that displays real-time output from the RPi 5 for live monitoring and debugging.
- Adding support for multiple graph types to improve data interpretability.
- Integrating drop-down toggle buttons for zone, price and country selection
- Replacing the on-screen display of output logs with a download option for logs and metrics.
- Implementing a dark mode for improved visual comfort in low-light conditions.

3.3 Operational Strategy

To support an efficient project execution, we decided at an early stage on an operational strategy that includes task management, prioritisation methods and version control. Each part of our strategy will be described in this section.

3.3.1 Work Process

Git and GitHub were used for version control. The main benefit of using Git and GitHub is its distributed quality, helping all team members keep a local version of the project that they then can synchronise with a centralised project that everyone has access to. The group decided to use a feature-branch-inspired approach, meaning that a new branch is created for every new feature. However, no strict guidelines were established for how pull requests and merging would be handled, nor were any naming conventions defined.

The project followed agile-inspired methodologies, to structure and manage tasks efficiently. Each week we had two recurring meetings. At the beginning of each week we reviewed the previous week's progress and planned and prioritised tasks for the days ahead. The new tasks were added as cards into a kanban board in *Trello* to effectively assign tasks. We also discussed possible improvements to the work done. At the end of the week we reflected on what had been accomplished and what remained unfinished. During these meeting we often focused on joining together the work done from earlier in the week. Hence, a lot of time was spent on merging pull requests and applying final touches to features.

3.3.2 System Requirements

To guide the design and development process, a set of system requirements was defined using the MoSCoW method (see table C.1 in Appendix C). This method categorizes each requirement based on its importance: MUST, SHOULD, COULD and WON'T. The requirements reflect functional and non-functional expectations across the system's back-end, front-end and communication layers.

3.3.3 Testing

During the development of the platform, testing was conducted continuously alongside implementation. The two primary testing domains are unit and integration testing. The unit tests ensure that individual components of the system works properly. Integration testing focused on verifying interaction between the system components, including, validating API tokens, establishing SSH connection, ensuring proper communication with the API endpoints and confirming that data could be retrieved as expected. If all tests passed successfully, the application was considered ready for use.

In addition to formal testing, informal debugging played a central role throughout development. A common approach involved inserting console print statements at key points in the code to trace program execution, verifying variable states and isolating runtime errors. This was particularly useful during the implementation of features like the download functionality and the real-time data display, where pinpointing the source of failure through formal test coverage was impractical. While not a substitute for structured testing, this method proved effective for rapid issue resolution during development.

4

Result

In this chapter, the results of the project are presented. The project resulted in a computing platform that enables users to measure the energy consumption and carbon footprint of software.

4.1 Platform Overview

When opening the platform, the user will see three main components; the carbon emission and power usage graphs, the metrics board and the RPi Terminal, as shown in figure 4.1.

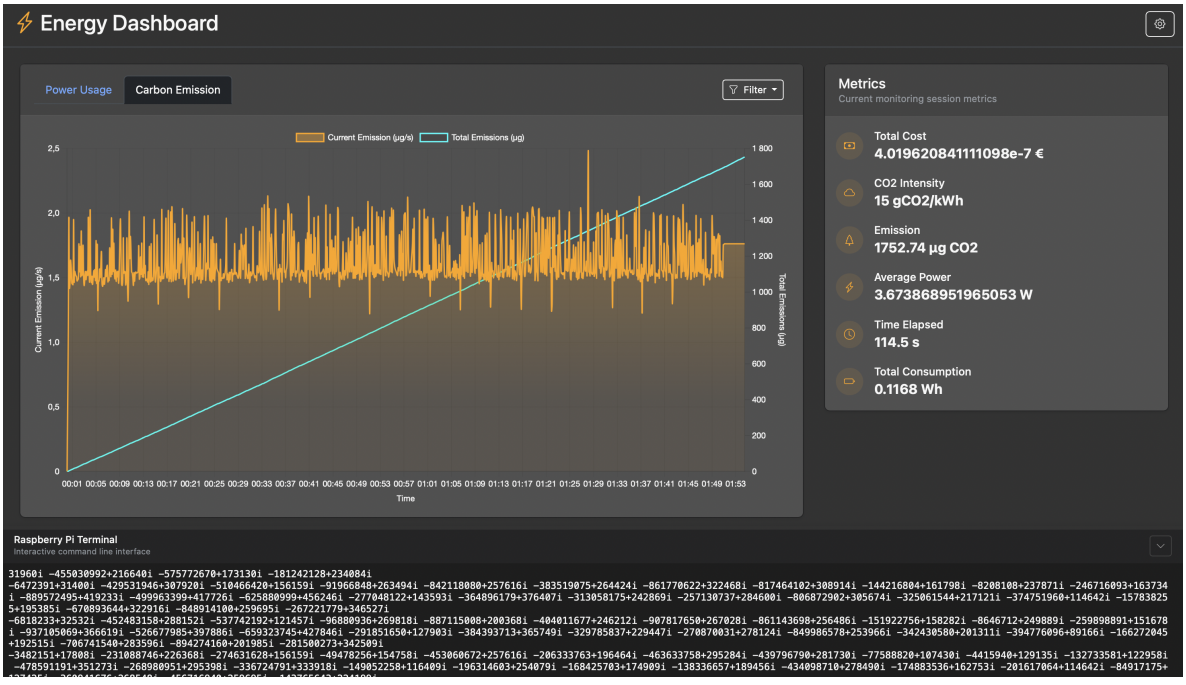

Figure 4.1: Platform overview with carbon emission graph

4.1.1 Graphs

The graphs display the data collected from the RPi 5 in real-time. The graphs are continuously updated with new data points to allow the user to observe trends in power consumption and carbon emissions generated by the running script.

There are two graphs that display different data. The first is the power usage graph (see figure 4.2), which shows the power consumption of each measurement in Watt (W) and the total power consumption in Watt hours (Wh). Power usage of each measurement is portrayed along the left y-axis with an orange graph, while the total power consumption is portrayed along the right y-axis with a blue graph. The second graph is the carbon emission graph, see figure 4.1. Similarly to the power usage graph, the carbon emission graph portrays the carbon emission of each measurement along the left y-axis with an orange graph. The graph also displays the total carbon emissions along the right y-axis with a blue graph.

Both the power usage graph and the carbon emission graph will automatically show both the orange and blue lines together, but the user has the option to deselect one of the graphs if their focus is limited to a single graph. To deselect one of the graphs the user has to click on the squares at the top-centre of the graph-window (marked in red), as shown in figure 4.2. To select the graph again you press the same button as to deselect.

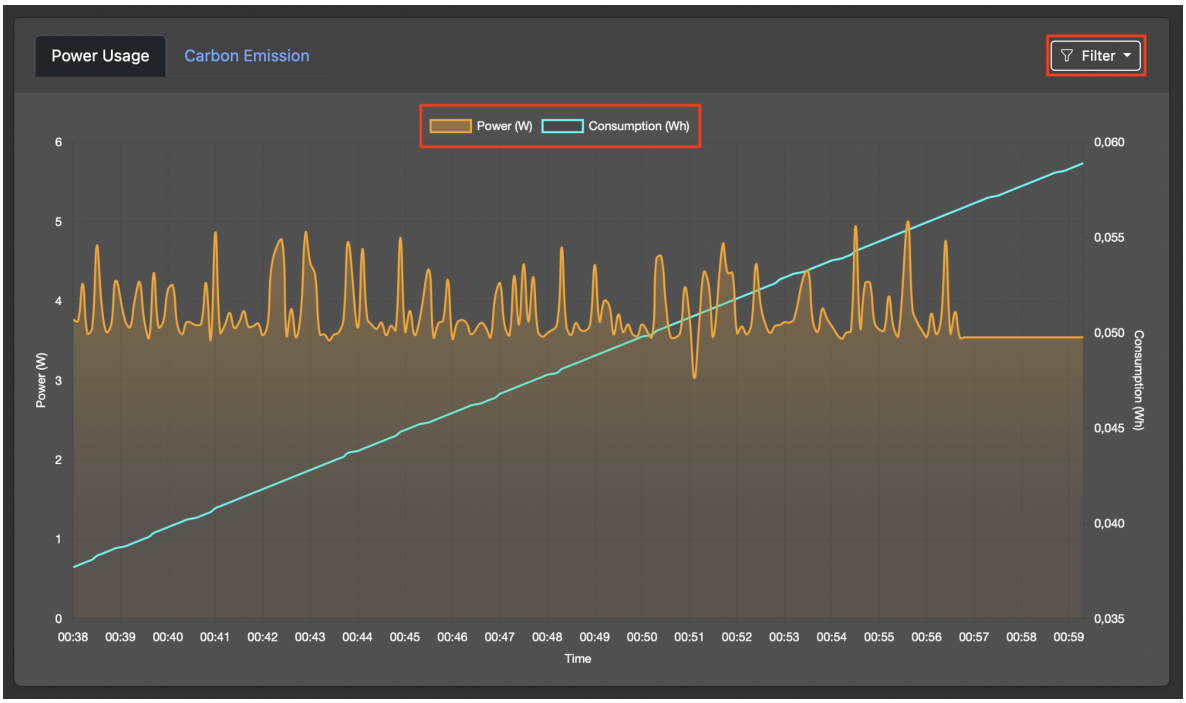

Figure 4.2: Power Usage Graph

The user can also choose different features for the graph by clicking on the filter button (figure 4.2), which opens a window, see figure 4.3. This window gives the user the option to select the maximum number of data points that will be shown in the graph from the start. Another option that the Filter window gives you is choosing if the graph should be interpolated, which smoothens the graph but decreases accuracy. Selecting the "Enable Fixed Width" enables the graph to scroll by displaying the most recent data and removing the older data points. The last option in the Filter window is choosing the frequency of how often new data should be gathered, the sampling frequency is given in Hertz (Hz).

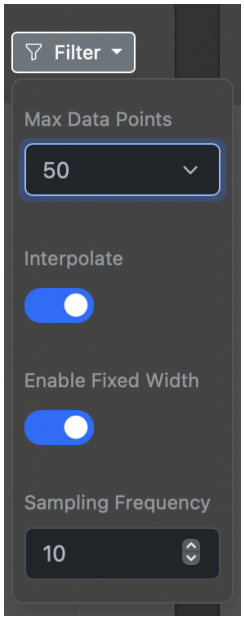

Figure 4.3: Filter

4.1.2 Metrics Board

The metrics board, in figure 4.4, is updated in real-time with data collected from the RPi 5 and they assist the user in getting an oversight of the current measurement data. The metrics that we display are:

- **Total cost:** Total cost of the energy used by the software since starting the computation.
- **Carbon intensity:** Carbon intensity (gCO_2/kWh) of the selected zone, since the carbon intensity is only updated every hour, this will be constant. This data is gathered from the Electricity Maps API. If no zone is selected in the Settings Tab (see section 4.1.4), then this metric shows the text *N/A*.
- **Emission:** Total CO_2 (μg) emitted by the software since starting the computation. Not selecting a zone also affects the emission metric by showing the text *N/A* instead of updating it in real-time, since emissions is dependent on carbon intensity.
- **Average power:** Taking all the data points that has been collected during the run time of the software and calculating the average power over this time period. This is measured in Watt (W).
- **Time elapsed:** Seconds elapsed since starting the computation.
- **Total consumption:** Total consumption of the software since starting the computation. Measured in Watt-hours (Wh).

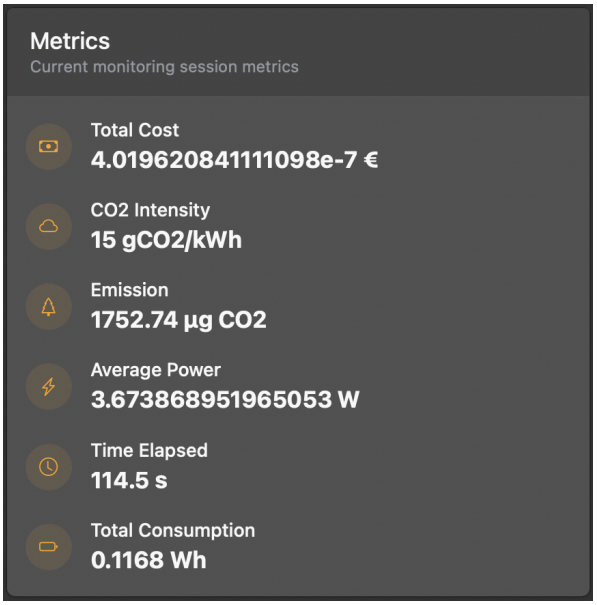

Figure 4.4: Metrics

4.1.3 Interactive Terminal

The interactive terminal, shown in figure 4.5, allows the user to interact with the RPi 5 through a simulated terminal that replicates the terminal of the RPi 5. All output from the RPi 5 is either used for the power measurement or is printed in the terminal, ensuring that the user receives all the information that is transmitted between the RPi 5 and their computer. Moreover, the terminal gives the user access to functions that would otherwise be laborious to replicate. As the terminal simulates the terminal of the RPi 5 the user has access to everything it would have access to if it were directly connected to the RPi 5, i.e., the user can change directories, run scripts without measuring the power consumption and additional terminal functionalities.

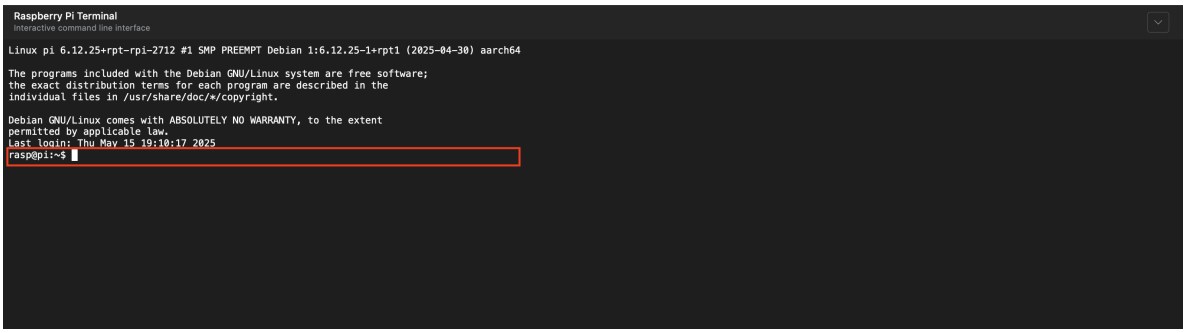

Figure 4.5: Interactive Terminal

4.1.4 Settings Tab

At the top right corner, the user can access *Settings* where it is possible to connect to the RPi 5 and select *country*, CO_2 *zone* and *price zone*, for the calculation of price and carbon emission, see figure 4.6.

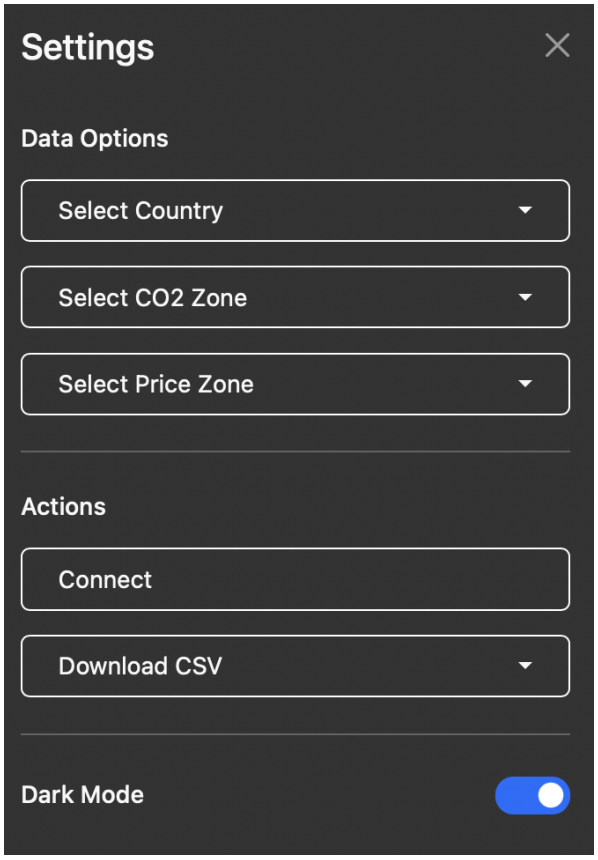

Figure 4.6: Settings

To select *Price zone* and CO_2 *Zone*, the user first has to select country in the drop-down menu labelled *Select Country* (figure 4.8). After selecting country the drop-down menu will be populated with the appropriate electricity price zones and carbon zones, from the selected country. The countries that are available for selection are the European countries offered by the API. This is due to the fact that ENTSO-E Transparency Platform, which is the platform that provides the API endpoint that we collect electricity price data from, only offers information about the European countries. We choose to not include other countries because we only want all available options to have data for both electricity price and carbon intensity.

To connect to a RPi 5 you use the *Connect* button in the *Settings* window. After pressing the button a module (figure 4.7) appears which asks the user to provide the information required to connect to a RPi 5 via SSH connection. This will be further explained in section 4.2.1.

To save the results after its execution, the user can use the *Download CSV* button. This will convert all the data points that has been measured into a CSV file containing the following information: sequential number of the measurement, date, elapsed time, power consumption and carbon emissions. When pressing the button you get the option to include or exclude the metrics (described in section 4.1.2) in the CSV file.

The last setting allows the user to enable or disable *Dark Mode*. This feature switches the colour theme, from lighter colours to darker ones, see figure 4.6. An explanation of how to set up the platform will be given below in section 4.2.

4.2 Running the Application

4.2.1 Connect to RPi 5

When launching the platform, the initial steps involves establishing a connection to the RPi 5. The connection process is facilitated through the *Settings* menu of the platform interface, where the user is presented with a connection button. The user will input three identifiers: hostname, username and password of the RPi 5, see figure 4.7. These identifiers must match exactly with those configured with the RPi 5 during the initial set up. In the event of a mismatch in any field, an error message will appear and instruct the user to revise the identifiers. Otherwise, the connection to the RPi 5 is successful, enabling further interaction with the platform.

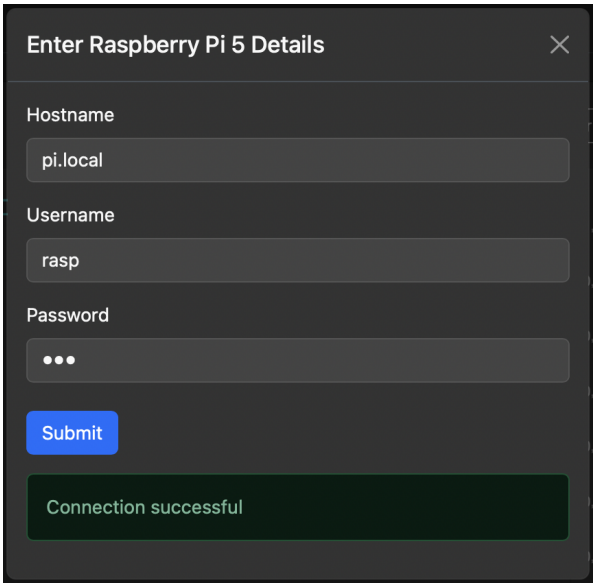

Figure 4.7: Login Interface

4.2.2 Select Zones

Once a successful connection to the RPi 5 has been established, the user can configure the environmental and regional parameters that will provide input parameters for calculating energy consumption. This process is again accessed through the *Settings* menu, where the user is provided with options to select country and a specific zone within that country for configuring both carbon intensity and electricity price (see figure 4.6). These selections are crucial, as they determine the relevant dataset for electricity price and carbon intensity value. The user typically begins by selection a country that corresponds to the physical location of where the platform is utilised (see figure 4.8). Following this, the user will select a geographical region within the country. This feature allows the platform to tailor its analysis by retrieving real-time data specific to the chosen location ensuring accurate metrics.

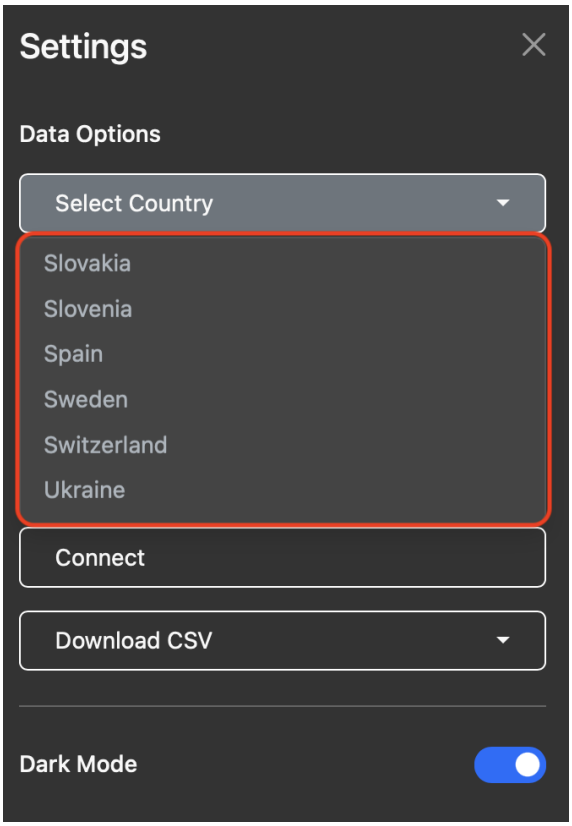

Figure 4.8: Region Selection

4.2.3 Compute Energy Consumption and Carbon Intensity

With the RPi 5 connected and the geographical zones selected, the platform is now ready for computations. The user navigates to the directory of the project they want to analyse, by using the interactive terminal. When the command *CO2* followed by a file name is entered the platform will begin to compute operational real-time price and carbon emission of the software, subsequently filling the graph with data points and updating the metrics. The steps outlined in sections 4.2.1 and 4.2.2 guarantee

that real-time data fetched from APIs is utilised for the calculations. Thereafter, when the project has finished running the measurement will automatically conclude, displaying the finalised graph and metrics.

4.2.4 Download Results

Following the computations, the platform allows the user to download the output logs and metrics in a CSV format. This option is available in the *Settings* menu, where the user can choose to either download both the output logs and the metrics, our just the metrics, as the output logs can be excessive, see figure 4.9. This allows for external analysis to be done.

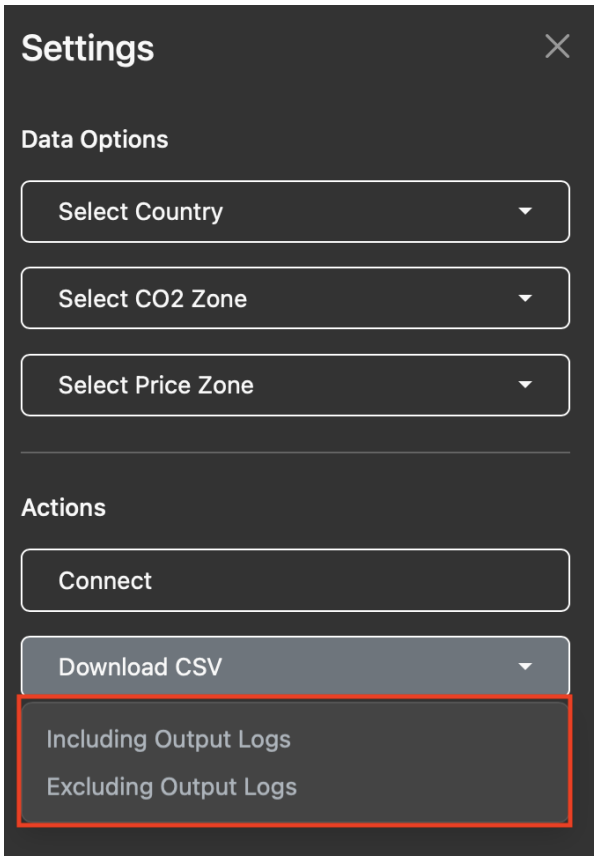

Figure 4.9: Download Interface

4.3 Use Case

This section seeks to demonstrate how the finished product can be used to guide developers, researchers and organisations through GSE practices by providing measurable insight into the environmental impact of their software. This demonstration is accomplished by comparing two sorting algorithms, binary insertion sort and insertion sort. Out of these, binary insertion sort is usually the faster alternative, whereas insertion sort is only faster in specific scenarios. Table 4.1 displays the emission

and energy data collected after running the two sorting algorithms, alongside the time it took for the sorting to be completed.

Table 4.1: Comparison of two sorting algorithms

Metric	**Binary Insertion Sort**	**Insertion Sort**
Total Emission ($\mu g\, CO_2$)	6790	9314
Energy Consumed (J)	964	1407
Time Elapsed (s)	278	386

To test the algorithms, both were given the task of sorting 100 000 random integers. These integers were provided in a text file that guaranteed both algorithms had identical inputs. From the result it is clear that the binary insertion sort completed the test substantially quicker than the insertion sort, which was to be expected. Consequently, both the emission estimate and energy consumption are higher for the insertion sort. This example illustrates how the platform enables developers to not only visualise and quantize the environmental impact of different software processes but also to make informed, sustainable decisions in their software design process.

5

Discussion and Analysis

In this chapter, we discuss the methods and results used in the development of the platform. We also evaluate how well the original requirements were met, identify possible improvements to the development process, and suggest directions for future work.

5.1 Requirements Evaluation

The system requirements (see table C.1 in Appendix C) were defined early in the development process using the MoSCoW method to categorize features based on their importance. This framework provided a clear structure for prioritizing implementation efforts. In this section, each category of requirements is reviewed to assess implementation outcomes and reflect on development decisions.

5.1.1 Summary of Fulfilment

As shown in table 5.1, all high-priority (MUST) requirements were successfully implemented, demonstrating that the core objectives of the system were met. Medium-priority (SHOULD) features were partially fulfilled, primarily due to complexity trade-offs. Optional (COULD) features were implemented selectively, based on feasibility. As planned, none of the excluded (WON'T) requirements were implemented, ensuring focus remained on defined scope boundaries.

Table 5.1: Summary of Requirement Fulfilment

Priority	**Total Requirements**	**Completed (%)**
MUST	9	100%
SHOULD	6	83%
COULD	7	43%
WON'T	4	0%

5.1.2 MUST Requirements

All MUST requirements were successfully implemented, fulfilling the core functionality of the system. These requirements defined the critical features necessary to establish real-time power monitoring, data transmission and effective front-end–back-end integration.

The most important and time-consuming tasks were requirements 1.1, 1.2 and 1.3, which formed the foundation of the system. These three requirements collectively represented the technical core of the system. Once they were completed, the remaining MUST requirements, such as back-end integration (1.6), frontend visualisations (1.8, 1.9) and third-party API integration (1.4, 1.5), were relatively straightforward to implement.

Overall, the successful fulfilment of all MUST requirements demonstrates that the Minimum Viable Product (MVP) goals were met. It also confirms the effectiveness of the MoSCoW method, which ensured that the most complex and critical components were addressed early in the development process.

5.1.3 SHOULD Requirements

The majority of SHOULD requirements were successfully implemented, with the exception of requirement 2.3. Overall, these mid-priority features were identified as valuable but not essential for the system's minimum viable functionality.

Requirement 2.3 involved integrating a database to store historical energy consumption data. Although this would have enabled trend analysis and long-term insights, it was omitted due to the added architectural complexity without a proportionate benefit.

The decision to dismiss requirement 2.3 and to keep the application stateless was made early in the development process. This decision brought with it positives as well as negatives. The positives being that no energy or time would have to go into creating a database structure to store data in, hence freeing up time in an already small time frame to focus on the main features of our application, the real-time electricity measuring. The negatives is that a feature that would make it easier to compare different measurements is not implemented.

The other SHOULD requirements, including features like cross-platform compatibility, a RPi 5 terminal interface and regional data selection, were successfully developed and integrated. Their implementation enhanced the system's usability and flexibility, although their absence would not have compromised core functionality.

5.1.4 COULD Requirements

The COULD requirements represented optional features that were considered desirable but not essential to the system's core functionality. Out of the seven COULD requirements defined, three were successfully implemented: 3.1, 3.2 and 3.6.

- 3.1 introduced a filtering option for customising data display. This allowed users to narrow down specific time ranges or metrics, enhancing the usability of the frontend interface.
- 3.2 added the ability to toggle between light and dark interface themes. Though purely aesthetic, it contributed to a more user-friendly and modern experience.
- 3.6 enabled users to download output data. This feature improved accessibility by allowing saving of system outputs.

The remaining COULD requirements, such as supporting additional devices (3.3), comparative regional analysis (3.4) and electricity cost insights (3.5), were not implemented. These were deemed to exceed the scope of this project and would have been too complex to implement in terms of data integration, UI design and system architecture.

5.1.5 WON'T Requirements

The WON'T requirements defined features that would enhance functionality of the application but were out of scope for this project. As planned, neither of these requirements were implemented and excluding them ensured that the focus remained on the core features. Although the WON'T requirements were not prioritized in this project, they are still relevant and serve as a good case for future development, which will be discussed more in section 5.6.

5.2 Insights on Data Collection

When observing and analysing the results it is important to note that the measured energy consumption includes that of the RPi 5 itself, not only the software. As a result, the total energy usage will not be entirely accurate. It also introduces a propagating error in the carbon footprint calculation and total cost. Given that the purpose of the platform is to raise awareness of a software's environmental impact and highlight the potential need for sustainable improvements, such a minor deviation is unlikely to significantly influence a company's willingness to make positive changes.

Another notable aspect to mention is that the electricity price occasionally can be negative, which may seem inconvenient. However, it can be easily explained by the principle of supply and demand. Typically, a product's price is determined by an equilibrium between supply and demand, but the key difference with electricity is its limited ability to be stored. As a result, when the demand is lower than

the supply it is not possible to store it and sell it later. Instead, the price decreases, sometimes falling below zero, in order to encourage usage. The electricity grid is highly sensitive to rapid fluctuations and an unstable distribution of power. Therefore, it is more efficient to consume the same amount of energy that is produced, as the system otherwise is at risk of failure, which is both expensive and time consuming to reset. With the increased reliance of weather-based energy sources, such as wind and solar power, fluctuations in the electricity grid are likely to become more frequent, resulting in more unstable electricity prices.

5.3 Credibility of APIs

The data of the electricity price and carbon intensity data are the basis for the platforms results. Thus, choosing reliable API sources are crucial for the platform's overall reliability.

In this project ENTSO-E Transparency platform is used. As they collect electricity price data directly from Transmission System Operators across Europe it is considers as a trustworthy source. It might be reasonable to believe that sources for electricity prices generally are easier to find, as pricing mechanisms are standardised in markets and closely monitored due to its direct economic impact on consumers and businesses. However, it is closely related to the structure of the local electricity market.

Carbon intensity, with the same reasoning, might be harder to find since the definition of carbon intensity is not as standardised a unit as price. For this project, carbon intensity data was gathered from Electricity Maps, which compiles information from government agencies, transmission and distribution system operators and other companies that directly manage electricity. There is no reason to believe that these sources are not credible but it is important to be aware that the quality might vary between countries and regions.

If the platform is to be extended to countries outside the scope of this project, it is crucial to assess the reliability of available data sources. Ideally, data should be sourced directly from governmental sources to ensure high accuracy.

5.4 User Interface

A potential flaw in the design of the user interface of the application is that no user testing was conducted nor any heuristic evaluation with external parties. Instead we tested the user interface ourselves and made improvements based on our own judgment. Hence, increasing the risk of potential bias and not including other perspectives than those of our own. This could implicate that the interface might not feel user friendly to the broad intended group. Certain features and small fixes increasing the usability of the platform could have been found and implemented through testing. The reason for not conducting any sort of testing of the user experience was due to an attempt to minimise the scope

of the project and focus more on the implementation of the product.

5.5 Open Source

One aspect that makes our platform unique compared to similar products is that it is open source and available on GitHub [25]. By making the code publicly accessible, the platform is free for anyone to use, build upon and improve. Subsequently making it accessible to educational institutions, allowing them to use it as a tool to teach the next generation of software developers about green coding practices. This approach not only supports ongoing development but also encourages knowledge sharing within the community. We hope that others will contribute and help create solutions that are both effective and environmentally responsible.

5.6 Future Development

While the core functionality of the system is implemented, there remain additional features that could enhance its capabilities. The modular architecture and open-source nature of the project provide a solid foundation for further development. This section outlines potential areas for future enhancements.

5.6.1 External Device Support

An important direction for future development is for the application to support more external devices. Currently, the design only provides support for monitoring the RPi 5. Expanding the system to support additional devices would massively improve the utility of the application.

Implementing support for more external devices would not only increase the functionality of the dashboard but also align the project with scalable system design principles, making it more applicable for further research and creates more use cases. Moreover, external device support would enable testing of more common programs and processes, one could for instance measure the power consumption of a LLM or compare power usage between IDEs.

5.6.2 Machine Learning Integrations

Another interesting area for future development is the integration of machine learning techniques into the dashboard. Currently, the application focuses on monitoring and visualizing real-time energy and power consumption data. By introducing machine learning, the dashboard could offer predictive tools to further enhance analysis. For instance, machine learning models could be trained to predict future energy consumption trends based on historical patterns or recommend optimisation strategies for reducing power usage.

5.6.3 API Improvements

As mentioned above, the project currently relies on data from two external APIs to display metrics. While these APIs provide useful information, they impose certain limitations. For instance, the API from Electricity Maps restricts data selection to a single country at a time, which limits the ability to compare metrics from different countries. Unless you have access to the paid version, which we did not have due to budget constrains.

A significant future improvement would involve integrating more flexible APIs that provide larger datasets or have less restrictions regarding queries. This enhancement would not only enable users to assess and compare the environmental impact of their power consumption across different countries but also open the door to a more detailed metrics view on the dashboard.

Integrating a new API into the system would be relatively straightforward, as it would primarily involve adjusting the data preprocessing stage to handle new data formats or structures. The modular design of the project makes it easy to swap and add new data sources.

5.6.4 Programming Language Optimisation

Sustainability is one of the main points in this project, but no attempt has been made regarding the sustainability of our own work. One possible improvement can consequently be to rewrite the code in another more energy efficient programming language such as C.

The current implementation prioritizes development speed and ease of maintenance, which is achieved by using a high-level language like Python. However, high-level languages tend to have a higher CPU usage than low-level languages and, consequently, have higher energy consumption.

Rewriting the application in a low-level language could significantly reduce the computational load, making the dashboard more sustainable and better aligned with the project's environmental goals. However, this would come at the cost of longer development times and increased complexity of the application.

5.6.5 User Interface Improvements

As discussed in section 5.4 no user testing of the interface was conducted, neither was any heuristic evaluation done. This means that while the interface may be functional from a developer's perspective, its actual usability and accessibility for end users remains largely unverified. It is left for future development to conduct testing and redesign the interface according to the received feedback.

5.6.6 Implementing a Stateful System

As discussed in section 5.1.3 the application is stateless as of now. While this design offers simplicity and scalability, it may hinder the application's ability to deliver more personalized functionality. A potential improvement would be to transition to a stateful architecture.

Implementing a stateful system would allow the application to maintain user-specific data, track session progress and store temporary states such as user preferences and login details. This could significantly enhance the user experience by reducing redundant operations and supporting features such as persistent sessions.

6

Conclusion

This project set out to address the growing demand for tools that facilitates sustainable software development by designing and implementing a platform capable of evaluating energy consumption and carbon emissions in real-time. By combining a Raspberry Pi 5, custom scripts and reliable third-party APIs, we developed a system that enables users to monitor power usage and its associated environmental impact through a user-friendly interface.

The platform successfully met its three primary objectives. The first objective **O1** was accomplished through measuring energy consumption, and collecting external carbon intensity and electricity price data. We reached objective **O2** by processing the collected data into energy consumption, electricity price and carbon emission metrics. Finally, objective **O3** was realized via the presentation of these metrics in an intuitive and interactive format. The open-source nature of the solution ensures accessibility, adaptability and potential for further development.

While certain limitations exist, such as the exclusion of machine learning predictions and a lack of access to APIs with fewer restrictions, the platform still offers a valuable foundation. Suggestions for further development include support for additional hardware devices, performance optimisation through more energy efficient programming languages and improvements in user experience testing.

In conclusion, this project contributes a meaningful tool to promote more sustainable development practices. It serves as a step toward bridging the gap between software engineering and environmental sustainability goals and encourages developers, researchers and organizations to make more conscious decisions about the energy and environmental costs of their code.

Bibliography

[1] (2024) Energy production and consumption. [Online]. Available: https://ourworldindata.org/energy-production-consumption

[2] N. Jones, "How to stop data centres from gobbling up the world's electricity," *Nature 561.7722*, 2018.

[3] A. Sampson, C. Delimitrou, K. Hazelwood, D. Brooks, G. Wei, and C. Wu, "Chasing carbon: The elusive environmental footprint of computing," in *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[4] D. Junger, M. Westing, C. Freitag, A. Guldner, K. Mittelbach, S. Weber, S. Naumann, and V. Wohlgemuth, "Potentials of green coding - findings and recommendations for industry, education and science," in *INFORMATIK 2023 - Designing Futures: Zukünfte gestalten*. Bonn: Gesellschaft für Informatik e.V., 2023, pp. 1289–1299.

[5] A. Bolla and A. M. Kandimalla, "Green coding practices in software development," Master's Thesis, Blekinge Institute of Technology, 2024.

[6] E. Kern, M. Dick, S. Naumann, A. Guldner, and T. Johann, "Green software and green software engineering–definitions, measurements, and quality aspects," in *First International Conference on Information and Communication Technologies for Sustainability (ICT4S2013), 2013b ETH Zurich*, 2013, pp. 87–91.

[7] (2024) Cloud computing with aws. [Online]. Available: https://aws.amazon.com/what-is-aws/

[8] (2024) Energy use optimization. [Online]. Available: https://aws.amazon.com/solutions/sustainability/energy-use-optimization/

[9] World Commission on Environment and Development, *Our Common Future*. Oxford: Oxford University Press, 1987.

[10] R. Verdecchia, P. Lago, C. Ebert, and C. de Vries, "Green IT and green software," *IEEE Software*, vol. 38, no. 6, pp. 7–15, Nov. 2021.

[11] (2024) Ensure access to affordable, reliable, sustainable and modern energy for all. [Online]. Available: https://sdgs.un.org/goals/goal7#overview

[12] O. Durojaye, T. Laseinde, and I. Oluwafemi, "A descriptive review of carbon footprint," in *Human Systems Engineering and Design II*, ser. Advances in Intelligent Systems and Computing, T. Ahram, W. Karwowski, S. Pickl, and R. Taiar, Eds. Springer, Cham, 2020, vol. 1026, pp. 960–968.

[13] S. Forti, J. Soldani, and A. Brogi, "Carbon-aware software services," in *Service-Oriented and Cloud Computing*, C. Pahl, A. Janes, T. Cerny, V. Lenarduzzi, and M. Esposito, Eds. Cham: Springer Nature Switzerland, 2025, pp. 65–80.

[14] H. Ritchie, "What share of global co_2 emissions come from aviation?" *Our World in Data*, 2024, https://ourworldindata.org/global-aviation-emissions.

[15] A. Schmidt, G. Stock, R. Ohs, L. Gerhorst, B. Herzog, and T. Hönig, "Carbond: An operating-system daemon for carbon awareness," in *Proceedings of the 2nd Workshop on Sustainable Computer Systems*. New York, NY, USA: Association for Computing Machinery, 2023.

[16] A. Noureddine, R. Rouvoy, and L. Seinturier, "A review of energy measurement approaches," vol. 47, no. 3, 2013, Conference paper, p. 42 – 49, cited by: 68; All Open Access, Green Open Access. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84901316580&doi=10.1145%2f2553070.2553077&partnerID=40&md5=e7ad9ded84795cab94da689f7cb9ca65

[17] M. Colmant, M. Kurpicz, P. Felber, L. Huertas, R. Rouvoy, and A. Sobe, "Bitwatts: A process-level power monitoring middleware," 2014, Conference paper, p. 41 – 42, cited by: 2; All Open Access, Green Open Access. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84985995577&doi=10.1145%2f2678508.2678529&partnerID=40&md5=df121c896826b167dbffe6d58d76d158

[18] C. Xian, L. Cai, and Y.-H. Lu, "Power measurement of software programs on computers with multiple i/o components," *IEEE Transactions on Instrumentation and Measurement*, vol. 56, no. 5, pp. 2079–2086, 2007. [Online]. Available: https://doi.org/10.1109/TIM.2007.904487

[19] R. P. Ltd, "We are raspberry pi. we make computers." accessed: April 9, 2025. [Online]. Available: https://www.raspberrypi.com/

[20] S. E. Mathe, H. K. Kondaveeti, S. Vappangi, S. D. Vanambathina, and N. K. Kumaravelu, "A comprehensive review on applications of raspberry pi," *Computer Science Review*, vol. 52, 2024.

[21] Intergovernmental Panel on Climate Change, "Special report on renewable energy sources and climate change mitigation: Summary for policymakers," 2011, accessed: 2025-05-15. [Online]. Available: https://www.ipcc.ch/site/assets/uploads/2018/03/SRREN_FD_SPM_final-1.pdf

[22] (2024) Electricity maps methodology. [Online]. Available: https://www.electricitymaps.com/methodology?utm_source=portal&utm_medium=referral&utm_campaign=datasets-info-methodology#introduction

[23] ENTSO-E, "Entso-e transparency platform," https://transparency.entsoe.eu/dashboard/show, accessed: 2025-04-14.

[24] (2025) Entso-e bidding zone configuration technical report 2025. [Online]. Available: https://eepublicdownloads.blob.core.windows.net/public-cdn-container/clean-documents/Publications/Position%20papers%20and%20reports/2025/ENTSO-E_Bidding_Zone_Configuration_Technical_Report_2025.pdf

[25] O. Odhe, A. Essman, V. Teiffel, F. Björnham, and H. Forsberg, "Environmental footprint platform," https://github.com/oliverodhe/environmental-footprint-platform, 2025.

A

Power Wrapper Script

```
Create temporary files

POWER=$(mktemp)

Get path of this file to avoid path errors

SCRIPT_PATH=$(dirname "$(realpath "$0")")

Start measuring power

"${SCRIPT_PATH}"/modified_power_sampler.sh $POWER &
PID=$!

Get start time in nanoseconds

start_time=$(date +%s%6N)

Start the actual program

$@

Kill power measurements

kill $PID

Get end time in nanoseconds

end_time=$(date +%s%6N)

elapsed_nsec=$((end_time - start_time))
elapsed_sec=$(awk "BEGIN {printf "%.6f",${elapsed_nsec}/1000000}")

Compute and display average power and energy
```

```
sed -i '$ d' $POWER

p=$(cat $POWER | awk '{sumX+=+$1;sumX2+=(($1)^2)}END{if (NR>1) printf "%.3f", sumX/(NR
    )}')
e=$(awk "BEGIN {printf "%.3f",${elapsed_sec}*${p}}")
rm $POWER

echo "Execution time = ${elapsed_sec}s"
echo "Average power   = ${p}W"
echo "Energy consumed = ${e}J"
```

B

Figma Design

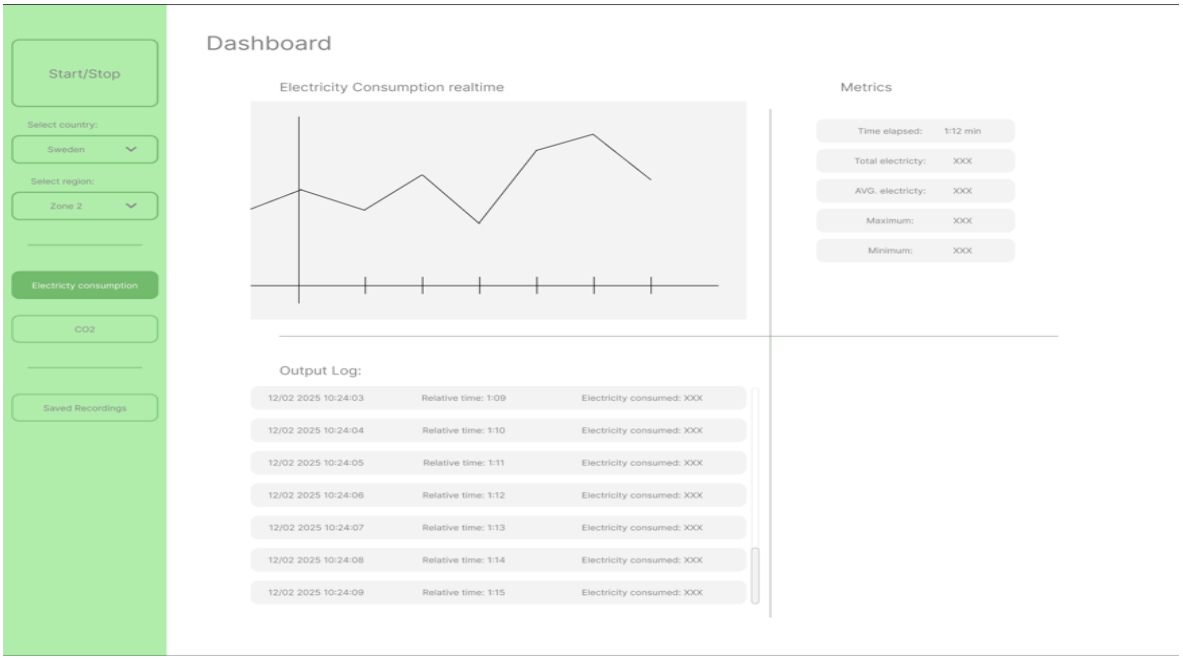

Figure B.1: Figma design of the interface

C

System Requirements

Table C.1: System Requirements Results

ID	Requirement	Completed
MUST		
1.1	A script must measure power consumption on a RPi 5 while other scripts are running.	[x]
1.2	A communication protocol using SSH must transmit data between the application and the RPi 5.	[x]
1.3	Real-time power consumption data must be provided from the RPi 5.	[x]
1.4	The system must integrate third-party API data to enhance electricity insights.	[x]
1.5	The system must integrate third-party API data to enhance energy insights.	[x]
1.6	A Flask-based backend must process and combine data from all sources.	[x]
1.7	A backend-for-frontend API must serve real-time backend data to the client.	[x]
1.8	The frontend must graphically display how power consumption changes over time.	[x]
1.9	The frontend must include a metrics board detailing power and electricity data.	[x]
SHOULD		
2.1	The frontend should include a terminal interface similar to that of the RPi 5.	[x]
2.2	A region selection option should be available for electricity and energy data display.	[x]
2.3	The backend should include a database for storing historical energy data to support trend analysis.	[ ]
2.4	Users should be able to initiate connections to the RPi 5 from the server.	[x]
2.5	The application should support MacOS, Windows, and Linux.	[x]

Continued on next page

ID	Requirement	Completed
2.6	The frontend should allow configuration of sampling intervals for RPi measurements.	[x]
COULD		
3.1	A filter feature could allow users to customize how data is displayed.	[x]
3.2	Users could toggle between light and dark themes.	[x]
3.3	The system could support power consumption measurement on devices beyond the RPi 5.	[ ]
3.4	Comparative analysis of power data between different regions could be supported.	[ ]
3.5	Information on electricity costs at different times could be provided.	[ ]
3.6	Users could download output data.	[x]
3.7	The RPi 5 could provide environmental data (e.g., CPU temperature, load).	[ ]
WON'T		
4.1	The system won't include machine learning for power consumption forecasting.	[ ]
4.2	The system won't include recommendation or optimization features.	[ ]
4.3	The application won't be hosted on an internet-accessible server.	[ ]
4.4	The application won't include any user account or login features.	[ ]