Term Project

Kubernetes Pod Scheduling based on Energy efficient Metrics

Semester: Fall 2024

Version: 1.0
Date: 2024-12-19 13:42:25Z

Project Team:	Marco Schnider
	Philipp Hutter
Project Advisor:	Prof. Laurent Metzger
Co-Advisor:	Fabio Daniel Marti
Technical Advisor:	Jan Untersander

School of Computer Science
OST Eastern Switzerland University of Applied Sciences

Abstract

Initial situation

In modern cloud environments, Kubernetes has become the de facto standard for orchestrating containerized applications. However, traditional Kubernetes pod scheduling algorithms primarily focus on resource allocation metrics such as CPU and memory utilization. While effective for performance, these methods often neglect energy consumption. This is where our project comes in. Our goal is to design a scheduler that works alongside the traditional scheduler, with a specific focus on renewable generated energy availability.

Approach and technologies

This project introduces an energy-aware Kubernetes pod scheduling approach that integrates energy availability metrics into the scheduling decision process. The proposed solution enhances the default Kubernetes scheduler by adding a new metric: the available energy for each worker node. For scheduling logic, we consider both the newly assigned priority of a pod and the available energy. Decisions are made based on these metrics, ranging from stopping pods to shutting down entire nodes. Additionally, we implemented a "move" function that allows Kubernetes pods to be manually transferred to another node. This feature demonstrates how workloads can be relocated seamlessly without noticeable service disruption.

Result

The project was deployed on a solar-powered infrastructure to validate its effectiveness. The system successfully managed pod scheduling by automatically shifting workloads when solar energy was unavailable, and the battery charge dropped to critical levels. Critical system parameters, such as battery thresholds, were made configurable and easily accessible through an intuitive graphical user interface (GUI). To enhance user engagement and improve target audience understanding, containerized workloads were represented as freight containers, giving the project a playful and visually intuitive design. These findings highlight the potential of energy-aware scheduling to improve the

sustainability and cost-effectiveness of Kubernetes-based cloud infrastructures, making it a promising step toward greener data center operations.

Management Summary

Introduction

Kubernetes is a system that helps manage and organize computer programs, which are often grouped together in so called pods. To make sure the programs run smoothly, it automatically places the pods on the right server, called a node, depending on available resources. If one of the nodes stops working, Kubernetes redistributes the pods onto different nodes. Given the surge in interest in renewable energy, the goal is to distribute the pods based on the energy available to the nodes connected to various renewable energy sources.

Goal

The goal of this project is to create a system that distributes pods onto different nodes depending on their current energy level. Additionally, a website will be created to showcase the scheduling process, allowing users to interact with Kubernetes in a playful manner. The website can then be used at career fairs to get people interest in the subject. It is designed to capture the attention of people walking by with its vibrant colors and engaging layout.

Approach

For the design of the website, the initial focus was on how to effectively display the information. Mockups were created to plan the layout of the different pages. The next step was to consider how to make the design engaging. The decision was made to design the main page like a container port, with container ships representing nodes and containers representing pods. This concept was implemented using modern web technologies, and the ships and container port were designed with Figma, a web application for interface design.

To manage the pod distribution, a system was created to handle scenarios when energy levels are running low. A priority system was introduced, where each pod could be assigned a priority that could be adjusted through the website. These priorities would

help determine which pods should be shut down or kept running when resources are limited. The system also ensures that pods are distributed across different nodes as needed. Pods that are turned off are only restarted once sufficient energy is available again.

Results

The result of the project is an eye-catching website which can be used to create interest in the subject. The use of vibrant colors and unique design will make sure it stands out among the abundance of colorful displays at career fairs or similar events. The custom scheduler, which distributes pods based on energy levels, demonstrates how computer science can be used to create innovative ways to use resources more efficiently. It also shows how to manage energy sources that can be unpredictable, such as solar panels, which depend on factors like time of day or weather conditions.

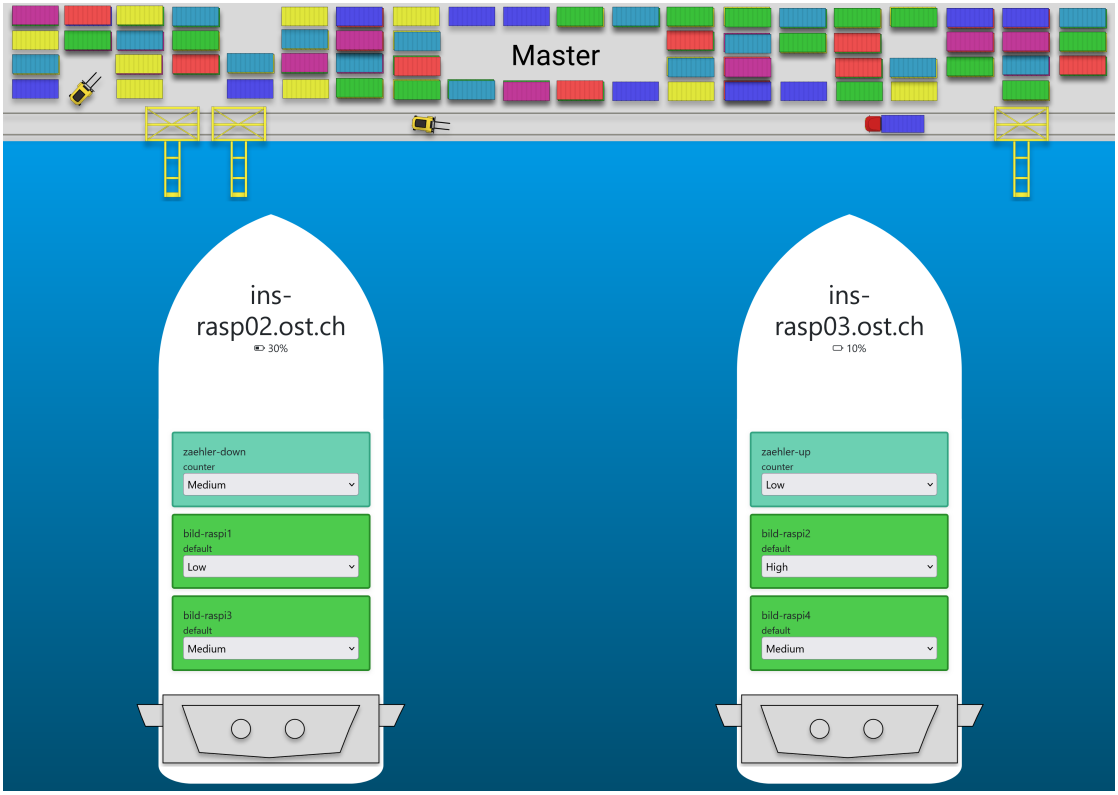

Figure 1: Overview showing the nodes and pods

The website gives people a practical example on how certain systems work. It keeps them engaged in a playful manner by allowing them to manipulate the system. Users can drag

and drop pods across different nodes or adjust certain thresholds used for the automatic scheduling. Additionally, the website provides useful information on the system that help to explain Kubernetes and scheduling.

Contents

Acronyms **x**

I Product Documentation **1**

1 Introduction **2**
1.1 Motivation . . . 2
1.2 Related work . . . 2
1.2.1 Research in scheduling to mitigate carbon emissions . . . 2
1.2.2 Research in emission-aware scheduling . . . 3
1.3 Differentiation from existing projects . . . 3

2 Goal of the project **4**
2.1 Task definition . . . 4
2.2 Goals . . . 4

3 General conditions **5**
3.1 Type of work . . . 5

4 Domain Analysis **6**
4.1 Domain model . . . 6

5 Requirements **7**
5.1 Functional Requirements . . . 7
5.1.1 Use case diagram . . . 7
5.2 Non-Functional Requirements . . . 12
5.3 User interface design . . . 15
5.3.1 Accessibility and Platform Independence . . . 15
5.3.2 Target Audience . . . 15
5.3.3 Core Functionalities . . . 15
5.3.4 Mockup . . . 15

6 Architecture **20**

6.1 C4 Architecture . . . 20
6.1.1 System context diagram . . . 20
6.1.2 Container diagram . . . 21
6.1.3 Component diagram . . . 22
6.2 Deployment . . . 23

7 Implementation Details **25**
7.1 Test Cluster . . . 25
7.2 GitLab CI/CD Pipeline . . . 25
7.3 Move pods . . . 26
7.3.1 Standard move . . . 26
7.3.2 Move with Deployments . . . 26
7.3.3 Move on the frontend . . . 26
7.4 Scheduler . . . 27
7.5 Dockerization . . . 28
7.6 Read energy . . . 29
7.7 Design . . . 29
7.7.1 Initial Approach . . . 30
7.7.2 Hybrid approach . . . 30
7.7.3 Result . . . 32

8 Results **34**
8.1 Achievements . . . 34
8.1.1 Functional requirements . . . 34
8.1.2 Non-functional requirements . . . 34
8.2 Shortcomings . . . 35
8.2.1 Functional requirements . . . 35
8.2.2 Non-functional requirement . . . 35

9 Conclusion and outlook **36**
9.1 Conclusion . . . 36
9.2 Outlook . . . 36
9.2.1 Cache energy data . . . 36
9.2.2 More complex scheduler . . . 37

II Project and Time Management **38**

10 Project Plan **39**
10.1 Collaboration . . . 39
10.2 Project Roles . . . 39
10.3 Minimum Viable Product (MVP) . . . 39
10.4 Long-term plan . . . 40
10.4.1 Milestones and goals . . . 41

10.5 Risk management . . . 42
10.5.1 Risk determination . . . 43
10.5.2 Contingency plan . . . 45
10.6 Jira . . . 46

11 Quality Measures **47**
11.1 Planning and Reviews . . . 47
11.2 Code Quality . . . 47
11.3 Branching and Peer-Review . . . 47
11.4 Test concept . . . 47
11.4.1 Process . . . 48
11.4.2 Goals . . . 48
11.4.3 Roles . . . 48
11.4.4 Environment . . . 48
11.4.5 Unit-Tests . . . 49
11.4.6 Schedule . . . 49
11.4.7 Risks . . . 50
11.4.8 Definition of Done . . . 50

12 Tooling **51**
12.1 Documentation Guidelines . . . 51
12.2 Code . . . 51
12.2.1 Code Guidelines . . . 51
12.2.2 Coding Setup . . . 52
12.3 GitLab . . . 52
12.3.1 Documentation Pipeline . . . 53
12.3.2 Dockerization Pipeline . . . 53
12.4 Issue Management / Time Tracking . . . 53
12.5 Teams . . . 53

13 Time Tracking Report **54**
13.1 Time per person . . . 54
13.2 Time per phase . . . 55

Bibliography **56**

List of Figures **58**

List of Tables **60**

III Appendix **60**

14 User Manual **61**
14.1 Installation of the ReadVoltage Service . . . 61

14.2 Prerequisites for Deployment 61
14.3 Deployment 62
14.3.1 Available Routes 62
14.4 Deployment of the Sample Apps 63
14.4.1 Available Routes 63

15 Test reports **64**
15.1 Test history 64
15.2 Usability-Test 65
15.2.1 Structure 65
15.2.2 Results 65
15.2.3 Conclusion 66
15.3 Usability-Test questions and answers 66

16 API Description **74**

Acronyms

ARM64 Advanced RISC (Reduced Instruction Set Computer) Machine 64-Bit. 25

CI/CD Continuous Integration / Continuous Deployment. 25, 28

CNI Container Network Interface. 23, 25

CSS Cascading Stylesheet. 30

GUI Graphical User Interface. 2, 11, 12, 13, 16, 29, 30, 40, 41, 49, 52

HTML Hypertext Markup Language. 30, 51

HTTP Hyper Text Transfer Protocol. 22, 23, 74, 75, 76

INS Institut für Netzwerke und Sicherheit. 26

MVP Minimum Viable Product. 39, 40, 42

RAM Random Access Memory. 25

RGB Red, Green and Blue. 30

VPN Virtual Private Network. 25

Part I

Product Documentation

Chapter 1

Introduction

1.1 Motivation

The motivation behind creating a Kubernetes pods scheduler based on energy efficient metrics stems from the rising interest and importance of renewable energy sources. Kubernetes is a widely used tool for container orchestration. Combining the rising interest of renewable energy sources with the popularity of Kubernetes leads to a project that can be of interest for different sectors and industries.

Apart from the scheduler another motivation of the project is to create a website to present Kubernetes and container orchestration to interested people at career fairs and other events. The aim is to give people the opportunity to engage with Kubernetes by playing around with the pods and nodes and learning the basics about how it works. The goal is to inspire attendees to consider pursuing a degree in computer science.

Both of these motivations coincide with the motivations of both team members. Marco is particularly interested in creating engaging GUI and frontend applications, while Philipp has a keen interest for network infrastructure. Additionally, both are eager to expand their knowledge in the other's area of expertise.

1.2 Related work

Considering the popularity of Kubernetes and interest in renewable energy it is important to consider the context of existing research and projects in the field. This section outlines related works that have informed the current project, providing an overview of the current state of research.

1.2.1 Research in scheduling to mitigate carbon emissions

There has been research and development in creating a scheduler based on energy data from renewable sources. One project for example distributes the pods in accordance to

the highest available share of renewable energy of a node in a cluster. Additionally, it takes weather conditions into account to make sure the applications are placed onto the optimal nodes in the long term. [5]

1.2.2 Research in emission-aware scheduling

Other research in the field focuses on emission-aware scheduling. Instead of just scheduling jobs onto nodes that are connected to renewable energy sources, it takes the current energy-mix into account. It looks at historical data of the power mix used by a datacenter and schedules jobs accordingly. [6]

1.3 Differentiation from existing projects

As outlined in the previous section, research on scheduling pods based on renewable energy measurements is an active area of study. This project differentiates itself by emphasizing the educational and presentation aspects of the topic. The primary goal is to develop an application that not only demonstrates Kubernetes and pod scheduling but also serves as an engaging learning tool to inspire interest in pursuing a degree in computer science. Therefore, our focus extends beyond the scheduler itself, prioritizing the creation of an interactive and captivating frontend to spark curiosity and enthusiasm for the subject.

Chapter 2

Goal of the project

2.1 Task definition

The initial task definition given was purposely defined loosely, to allow the team members to define themselves what they wanted to focus on. The main objective was to create a scheduler that schedules pods based on energy efficient metrics. As the project progressed it was made clear that one goal of the project should be to create an application that could be shown at career fairs.

2.2 Goals

Two goals were defined for the project. They are both of the same priority.

The first goal is to create a scheduler that takes energy measurements from renewable sources into account when scheduling the pods on a node. It should periodically check the energy levels of the nodes and act accordingly. Pods that need to be stopped when there is not enough energy should be restarted once the energy levels have recovered.

The second goal is to create a website that gives an overview over the Kubernetes-Cluster. The website allows the user to interact with the cluster by moving pods between the different nodes. Additionally the scheduler can be manually started and adjusted from the site. As mentioned before, the focus on career fairs led to certain adjustments in the goals. The website should have an eye-catching design and a playful element. It should also include some educational aspects to help interested people understand Kubernetes and container orchestration.

Chapter 3

General conditions

3.1 Type of work

This work is a student research project (term project) that was carried out in the fall semester of 2024. The planned time budget is 17 hours per week. With 14 weeks, that makes a total of 238 hours. The thesis is worth 8 ECTS credits.

Chapter 4

Domain Analysis

4.1 Domain model

The following graphic describes the domain model. The system has zero to many nodes that themselves have zero to many different pods that run on the nodes.

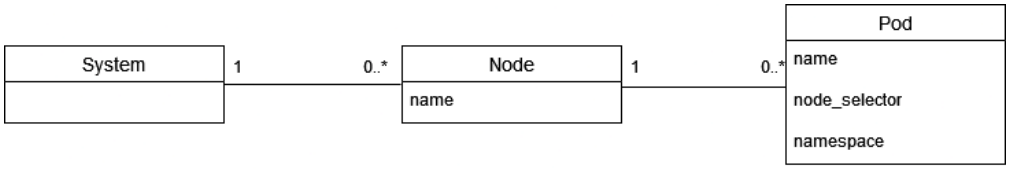

Figure 4.1: Domain model

Chapter 5

Requirements

5.1 Functional Requirements

The functional requirements include all planned functionalities. They are split into two categories. Requirements for the frontend and requirements for the backend. Additionally some functionalities are marked as optional, these will only be implemented if there is enough time.

5.1.1 Use case diagram

The only actor of the software is the user who interacts with it through the web application. The use case diagram shows the different interactions of the user with the website.

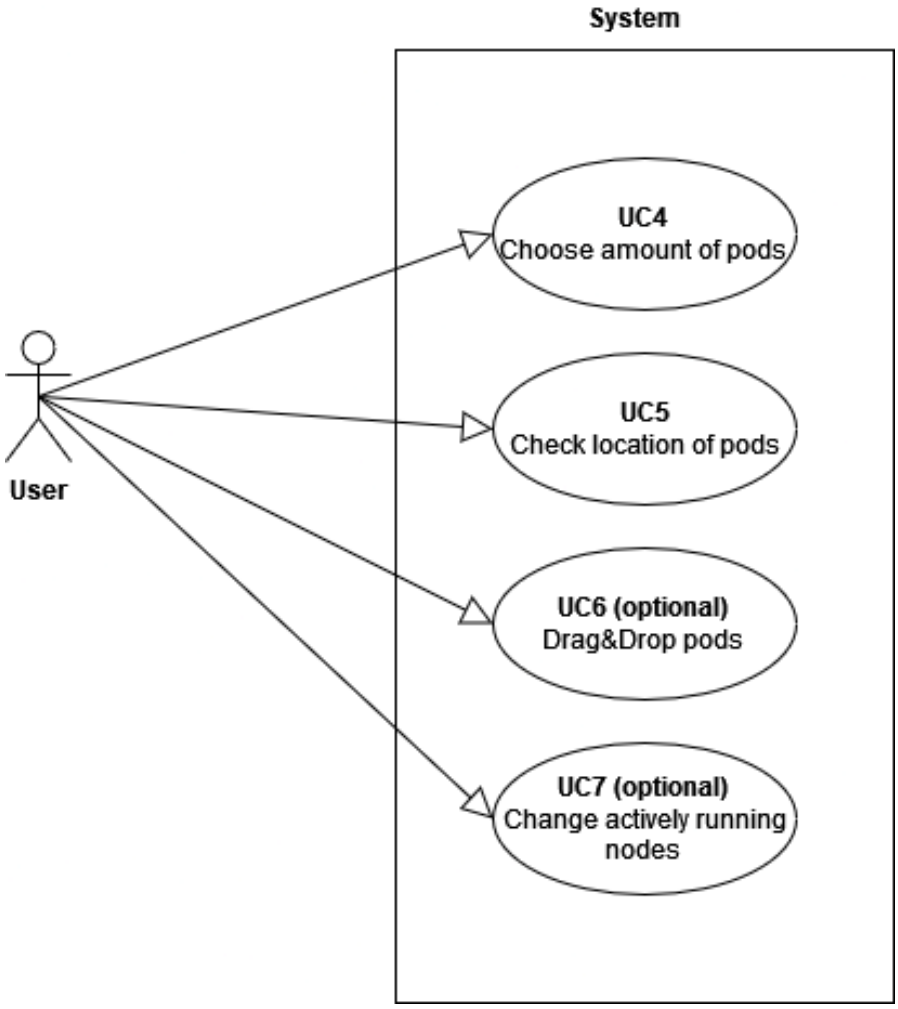

Figure 5.1: Use case diagram

Backend

The backend includes all interactions with the Kubernetes-Cluster and the scheduler logic.

ID	**UC1**
Name	Process energy measurement
Actor	System
Description	The program processes the energy measurement it receives from a specified source.
Precondition	Energy measurements exist.
Standard process	1. The program connects to a websocket or reads in a text file. 2. The measurements get processed.
Alternativ process	The measurements cannot get processed. The program throws an error.
Postcondition	The measurements can be used for further tasks.
Result	Reading the energy data was implemented, however the measurements takes a long time (1 minute) to compute. Considering this and the time constraint the final version of the project uses random numbers for the energy measurements. More is described in the implementation details.

Table 5.1: Description Use Case 1

ID	**UC2**
Name	Pods get deployed on nodes according to the energy measurements.
Actor	System
Description	The energy measurements get processed and the pods are efficiently distributed across the nodes.
Precondition	There are nodes to run the pods on. The amount of pods is specified.
Standard process	1. The program processes the amount of energy available on every node. 2. The program distributes the pods efficiently across the nodes.
Alternativ process	There is not enough energy to deploy the specified amount of containers. The program throws an error.
Postcondition	The specified number of pods get deployed on the nodes.
Result	The scheduler is called every minute, when turned on and distributes the pods according to the energy measurements.

Table 5.2: Description Use Case 2

ID	**UC3**
Name	Node does not have enough energy
Actor	System
Description	When a node which is hosting one or multiple pods, runs out of energy the pods need to be redistributed onto nodes with enough energy.
Precondition	There are nodes that can host pods.
Standard process	1. The energy measurements get processed. 2. One node does not have enough energy. 3. The pod on the node gets deactivated. 4. The pod gets deployed on a different node.
Alternativ process	There are no nodes with enough energy to deploy the pods.
Postcondition	The correct amount of pods are running.
Result	The pods are correctly redistributed onto other nodes.

Table 5.3: Description Use Case 3

Frontend

The frontend is where the user interacts with the application.

ID	**UC4**
Name	Choose amount of pods that are supposed to run.
Actor	User
Description	The user can specify the number of pods that are supposed to run on the nodes directly on the website.
Precondition	There are pods to run on the nodes.
Standard process	1. The user opens the website 2. The user specifies the number of pods they want to run by typing in a number into a textbox.
Alternativ process	The input is invalid. The program throws an error.
Postcondition	The program is processed further.
Result	The user cannot specify the number of pods directly, however they can move the pods to the nodes that they want them to run on.

Table 5.4: Description Use Case 4

ID	**UC5**
Name	Check location of the pods
Actor	User
Description	The user can check which node currently runs which pods by looking at the diagrams on the website.
Precondition	There are nodes that can run pods.
Standard process	1. The user opens the website. 2. The containers that run on each node are illustrated.
Alternativ process	There are currently no pods running.
Postcondition	The user can see which pods are running on which nodes.
Result	The Graphical User Interface (GUI) shows which pods are running on which nodes. Additionally the pods can be filtered according to their namespace.

Table 5.5: Description Use Case 5

ID	**UC6 (optional)**
Name	Drag&Drop pods
Actor	User
Description	On the website the user can drag one illustrated pod and drop it on another node.
Precondition	There are nodes that run pods.
Standard process	1. User drags a pod from a node. 2. User drops the pod on a different node.
Alternativ process	There is an error. An error message is thrown. The pod continues running on the old node.
Postcondition	The pod is deployed on the new node and shut off on the old node.
Result	The pods can be dragged and dropped between the nodes and are moved accordingly.

Table 5.6: Description Use Case 6

ID	**UC7 (optional)**
Name	Change actively running nodes
Actor	User
Description	Using the website, the user can change the amount of currently actively running nodes manually.
Precondition	There are nodes that run pods.
Standard process	1. User opens website. 2. User activates/deactivates certain node.
Alternativ process	There is an error. An error message is thrown.
Postcondition	Node gets activated/deactivated.
Result	Not implemented due to time restraints.

Table 5.7: Description Use Case 7

ID	**UC8 (optional)**
Name	Historical Data
Actor	System
Description	Historical Data is available in the web-GUI for analysis and optimization
Precondition	Energy information is available
Standard process	1. System receives energy information 2. System stores energy information in database 3. System displays historical energy information in web-GUI
Alternativ process	There is an error. An error message is thrown.
Postcondition	Historical energy information is available/not available.
Result	Not implemented due to time restraints.

Table 5.8: Description Use Case 8

5.2 Non-Functional Requirements

The non-functional requirements focus on the design and responsiveness of the GUI, performance and code quality.

ID	**NFR1**
Description	The Frontend is usable from desktop-devices with a 13” screen or larger
ISO/IEC 25010 requirement	Compatibility - Interoperability
Priority	Medium
Measurement	Manuel tests will be conducted using different desktop-devices and different screen sizes
Testing	At the end
Result	The frontend is usable and looks good on 13” or larger screens.

Table 5.9: Description Non-Functional Requirement 1

ID	**NFR2**
Description	The Frontend is responsive enough to detect changes in energy level and generation within five minutes.
ISO/IEC 25010 requirement	Performance Efficiency - Time Behaviour
Priority	Medium
Measurement	Manual test will be conducted.
Testing	At the end
Result	If the automatic scheduling is enabled the frontend checks the energy levels every minute.

Table 5.10: Description Non-Functional Requirement 2

ID	**NFR3**
Description	The Frontend is created with the goal of simplicity and to be self-explanatory.
ISO/IEC 25010 requirement	Usability - Operability
Priority	Medium
Measurement	The final GUI will be tested by different people and the feedback will be implemented accordingly.
Testing	At the end
Result	A usability test was conducted with positive results. More details can be found in the Usability Test section.

Table 5.11: Description Non-Functional Requirement 3

ID	**NFR4**
Description	Failed operations will be logged to help debug any possible errors.
ISO/IEC 25010 requirement	Maintainability - Analyzability
Priority	High
Measurement	Errors are manually checked in the log file.
Testing	At the end
Result	Failed operations in the backend are logged into a log file.

Table 5.12: Description Non-Functional Requirement 4

ID	**NFR5**
Description	Possible errors will be caught and presented to the user in a usable manner to inform what happened without the crash of the program.
ISO/IEC 25010 requirement	Reliability - Fault tolerace
Priority	High
Measurement	Manual tests will be conducted to ensure errors are caught appropriately.
Testing	At the end
Result	Errors and success messages are shown to the user with toasts.

Table 5.13: Description Non-Functional Requirement 5

ID	**NFR6**
Description	The code quality is according to rules expected by the linter.
ISO/IEC 25010 requirement	Maintainability - Analyzability
Priority	High
Measurement	No errors generated by the linter.
Testing	Periodically
Result	There a no linter errors, apart from errors concerning docstrings in the backend, which we decided to ignore.

Table 5.14: Description Non-Functional Requirement 6

ID	NFR7
Description	The frontend can be accessed using the browsers: Microsoft Edge, Google Chrome and Mozilla Firefox.
ISO/IEC 25010 requirement	Compatibility - Interoperability
Priority	High
Measurement	No console errors are generated by the Browsers
Testing	At the end
Result	The frontend works on all required browsers.

Table 5.15: Description Non-Functional Requirement 7

5.3 User interface design

Part of the project is a user interface that visualizes the entire process. Before the user interface is designed and developed, it is important to define key considerations to ensure its functionality, accessibility, and relevance to the target audience.

5.3.1 Accessibility and Platform Independence

The user interface should be easily accessible via a website. This makes it platform-independent and quickly accessible. Additionally it allows for easy adjustments and expansion of features.

5.3.2 Target Audience

The user interface should offer added value not only for computer scientists but also for people interested in computer science at professional, educational or other types of events. It should present the project in a simple way and make the process clearly visible. Furthermore it should be interactive so interested people can engage with the technology directly.

5.3.3 Core Functionalities

Apart from showing how the project works, the interface should also be functional. It should be possible to move pods between the different nodes and start the scheduling process among other things.

5.3.4 Mockup

As described in the non-functional requirements, the frontend is to be kept deliberately simplistic.

There are four different tabs planned, which separate the content and functionality of the website.

Overview page

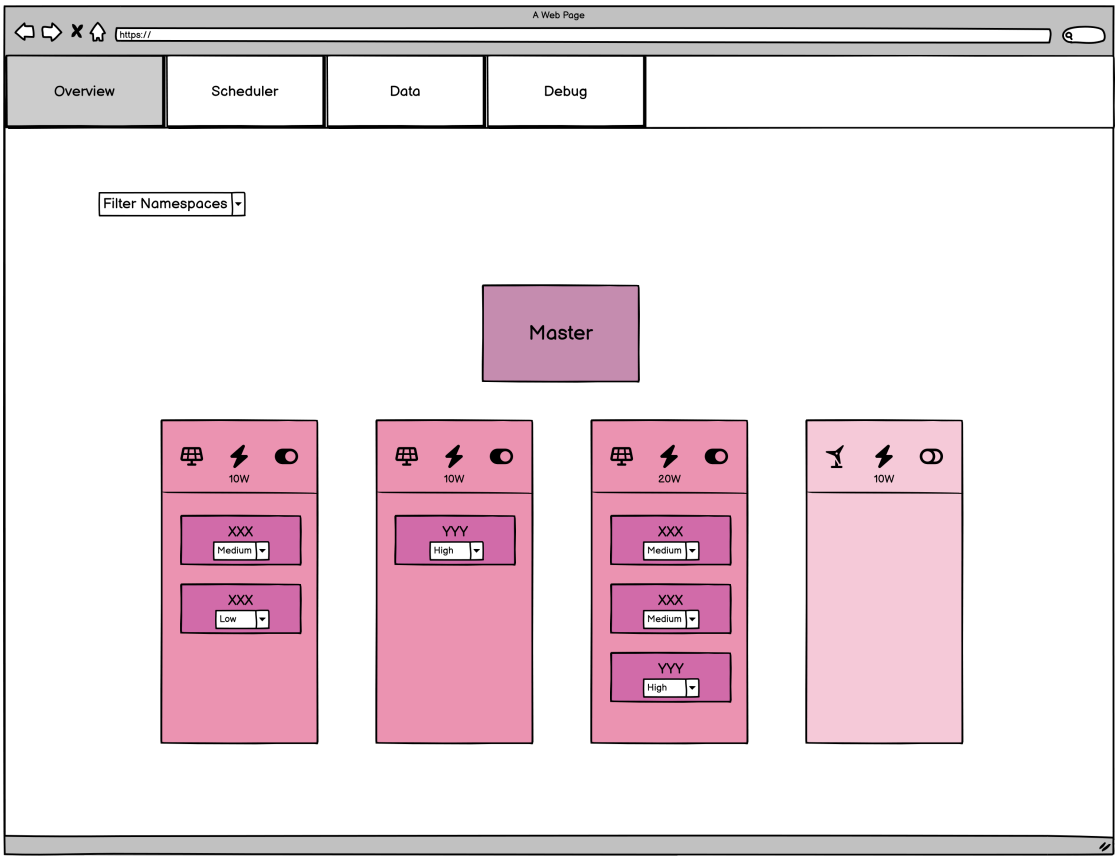

Figure 5.2: Mockup of the overview page

The main part of the GUI is the illustration of the different nodes and the currently running pods. Each worker node shows, where its energy comes from (solar, wind) and how much energy is currently stored. Additionally there is a toggle-button to activate or deactivate the node, this toggle will only be available if the optional use case 7 has been implemented.

At the top the user can choose to filter the currently shown pods according to their namespace.

Scheduler page

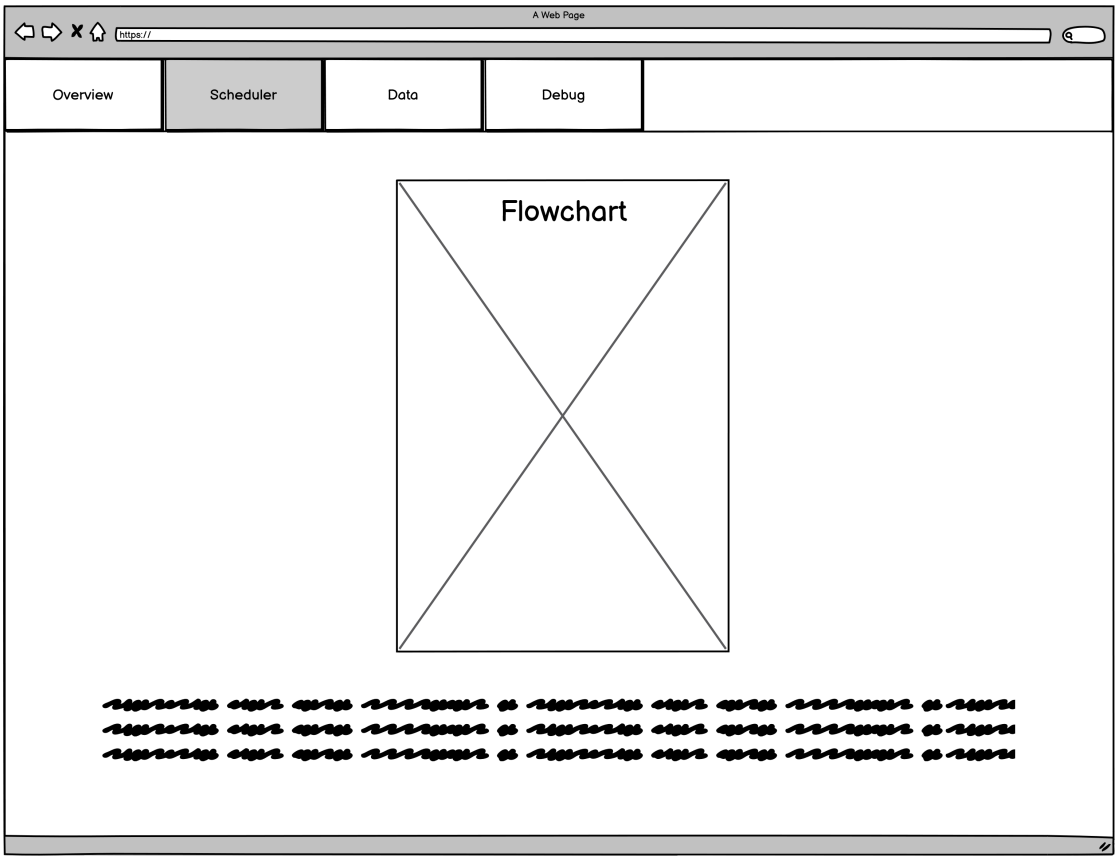

Figure 5.3: Mockup of the scheduler page

The scheduler page shows the flowchart of the scheduling process and explains it in a short text. This page can be used as an aid when explaining the process of the application at events.

Data page

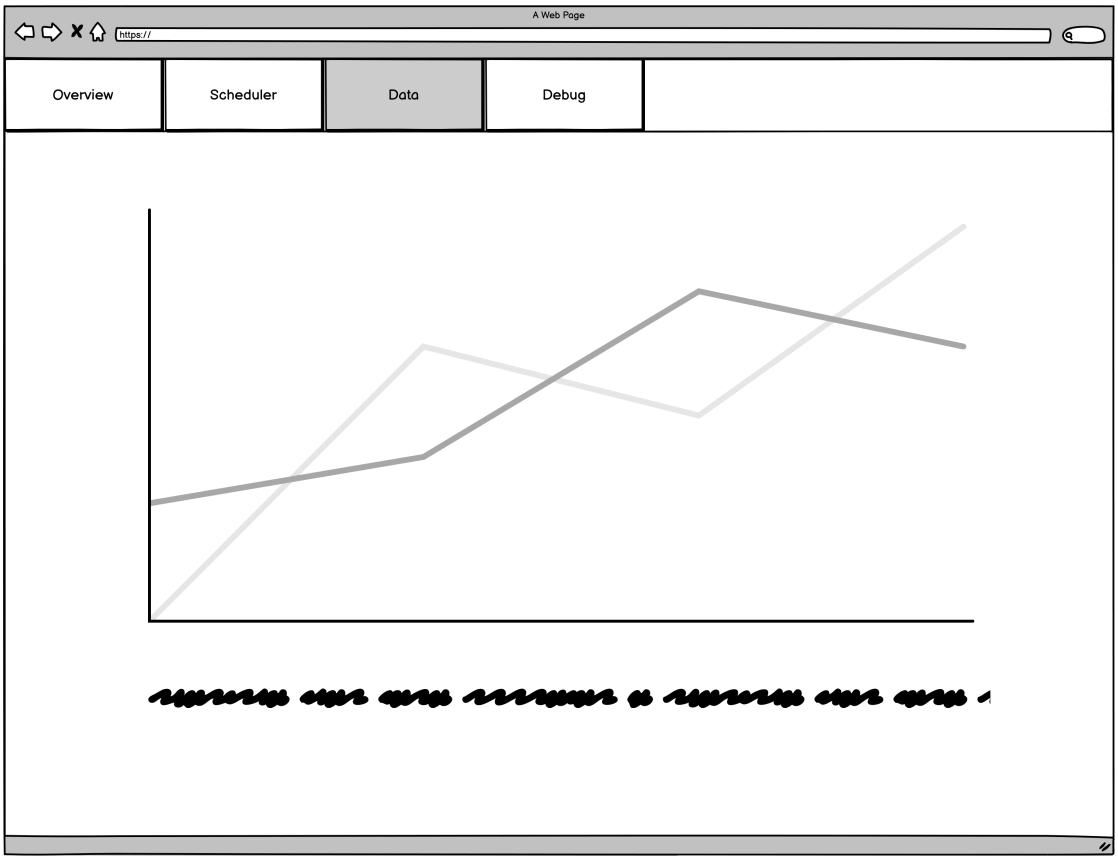

Figure 5.4: Mockup of the data page

The data page illustrates the scheduling process in a graph. It will show the energy information of the different nodes throughout the day. This page will only be available if the optional use case 8 has been implemented.

Debug page

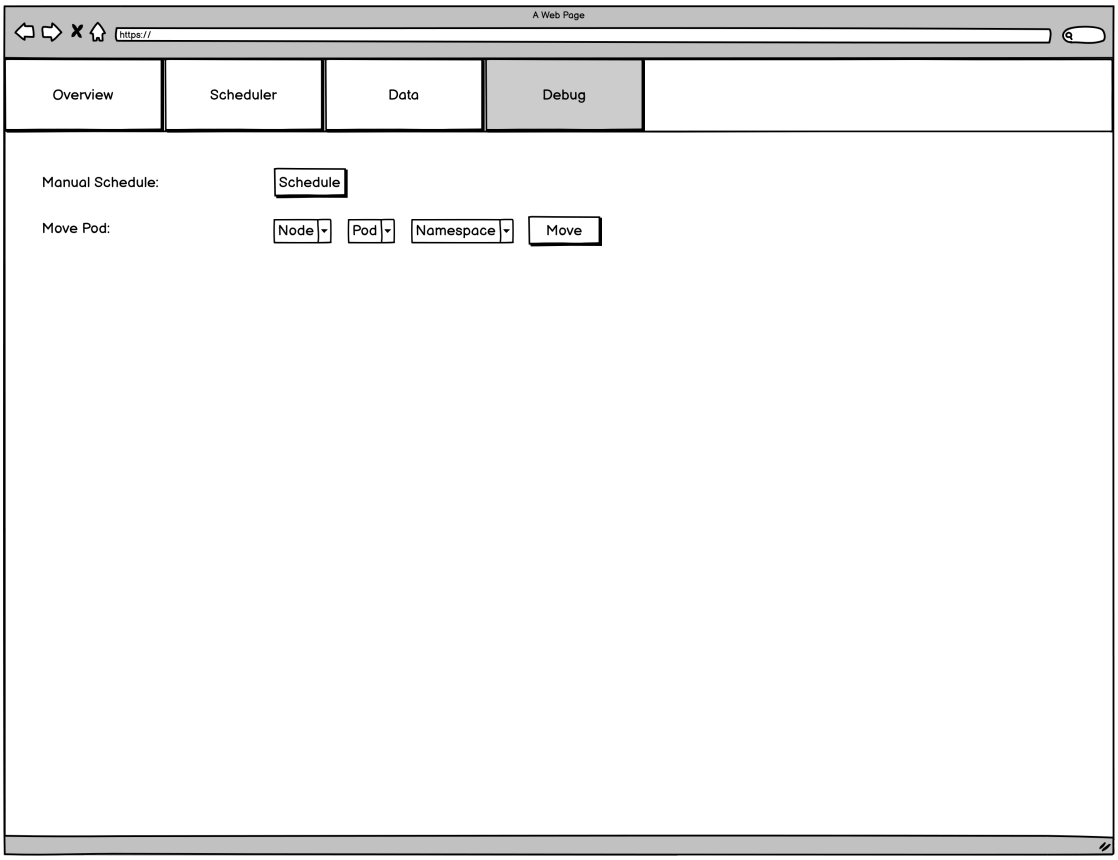

Figure 5.5: Mockup of the debug page

The debug page gives the option to start the scheduling process manually or to move pods around without using drag and drop. More functionality might be implemented here if needed in the future. The page is designed to give potential presenters the option to show certain processes, for example the scheduling process, at a convenient time instead of waiting for it to happen automatically.

Chapter 6

Architecture

6.1 C4 Architecture

The C4 architecture model was chosen because it is sufficient for our project, as there are no complex architectural features. Thus, the C4 architecture is particularly suitable for visualizing our software architecture.

The C4 model consists of four parts explained in the following chapters:

- System context diagram
- Container diagram
- Component diagram
- Code (omitted)

6.1.1 System context diagram

The System context diagram shows the overall picture of the project. The details are not important yet. It focuses on the user and the software system, not on technologies or protocols. [4]

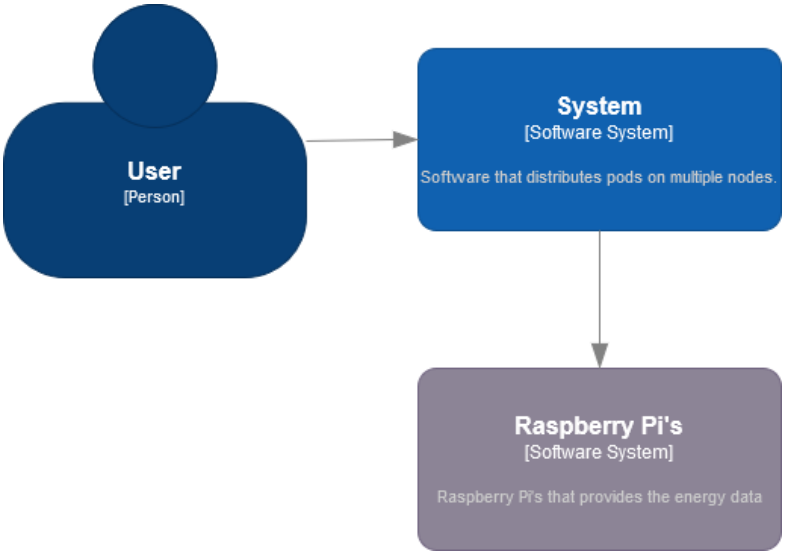

Figure 6.1: System context diagram

A user can interact with our system. Our system interacts with a different existing system on the Raspberry Pi's which provides the energy data.

6.1.2 Container diagram

The container diagram shows the structure of the software architecture at a high level and the distribution of responsibilities. A container is a separately executable / deployable unit that executes code or stores data. The diagram shows the high-level shape of the software architecture. It also focuses on the technologies chosen for the software.[2]

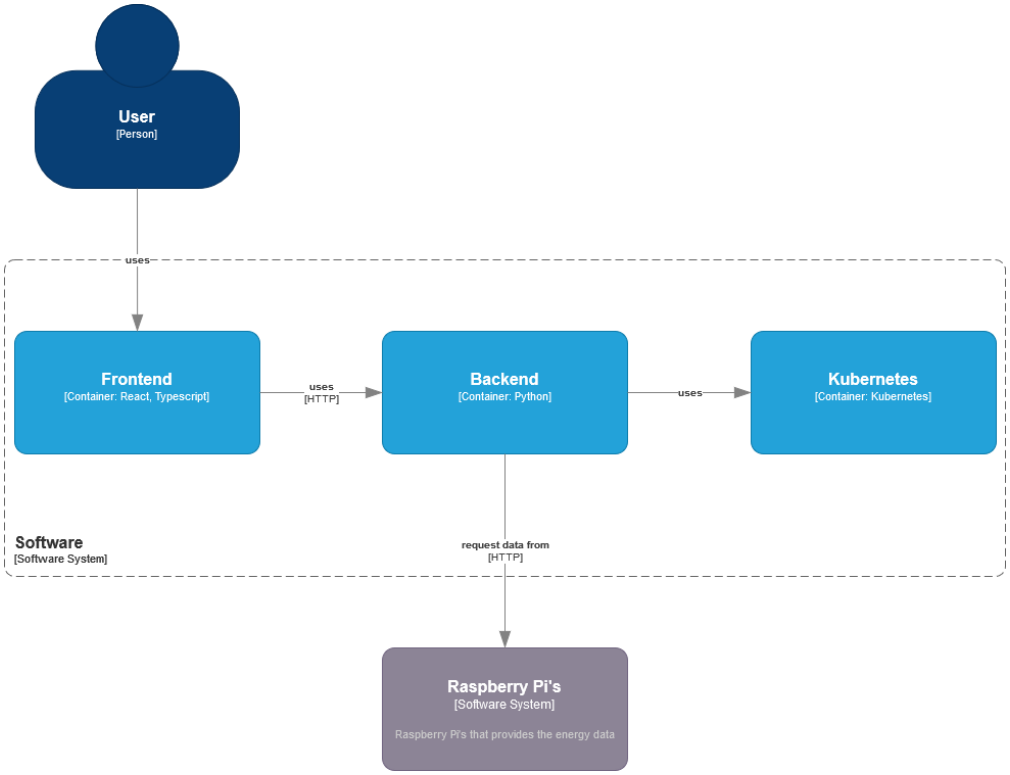

Figure 6.2: Container diagram

By dividing the project into frontend, backend, and Kubernetes cluster, the principle of separation of concerns is followed, helping to maintain clarity and organization within the project. The division also enables us to scale each component separately. The independent nature of our project structure brings the additional benefit of allowing us to test, develop and maintain part of the project separately.

Additionally there is the mentioned external resource. The energy measurements are provided to our system via HTTP requests to a Flask webserver.

The frontend is built using React with TypeScript, based on positive experiences with TypeScript in a previous project. React is chosen to facilitate the development of a clean frontend application. For the design, React Bootstrap and PrimeReact are used.

For the backend, Python was chosen with the Flask framework, as both developers have prior experience with Python. Additionally, there are many useful resources available on how to use Kubernetes with Python, which should aid in the development process.

6.1.3 Component diagram

The component diagram shows how a container is composed. It also explains what its states are and the details of the technology. [1]

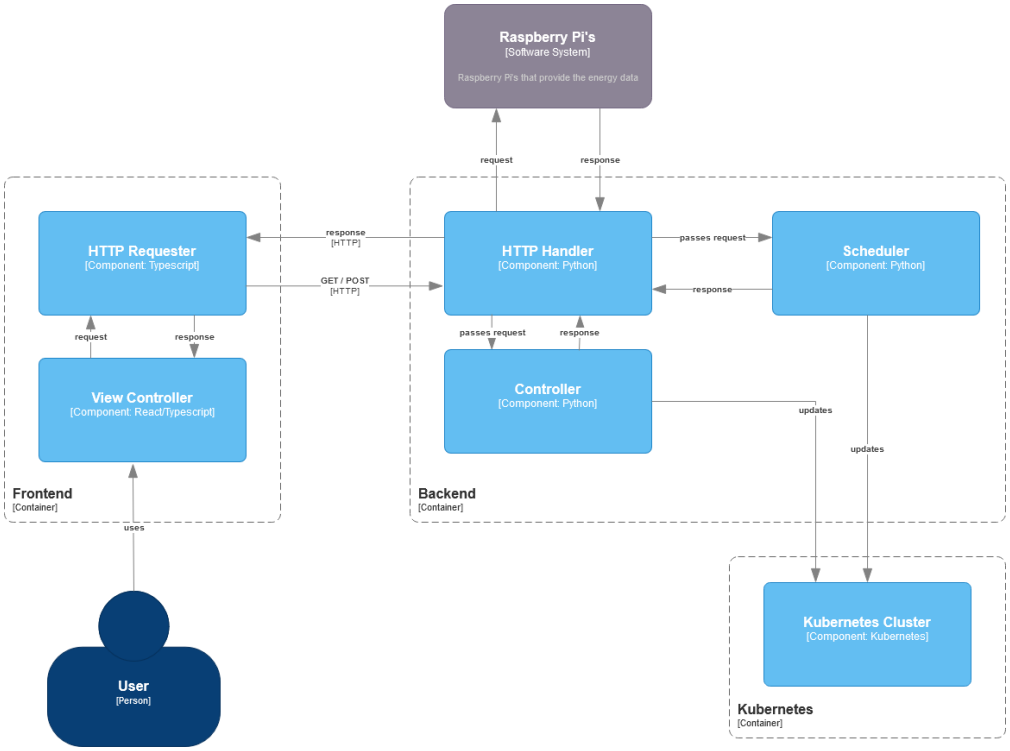

Figure 6.3: Component diagram

The user interacts with the View Controller over the website. The View Controller encompasses all of the pages and React Components used on those pages. The View Controller passes user interactions to the HTTP Requester, which sends GET / POST request to the backend. There the HTTP Handler handles these request and passes them either to the Controller or to the Scheduler. The Controller handles basic interactions with the Kubernetes-Cluster. The Scheduler handles the entire scheduling process.

6.2 Deployment

The entire application is deployed on a Kubernetes cluster. The cluster itself runs on three Raspberry Pi 4s, providing a lightweight and cost-effective solution for hosting the application. The Kubernetes distribution used is K3s, chosen for its simplicity and suitability for resource-constrained environments. [3]

Flannel serves as the Container Network Interface (CNI), enabling seamless communication between pods across the nodes in the cluster. To expose the application to external traffic, a Traefik ingress controller is utilized, which also handles load balancing and routing traffic to the appropriate services.

The deployment process ensures that the application is highly available and capable

of scaling horizontally to meet demand. All components are containerized and orchestrated within the cluster, allowing for efficient resource utilization and streamlined updates.

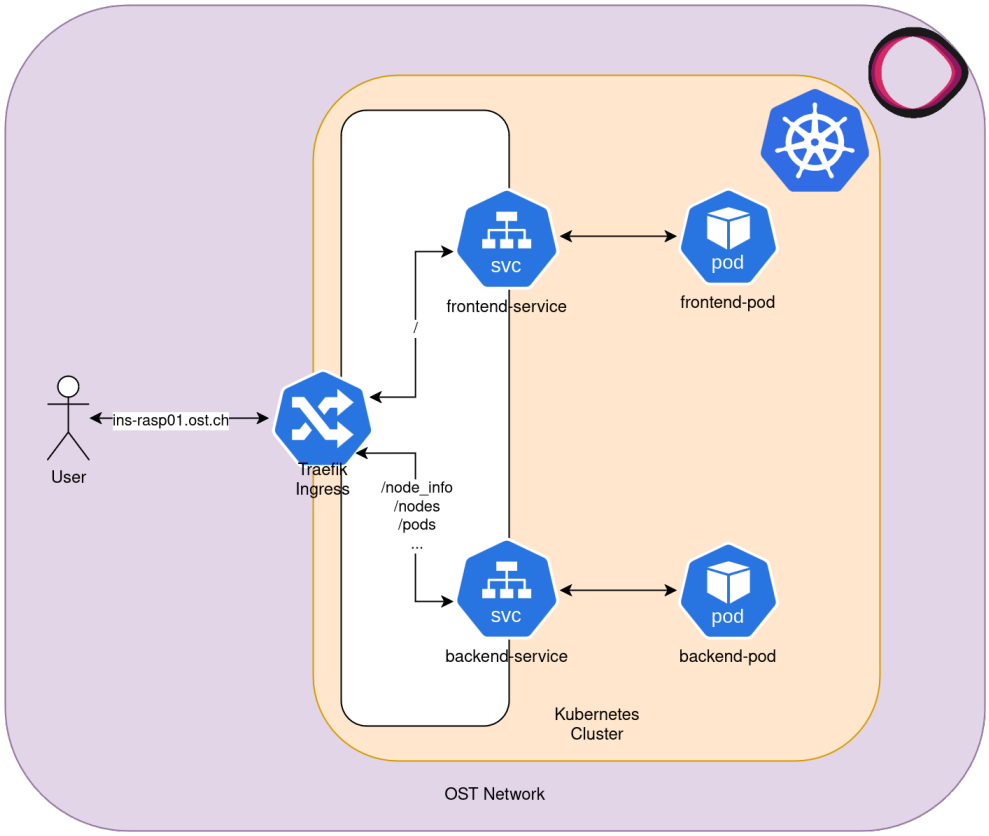

Figure 6.4: Deployment diagram

Chapter 7

Implementation Details

7.1 Test Cluster

A test Kubernetes cluster was set up using the hardware provided, which consisted of Raspberry Pis. Given that K3s is specifically designed for production workloads on resource-constrained devices, it was chosen as the Kubernetes distribution. During the setup, it was realized that the Raspberry Pi 3s did not have enough RAM to build the cluster. After extensive troubleshooting, a Dell Mini PC was used as the master node, which worked perfectly. [3]

At the beginning of the test cluster setup, Cilium was selected as the CNI, as Philipp already had some knowledge about it, and it was also recommended by our Technical Advisor. But as the troubleshooting continued on, one step of it was the removal of the custom Cilium CNI for the default Flannel CNI. In the end, this was not a major issue, as the advanced features of Cilium, such as Network Policies, were not required during the project.

The Test Cluster was setup at Philipps home, and shared using a Tailscale VPN. This solution allowed remote access to the cluster, enabling both of us to connect and test newly developed features easily. The setup with the Tailscale VPN was very straight forward and intuitive.

7.2 GitLab CI/CD Pipeline

A GitLab CI/CD pipeline was created to automate the Docker image build process. The goal was simple: every time a new Git commit is made, the CI/CD pipeline automatically creates a new Docker image for deploying the application. There was a problem however, the application needed to run on low-powered Raspberry Pi's, which are based on the ARM64 processor architecture. This required modifications to the initially planned pipeline. After spending a lot of time adjusting it, the pipeline still would not work.

Following extensive troubleshooting, the decision was made to reach out to the technical advisor for assistance. He shared a pipeline that the INS currently uses in production. Upon reviewing it, the solution became clear: the wrong Docker base image had been used for the pipeline. Using the correct image, the pipeline worked flawlessly.

7.3 Move pods

For the success of the project, the move functionality is of utmost importance. Without the ability to move the pods, adjusting the scheduling based on energy measurements would not be possible. While implementing this functionality, two different strategies were tested.

7.3.1 Standard move

It is important to note that a pod cannot be directly moved from one node to another in Kubernetes. Instead, it must be destroyed and then redeployed. Initially, this was the approach. However, it quickly led to an issue. Every time a pod was destroyed, Kubernetes redeployed it immediately, usually on the wrong node. Kubernetes tracks the number of pods that should be running for each deployment and ensures that the desired count is maintained, which caused the problem. As a result, a different strategy had to be developed.

7.3.2 Move with Deployments

Through further research and advice from the advisors it was decided to use deployments to our advantage. All moveable pods are part of a deployment. The deployments are assigned a node affinity during the moving-process, with the help of which the pods are scheduled on the correct node. All nodes are assigned a label with their name. During the move process the node affinity is used to specify that the pods of a certain deployment can only be deployed on nodes with the specified name. Once the node affinity is updated, Kubernetes will destroy the pod and redeploy in on the correct node automatically.

7.3.3 Move on the frontend

As the application needs to be interactive, so it can aid to get people interested in the subject, having a drag and drop functionality to manually move the pods gives a interesting hands-on experience. The drag and drop functionality was implemented using React dnd kit. This made it simple to assign the pods on the overview as Draggable-Containers and the nodes as Droppable-Containers. A request is send to the backend with the pod and the node it has been moved to and the moving-process starts as described above.

7.4 Scheduler

As the worker nodes are not directly connected to the power grid but are powered by solar or wind energy, it can happen that they have little or no energy. In this case, a strategy for handling this scenario is defined.

The pods that are deployed on the nodes are assigned a priority level: High, Medium or Low. This priority level is used to decide which pods are switched off first when the energy is low. By default, a pod has the priority level Medium.

If a worker node only has a battery power of less than or equal to 10%, the pods running on this worker are allocated to the other workers. If the other workers have no energy for all these pods, the priority level is used to decide which pods are redeployed and which pods are switched off.

If only one worker node is still running, the focus is placed on the high-priority pods. If the last worker has less than 40% battery power, all but the high-priority pods are switched off.

If the last worker node also has less than 10% power, all pods are switched off.

All mentioned thresholds are modifiable on the website.

The graphic below illustrates this scheduler logic.

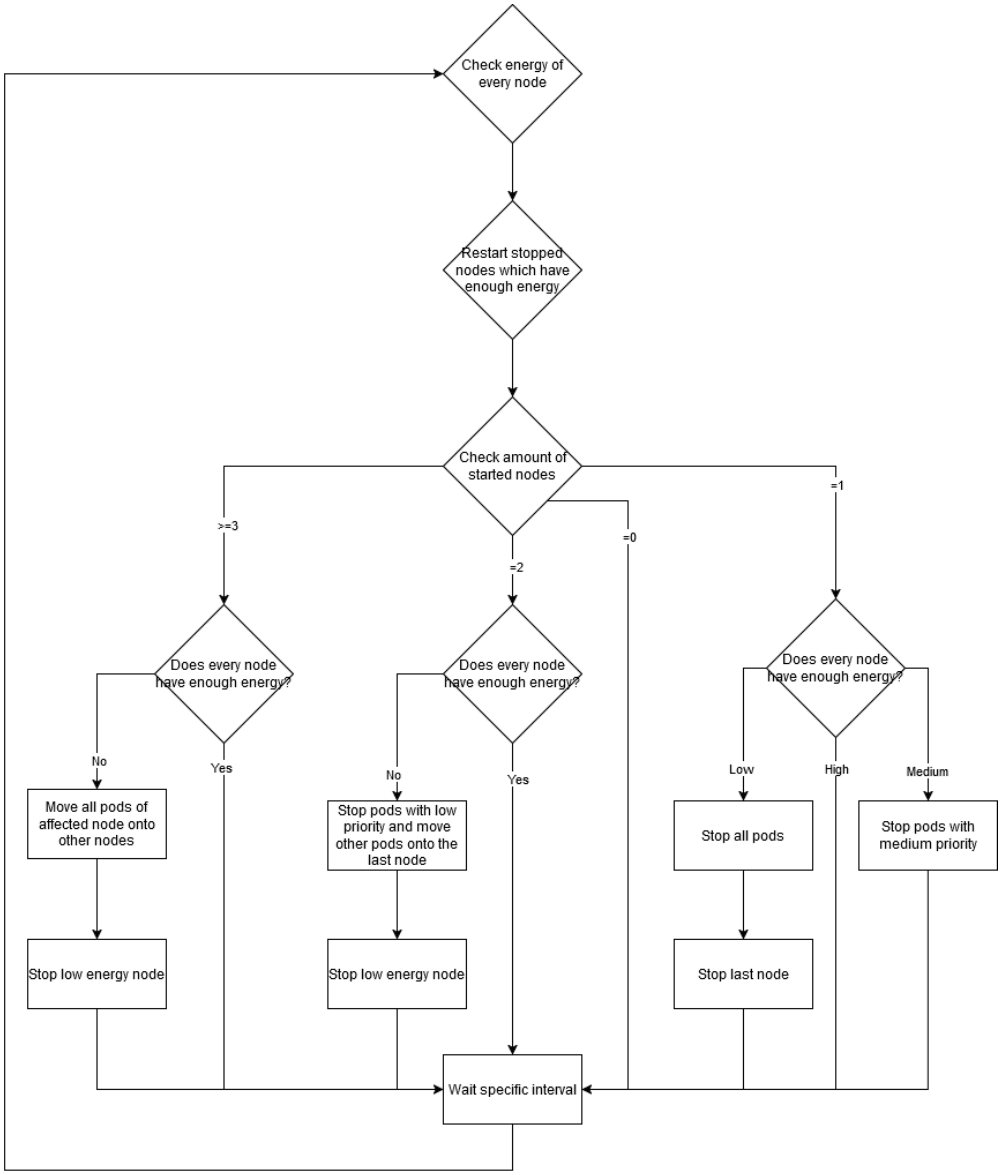

Figure 7.1: Scheduling diagram

7.5 Dockerization

In order to be able to deploy both pieces of the application, the frontend and the backend. It needs to be containerized. This was done using the CI/CD pipeline mentioned above. The frontend is built using a multi-stage build, which first builds the application and then copies the build into a nginx container. The backend is built using a simple Dockerfile, which copies the source code into a Python container. To efficiently deploy

the application, environment variables were used. This way the backend can be configured to use the correct Kubernetes Config and the frontend can be configured to use the correct backend. Challenges arose when the frontend was built with the environment variables, as the React App was not able to access them. The solution chosen was the contrary to the best-practice of using a .env-file. A simple script was added to the frontend, which replaces the placeholders with the correct value during the first run. This approach was chosen because the environment variables are needed to be set at runtime, whereas React sets them at build time.

7.6 Read energy

The external system to read the energy data was only available at the very end of the project. However it was still implemented.

The script was provided by the project advisor, and no changes were made to it. In essence the script accomplishes three things:

- Reads the energy data from a serial device.
- Stores battery voltage and charging current in a dictionary.
- Serves the data using a Flask webserver.

Due to the very important nature of this script, certain steps were taken to ensure its reliability. The script is run as a service on the Raspberry Pi, which means it is started automatically when the Raspberry Pi boots up. This ensures that the energy data is always available.

Unfortunately, reading the energy from the Raspberry Pi takes approximately one minute. The original plan was to retrieve the energy data every time it was needed, whether for scheduling or displaying it on the website. However, the one-minute delay proved to be too long for practical use. As a result, and due to time constraints, two versions of the project were created: The main version, which uses randomly generated energy values for faster performance, and a secondary version that retrieves actual energy data, but is slower. Possible improvements to address this issue are discussed in the Outlook section.

7.7 Design

As the project progressed, it became clear that the design of the GUI would be just as important as its functionality. This led to a reevaluation of the initial approach. After some brainstorming, the decision was made to use the metaphor of shipping containers, which Docker and Kubernetes already use, and design the GUI around that concept.

The overview was designed in a way to make the nodes look like container ships and the

pods would be containers on said ships. The containers are colored differently depending on their namespace. To keep the application dynamic, each namespace could not be simply assigned a different color, as new namespaces could be added in the future. The problem was solved by generating a hash from the namespace and using this hash to create different RGB values. This way every new namespace will get a unique color and the colors will persist over page reload without having to be saved into a database.

To keep with the metaphor the master node represents the port were everything gets coordinated.

7.7.1 Initial Approach

Initially, all elements of the GUI were intended to be created with CSS. This way the design would be very flexible and quickly adjustable. This however, proofed to be a difficult task. Creating the front of the container ship already lead to problems. The goal was to make sure to get the shape of it right, however using only CSS, a satisfying result could not be achieved. In order to get the desired result, Figma, which is a design tool, was used to create the more complex shapes of the design.

7.7.2 Hybrid approach

Using this hybrid approach of Figma and CSS, the container ship and port was created.

Container ship

After familiarizing ourselves with Figma, the front and back parts of the container ship were created first.

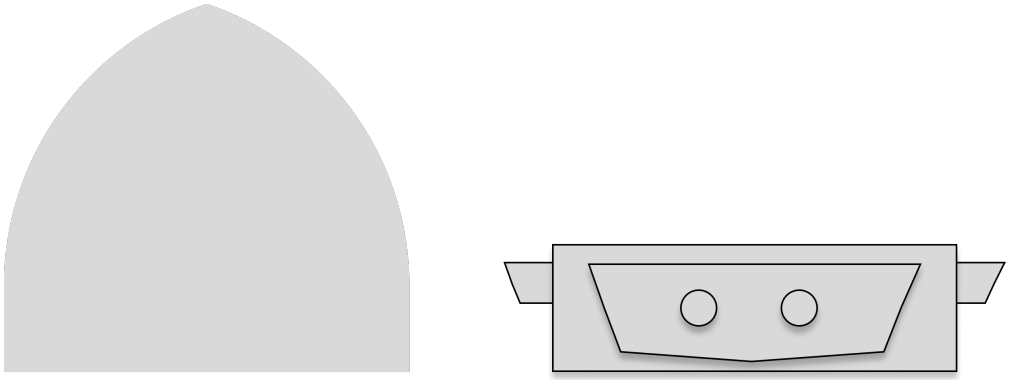

Figure 7.2: Ship parts created with Figma

The space between these two parts is created using regular HTML Elements and CSS, this way the size of the ship can still be flexible.

Containers

The port was to be filled with containers. For this different colored boxes were created and detailed so they would look like shipping containers.

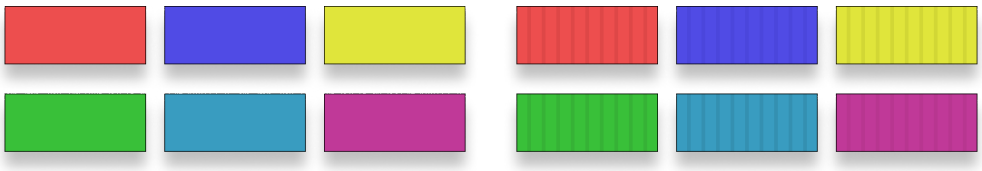

Figure 7.3: Containers for the port

Crane

In order for the port to represent the master node, it has to visually communicate that it is responsible for the distribution of the pods (containers) on the nodes (container ships). To achieve this cranes were illustrated and placed onto the port.

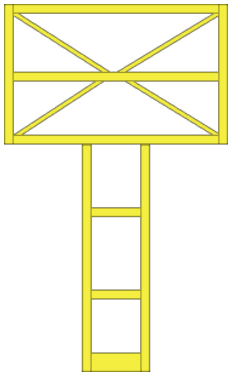

Figure 7.4: Crane for the port

Vehicles

To add more visual interest, two vehicles were made for the port. A forklift and a truck with a container.

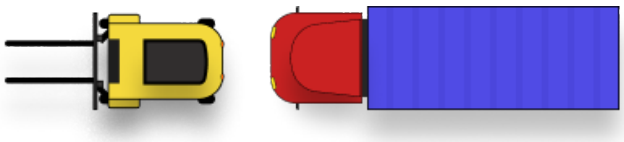

Figure 7.5: Vehicles for the port

Port

Using the mentioned elements the port was put together. Some of the containers are stacked on top of each other like they would be at a real cargo port. Additionally tracks for the cranes were added.

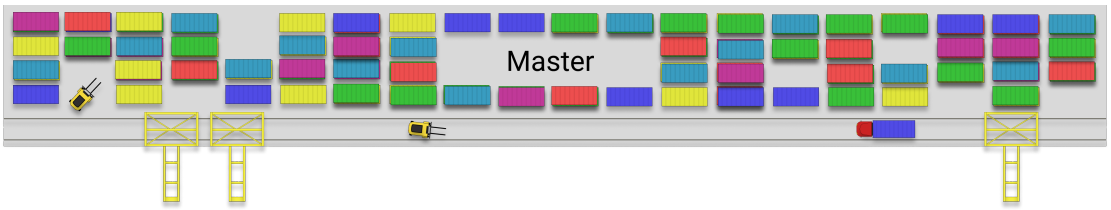

Figure 7.6: Port representing the master node

7.7.3 Result

Putting all the elements together the following result was achieved. At the top left is the timer until the next automatic schedule with a start/stop toggle. In the middle is the navigation and the filter for the different namespace. On the right an example pod, which serves as a legend. The main part of the page is the node overview.

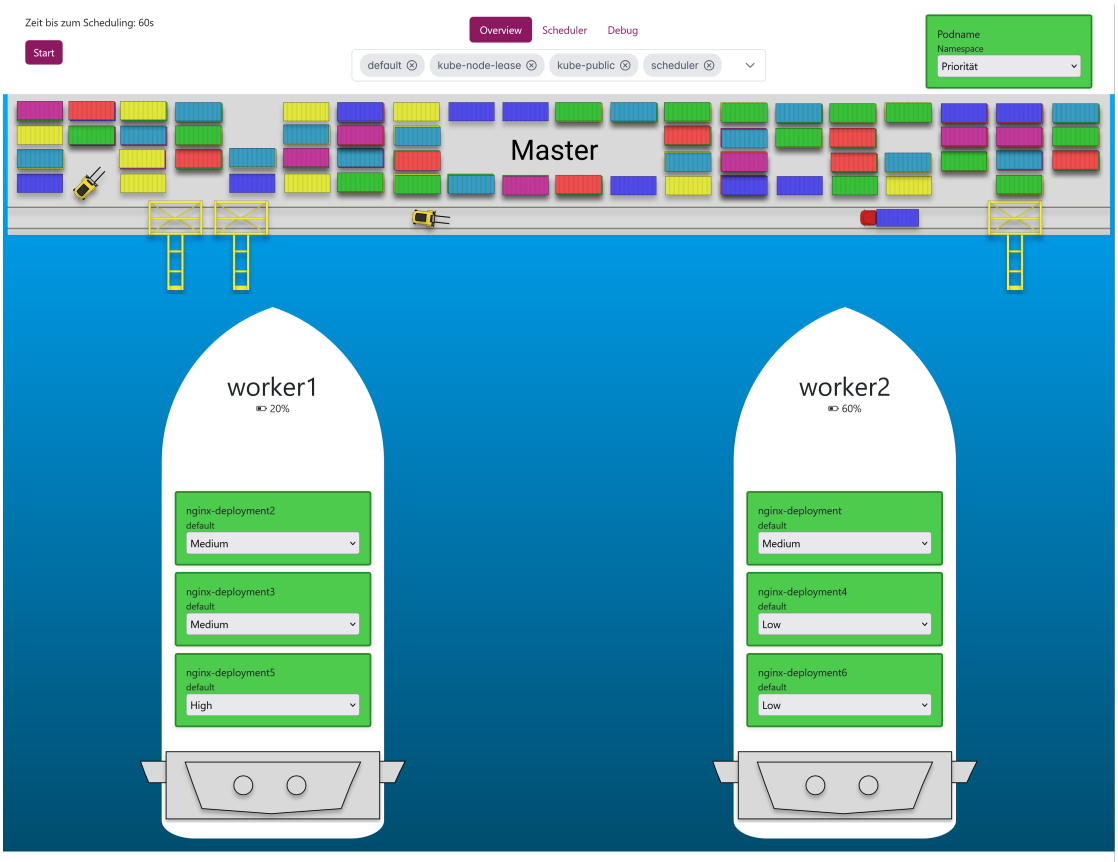

Figure 7.7: Overview page

Chapter 8

Results

In this chapter the achievements and shortcomings are laid out. The results are organized according to objectives defined in the requirements chapter.

8.1 Achievements

The achievements are split into two parts. First the functional requirements are discussed and then the non-functional requirements.

8.1.1 Functional requirements

The main functional requirements and an optional one were achieved.

Use cases UC2 and UC3, which involve the scheduler, could be implemented successfully. The scheduler can be turned on or off on the website. When turned on, the scheduler runs every minute and distributes the pods according to the energy measurements when necessary.

Use case 4 is about choosing the amount of pods that should be running on a node. Originally it was planned that the user can input the amount of pods that they want to run on a node into an input-field, this behaviour was not implemented however. Instead the user can drag&drop the pods onto the different nodes like described in the optional use case 6. This also ties into use case 5, the user can see all nodes and pods in the overview.

8.1.2 Non-functional requirements

All non-functional requirements could be achieved. For NFR3 the original goal was to create a frontend which is simple and self-explanatory. During development is became clear that the frontend needed more eye-catching elements as described in the design-

implementation. However the goal of it being self-explanatory as proven by the Usability-Test was still achieved.

8.2 Shortcomings

The shortcoming will also be split into functional requirements and non-functional requirements.

8.2.1 Functional requirements

The first functional requirement UC1 could not be fully implemented. While the energy data could be accessed, the process took too much time. Since access to the energy measurements was granted late in the project, there was not enough time to resolve these issues.

Additionally the two optional use cases UC7 and UC8 were not implemented due to time restraints.

8.2.2 Non-functional requirement

As stated above, all non-functional requirements could be achieved.

Chapter 9

Conclusion and outlook

As we conclude this project, it is essential to reflect on its outcomes while also considering potential improvements and directions for future work.

9.1 Conclusion

This thesis aimed to extend a scheduler so that it also takes into account the availability of electricity from renewable sources. Additionally the system should also be presented in a understandable way to people interested in a computer science degree.

As mentioned in the results chapter we could achieve most of our goals. We have a working scheduler and a good looking and functional website.

The scheduler works well and is, apart from the slow energy readings, very responsive. The priority labels give the users control over the scheduler which is useful as the energy generation of a solar panel can fluctuate.

The website adds the playful element we sought out to implement and should help make the complex topics of Kubernetes and container orchestration easier to understand.

9.2 Outlook

In the future the project could be improved and expanded upon in the following ways:

9.2.1 Cache energy data

As discussed in the results chapter, reading the energy data from the nodes is currently a time-consuming process. In the future, this could be improved by implementing a system that periodically collects and caches the data in the background. This approach would reduce the delay when accessing the data.

9.2.2 More complex scheduler

The scheduler could be enhanced in several ways. One potential improvement would be to incorporate weather forecasts into the scheduling process. Nodes connected to solar panels, which would generate less energy during poor weather conditions, could be given lower priority in the scheduling. In addition to weather data, other factors such as historical energy production and geographical data could also be considered to further optimize the scheduling process.

Part II

Project and Time Management

Chapter 10

Project Plan

10.1 Collaboration

Kanban is used as our project management methodology. As there are only two of us working on this project, it is not necessary to use SCRUM. The Kanban board and the weekly meetings are enough to keep everybody up to date. Daily stand-ups are therefore not necessary and the time can be used for other purposes.

10.2 Project Roles

Due to our limited group size it does not make sense to create elaborate roles for this project. Nonetheless these two roles were defined:

Role	**Person**	**Area of responsibility**
Lead Developer Frontend	Marco Schnider	Creation of a Web Frontend using React
Lead Systems Engineer	Philipp Hutter	Initial Kubernetes Setup

Table 10.1: Project Roles

For every other task the team members have to work together and responsibilities are assigned when needed.

10.3 Minimum Viable Product (MVP)

The Minimum Viable Product (MVP) is a version of a product that has just enough features to satisfy early customers and gather feedback for future product development. It is a strategy commonly used in software development and startups to quickly test and

validate a product idea before investing significant time and resources. In our case, the MVP will be equivalent to the initial release, covering all essential, non-optional features of our application.

MVP Date: 24.11.2024 (this was achieved)

10.4 Long-term plan

In this chapter, the long-term plan was outlined at the beginning of the project. First, the total time available for the project was listed in weeks. Then, the features and goals of the project were identified, and rough deadlines were set for their completion. Additionally, the Milestones were defined.

Phase	Week	Date	Goals	Milestones	Soll	Ist
Inception	1	16.09 - 22.09	Project Proposal, Plan	M1	22.09.2024	22.09.2024
Elaboration	2	23.09 - 29.09	Requirements		29.09.2024	29.09.2024
	3	30.09 - 06.10	Architecture	M2	06.10.2024	06.10.2024
	4	07.10 - 13.10	Mockups GUI		13.10.2024	13.10.2024
	5	14.10 - 20.10	F1.1) Prototype	M3	20.10.2024	20.10.2024
Construction	6	21.10 - 27.10	F1.7		27.10.2024	27.10.2024
	7	28.10 - 03.11	F1.3, F1.8		03.11.2024	03.11.2024
	8	04.11 - 10.11	F1.4, F2.3		10.11.2024	10.11.2024
	9	11.11 - 17.11	F1.5		17.11.2024	17.11.2024
	10	18.11 - 24.11	F1.6, F1.9	M4	24.11.2024	24.11.2024
	11	25.11 - 01.12	F2.1, F2.2		01.12.2024	02.11.2024
	12	02.12 - 08.12	F2.4, F2.5, F1.2		08.12.2024	16.12.2024
Transition	13	09.12 - 15.12	Testing, Abstract	M5	15.12.2024	16.12.2024
	14	16.12 - 20.12	Submission	M6	20.12.2024	20.12.2024

F = Feature

Figure 10.1: Container diagram

The features are the following:

- F1.1) Assign pods to nodes (manually)
- F1.2) Read energy data
- F1.3) Automatic allocation based on energy
- F1.4) Create GUI (with different Tabs)
- F1.5) Develop backend
- F1.6) Dockerize the application
- F1.7) Adjust scheduler
- F1.8) Error logging

The following additional (optional) features were defined:

- F2.1) Scalable Frontend (Frontend can work with a dynamic number of nodes)
- F2.2) Scalable Scheduler (Scheduler can work with a dynamic number of nodes)
- F2.3) GUI: Drag & Drop of pods onto nodes
- F2.4) GUI: Change the number of active nodes
- F2.5) Display historical data

As the long term plan shows most of the features could be finished in time. The only delay was caused by the feature F1.2, as we had to wait for the system to read the energy to be finished.

10.4.1 Milestones and goals

The following milestones will help us to gradually achieve our goals for this project. At the end of the project all milestones will have been reached.

M1: Set up project

- Set up basic project structure
  - Setting up GitLab
  - Configuring the time tracking
  - Configuring the time reporting
  - Configuring issue management
- Analyze and evaluate project risks and discuss measures

M2: Requirements and Architecture

- Define functional and non-functional requirements
- Define use cases
- Define project architecture
- Define tooling
- Specify long-term-plan

M3: Prototype

- Create initial backend functionality.
- Test plausibility of the project based on the prototype.
- Create test concept with the required unit tests, usability tests and system tests.

M4: MVP

- Basic MVP functions are implemented and users are ready to use the application.
- Initial user feedback is obtained.
- Identifying and fixing bugs or problems that were discovered during tests.
- Documentation: Create user manual, give reasons for the chosen technologies, logical and physical technologies, describe the logical and physical distribution of the components and demonstrate the expandability of the architecture.

M5: Production release

- Functional completeness: Ensure that all planned functions are implemented and work as expected.
- Final testing: Carry out a thorough testing of the software to ensure that it meets all functional and non-functional requirements.
- Bug fixing: Identifying and fixing any remaining bugs or problems in the software.
- Performance optimization: Optimizing the performance of the software to ensure that it runs smoothly and efficiently.
- Documentation: Finalizing the user manual based on feedback.

M6: Final delivery

- Deliver a high-quality software product that fulfills all requirements
- Complete all defined development tasks
- Thoroughly test all functions
- Resolve all problems and errors
- Document project comprehensively

10.5 Risk management

In the following tables the risks and their probability and severity are defined.

10.5.1 Risk determination

Probability / Severity	1 Very unlikely	2 Unlikely	3 Occasionally	4 Likely	5 Frequent
4-Catastrophic	R6				
3-Critical		R3	R7		
2-Significant			R2		
1-Low			R1, R4	R5	

Table 10.2: Risk Matrix

ID	**R1**
Risk	Learning curve
Comment	Possible learning curve for team members who are not familiar with the tools and technologies used.
Preventive measures	With the role allocation, the development leads are clearly defined. The corresponding leaders already have experience in all of the main technologies in use.
Corrective measures	The features are divided into as small tasks as possible. The individual leaders will support the team members with less experience.

Table 10.3: Description Risk 1

ID	**R2**
Risk	Scope Creep
Comment	Unclear definition of the requirements, which will lead to a continuous expansion of the project.
Preventive measures	In the meetings, it is precisely defined what the goals for the project are.
Corrective measures	Optional features can be scrapped.

Table 10.4: Description Risk 2

ID	**R3**
Risk	Limited resources
Comment	Absence of a team member.
Preventive measures	Both team members have to document on which functionality they are currently working on.
Corrective measures	The tasks that fall to the person concerned are redistributed in an impromptu meeting.

Table 10.5: Description Risk 3

ID	**R4**
Risk	Compatibility problems
Comment	Different versions of tools and technologies.
Preventive measures	The version of each technology used is documented, and all members must use this specific version for the duration of the project.
Corrective measures	For different versions, upgrades and downgrades are carried out.

Table 10.6: Description Risk 4

ID	**R5**
Risk	Time estimation
Comment	Too much or too little time will be allocated to certain tasks.
Preventive measures	The regular meetings enable us to recognize differences early on and initiate measures.
Corrective measures	Optional Features can be scrapped or implemented.

Table 10.7: Description Risk 5

ID	R6
Risk	Dependencies on third parties
Comment	Dependence on external libraries, APIs or services with potential risks in terms of access reliability, compatibility or support.
Preventive measures	Use as few external resources as possible. In addition, it is also possible to integrate external libraries into a separate class.
Corrective measures	If an external resource fails, in a few cases it can mean the end for our project.

Table 10.8: Description Risk 6

ID	R7
Risk	Problems with the master node
Comment	The Raspberry Pi, which is used as the master will not be powerful enough.
Preventive measures	Use lightweight OS and Kubernetes tools.
Corrective measures	If the master node cannot be made to work, it will be necessary to consider using a VM on a more powerful machine as the master node.

Table 10.9: Description Risk 7

10.5.2 Contingency plan

Not all risks can be recognized from the beginning. The contingency plan can be used if an unexpected risk occurs.

1. The problem will identified and discussed with the fellow team member.
2. If the problem cannot be fixed easily, it will be promoted to a standalone task and more time will be spend on it.
3. If the problem cannot be fixed it will be discussed with the project advisor.

The purpose of this contingency plan is to define a planned response in the event of incidents that affect the normal course of the project. Because the process has already been defined, the interruption within the project can be minimized through a targeted response.

10.6 Jira

To plan the project and assign tasks, Jira is used, with the main focus on short-term planning and the issue tracking. The project is divided into 4 task types:

- **Task** Default type that document the work that needs to be done.
- **Milestone** Special type that mark milestones in our project.
- **Meeting** Special type in which time spent on meetings is noted, as it cannot be directly assigned to another task.
- **Bug** Error that was found during testing and needs to be resolved.

At every weekly meeting the work that needed to be done to advance in our project according to our Long-Term-Plan was discussed.

Chapter 11

Quality Measures

11.1 Planning and Reviews

Through planning with Jira, the project is effectively implemented by assigning, tracking, and coordinating tasks to ensure a smooth process and successful completion. The meetings are used to discuss progress, identify challenges, and to develop solutions to ensure the goals of the project can be achieved with the desired quality.

11.2 Code Quality

How the quality of the code is ensured is defined in the coding setup. Namely through a linter, Prettier and Husky hooks. These tools were chosen because they are suitable for the project, and both team members have prior experience with them.

11.3 Branching and Peer-Review

The branching strategy is defined in the GitLab guidelines. Features and bug fixes are developed in dedicated branches that branch of from the develop branch. These branches are later merged back into the develop branch to ensure conflict-free integration. Once all tests have passed successfully and all issues have been resolved, the changes are merged into the main branch. This ensures that there is always a working version of the project on the main branch.

11.4 Test concept

The test concept describes what is tested, how it is tested, what tools are used, and the process of the tests. Additionally, responsible individuals are assigned to the tests, and the risks are assessed in case a test fails.

11.4.1 Process

The process is as follows:

- Clarify what needs to be tested based on the requirements.
- Decided how the tests will be conducted.
- Execute the tests.
- Document successes and errors.
- After the errors have been resolved, re-run the tests.

11.4.2 Goals

The following functionalities and elements are tested:

- All use-cases
- Compatibility
- Performance
- Usability
- Maintainability
- Reliability

The goals are taken from the use-cases and the non-functional-requirements.

11.4.3 Roles

There are two different roles:

- Test lead: Creates the test cases.
- Tester: Executes the tests and documents the findings.

11.4.4 Environment

The tests are executed in the following environment:

Client

- Operating System: Windows 11
- Browsers: Microsoft Edge, Google Chrome, Mozilla Firefox
- React version: 18.3.1
- Typescript: 4.9.5

Server

- Operating System: Raspberry Pi OS (Debian GNU/Linux 12)
- Flask version: 3.0.3
- Python version: 3.10

Kubernetes and Docker

- Kubernetes version: 1.30.6+k3s1
- Container Runtime: containerd://1.7.22-k3s1

11.4.5 Unit-Tests

Unit tests were initially planned for the project, but ultimately it was decided against creating them. The focus shifted to the GUI design, as discussed in the Design chapter. Given the need to set up mocks for the Kubernetes client to properly test most functions, time constraints became a challenge. Although the importance of automated testing is recognized, priorities had to be adjusted.

11.4.6 Schedule

The tests will be completed during the transition phase. For more information see the long-term plan.

ID	**T1**
Name	Functionality-Tests
Process	All functional and non-functional requirements of the application are to be tested. Every requirement will be tested one-by-one and checked if the results are as expected. This way the connection between the frontend, backend and Kubernetes Cluster is also tested.
Method	Manual
Risk	High
Relevant for	All goals
Test environment	Client and Server
Interval	At the end

Table 11.1: Process Functionality-Tests

ID	**T2**
Name	Performance-Tests
Process	The performance of the application as described in NFR2 is to be tested.
Method	Manual
Risk	Medium
Relevant for	Performance
Test environment	Client and Server
Interval	At the end

Table 11.2: Process Performance-Tests

ID	**T3**
Name	Usability-Tests
Process	The final application will be tested for its usability by showing it to at least 2 uninvolved people and having them play through certain scenarios.
Method	Manual
Risk	Medium
Relevant for	Usability, Compatibility
Test environment	Client
Interval	At the end

Table 11.3: Process Usability-Tests

11.4.7 Risks

The tests are associated with different risks. Tests that cover critical aspects of our project (parts that the project cannot function without) are associated with high risk. Other tests like the usability-test are not as high of a risk as the core goal of the project can still be achieved without them fully succeeding.

11.4.8 Definition of Done

The use-cases are defined as done when the following criteria are achieved:

- All tests pass.
- Coding guidelines were complied with.
- The process and the result are as described for the use case.
- The implementations correspond to the specifications described for the NFRs.

Chapter 12

Tooling

12.1 Documentation Guidelines

The following guidelines for the documentation were worked out:

1. Document in english
2. Document using LaTeX.
3. Use the given template.
4. Use meaningful titles that divide the text well.
5. Use clean formatting that helps to make the text easy to read.
6. Identify references from external sources as such.
7. Write a caption for each figure and table.
8. Write reference names of the captions for figures and tables in the kebab-case.
9. When drawing a diagram, use the same Flowchart maker, diagrams.net (former draw.io).

12.2 Code

12.2.1 Code Guidelines

The code should be written according to the following guidelines:

1. Write TypeScript variable and function names in camelCase.
2. Write TypeScript constants in uppercase, with individual words separated by underscores.
3. Write HTML element identifiers in kebab-case.

4. Give arrays a plural name.
5. Write comments where necessary, the code should be written in a self-explanatory way.
6. Use the same words for the same concepts.
7. Use ESLint/PyLint so that the code is uniform.

12.2.2 Coding Setup

To comply with the coding style, the standard rules of ESLint and PyLint are used. An additional rule has also been added:

- No log calls with the console, instead a separate logger should be used for the final product. (Does not prevent pushing but gives a warning).

In addition, certain processes are automated in the coding projects to facilitate development. The commands and their functions are described in the README.md and package.json files:

Backend

- README.md

Frontend

- README.md
- package.json

Before changes can be committed, the linter and prettier are automatically started using Husky hooks, if an error occurs during these processes, the commit is not executed. This ensures that changes follow the coding guidelines. The same happens with no-main-push but only when a commit is being pushed to the main branch of the remote repository. This is done to ensure that all relevant changes are merged using the GitLab GUI and a pull request and can then be viewed directly by another team member. The exception to this pre-commit-hook is the Documentation Repository.

12.3 GitLab

GitLab is used for version management. The project is divided into three main repositories. One each for the backend and frontend and one for the documentation.

1. Write commit messages in english.
2. At each weekly meeting, the changes are discussed and merged from the Develop branch into the Main branch.

12.3.1 Documentation Pipeline

Pipelines are used for the automatic creation of PDF files. These are executed with every commit to the documentation repository.

12.3.2 Dockerization Pipeline

To create docker images an additional pipeline was used. More information on the pipeline can be found in the Implementation chapter.

12.4 Issue Management / Time Tracking

Jira is used for issue tracking and management, as it also functions as a time tracking tool. To ensure this functionality, Jira was configured with additional fields in the task template.

12.5 Teams

Microsoft Teams with its own group is used to exchange information and inform about extraordinary meetings or ask for help.

Chapter 13

Time Tracking Report

13.1 Time per person

The following diagram shows the time dedicated by each team member for the project. Both team members contributed an equal amount of work towards the project.

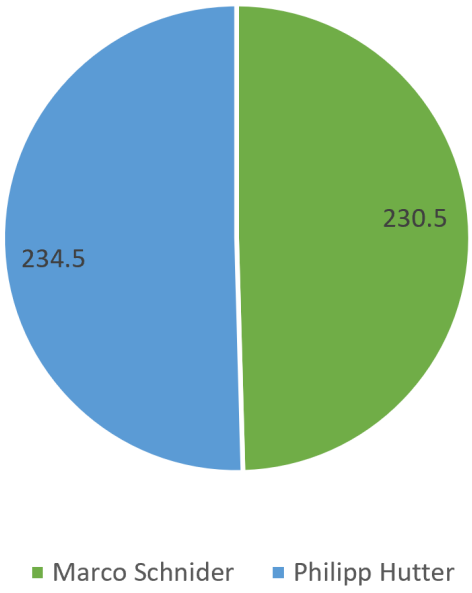

Figure 13.1: Diagram: Time per person

13.2 Time per phase

The following diagram illustrates the estimated time and actual time spend on the different phases of the project.

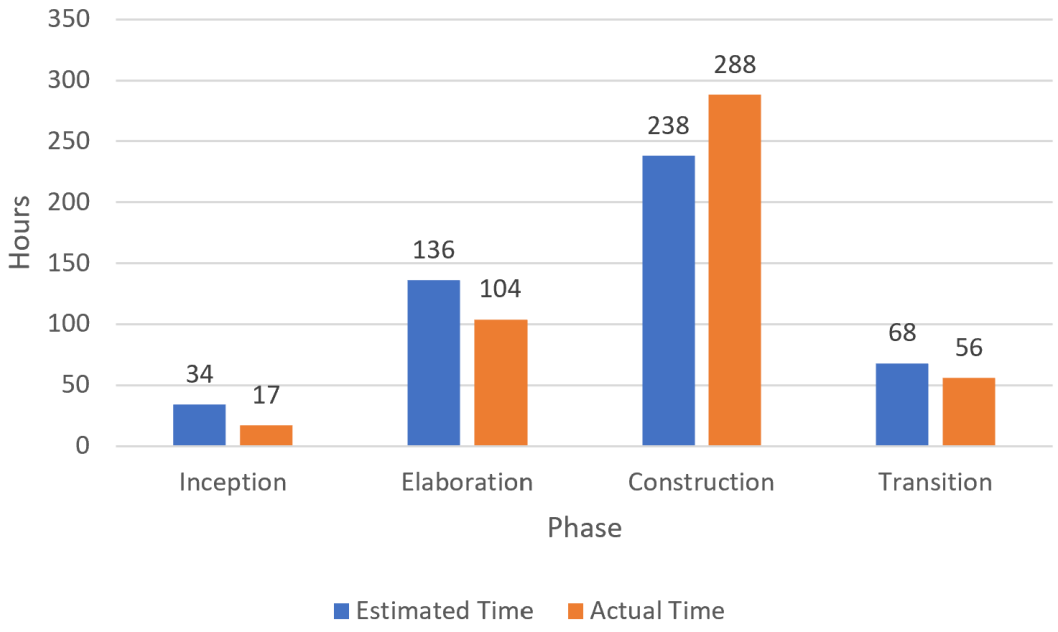

Figure 13.2: Diagram: Time per phase

Bibliography

[1] Component diagram. `https://c4model.com/diagrams/component`. Accessed: 2024-12-16.

[2] Container diagram. `https://c4model.com/diagrams/container`. Accessed: 2024-12-16.

[3] Homepage. `https://k3s.io/`. Accessed: 2024-12-19.

[4] System context diagram. `https://c4model.com/diagrams/system-context`. Accessed: 2024-12-16.

[5] Timo Kraus. Renewable-aware kubernetes scheduling to mitigate carbon emissions in distributed systems. Master's thesis, Technische Universität Berlin, 2022.

[6] Marco Aiello Tobias Piontek, Kawsar Haghshenas. Carbon emission-aware job scheduling for kubernetes deployments. *The Journal of Supercomputing*, 65(3), 2023.

List of Figures

1 Overview showing the nodes and pods iv

4.1 Domain model 6

5.1 Use case diagram 8
5.2 Mockup of the overview page 16
5.3 Mockup of the scheduler page 17
5.4 Mockup of the data page 18
5.5 Mockup of the debug page 19

6.1 System context diagram 21
6.2 Container diagram 22
6.3 Component diagram 23
6.4 Deployment diagram 24

7.1 Scheduling diagram 28
7.2 Ship parts created with Figma 30
7.3 Containers for the port 31
7.4 Crane for the port 31
7.5 Vehicles for the port 31
7.6 Port representing the master node 32
7.7 Overview page 33

10.1 Container diagram 40

13.1 Diagram: Time per person 54
13.2 Diagram: Time per phase 55

15.1 Questions and answers 1-2 66
15.2 Questions and answers 3-4 67
15.3 Questions and answers 5-6 68
15.4 Questions and answers 7-8 69
15.5 Questions and answers 9-10 70
15.6 Questions and answers 11-13 71

15.7 Questions and answers 14-15 72
15.8 Questions and answers 16-18 73

List of Tables

5.1 Description Use Case 1 9
5.2 Description Use Case 2 9
5.3 Description Use Case 3 10
5.4 Description Use Case 4 10
5.5 Description Use Case 5 11
5.6 Description Use Case 6 11
5.7 Description Use Case 7 12
5.8 Description Use Case 8 12
5.9 Description Non-Functional Requirement 1 13
5.10 Description Non-Functional Requirement 2 13
5.11 Description Non-Functional Requirement 3 13
5.12 Description Non-Functional Requirement 4 14
5.13 Description Non-Functional Requirement 5 14
5.14 Description Non-Functional Requirement 6 14
5.15 Description Non-Functional Requirement 7 15

10.1 Project Roles 39
10.2 Risk Matrix 43
10.3 Description Risk 1 43
10.4 Description Risk 2 43
10.5 Description Risk 3 44
10.6 Description Risk 4 44
10.7 Description Risk 5 44
10.8 Description Risk 6 45
10.9 Description Risk 7 45

11.1 Process Functionality-Tests 49
11.2 Process Performance-Tests 50
11.3 Process Usability-Tests 50

15.1 Test History: T1 64
15.2 Test History: T2 65
15.3 Summarized results of the Usability-Test 65

Part III

Appendix

Chapter 14

User Manual

To deploy our application, this manual can be used. It contains all the necessary information to install and run the application. It also contains information about the different functionalities of the application and how to use them.

14.1 Installation of the ReadVoltage Service

The ReadVoltage service reads the energy consumption of the worker nodes. It is necessary for the scheduler to work properly. The installation process is described in the following steps:

1. Clone the ReadVoltage Git Repository
2. Copy the readVoltage.service file to the systemd directory.
3. Reload the systemd daemon.
4. Enable the service to start on boot.
5. Start the service.

14.2 Prerequisites for Deployment

The following points must be provided before the deployment:

- Kubernetes Cluster with at least 2 nodes
- Traefik installed on the cluster
- ReadVoltage Service installed on the Worker Nodes
- Git installed on the nodes
- Access to the Git Repository

- Create a Git Access Token

14.3 Deployment

The deployment process is described in the following steps:

1. Clone the Deployment Git Repository
2. Apply the namespace-scheduler.yaml
3. Create a Kubernetes secret for the container registry in the namespace "scheduler"
4. Apply the service-account-scheduler.yaml
5. Apply the backend-scheduler.yaml
6. Apply the cors-middleware.yaml
7. Apply the ingress.yaml
8. Apply the frontend-scheduler.yaml

This will replicate the entire deployment. To achieve a different deployment, the same steps can be used, but the files need to be adjusted accordingly.

14.3.1 Available Routes

After the successful deployment, the following routes are available:

- / - Frontend
- /nodeInfo - [GET] Node Information
- /nodes - [GET] List of Nodes
- /pods - [GET] List of Pods
- /namespaces - [GET] List of Namespaces
- /schedule - [GET] Start a scheduling run
- /setPriority - [POST] Set the priority of a pod
- /move - [POST] Move a pod to a different node
- /scheduleWithCustomEnergy - [POST] Start a scheduling run with custom energy values
- /setBatteryThresholds - [POST] Set the battery thresholds for the scheduler
- /getBatteryThresholds - [GET] Get the battery thresholds for the scheduler

A detailed description on how to use them can be found in the API description chapter.

14.4 Deployment of the Sample Apps

To deploy the sample apps, the following steps need to be taken:

1. Clone the Sample Apps / Deployment Git Repository
2. Apply the namespace-counter.yaml
3. Create a Kubernetes secret for the container registry in the namespace "counter"
4. Create a Kubernetes secret for the container registry in the namespace "default"
5. Apply the counterup.yaml
6. Apply the counterdown.yaml
7. Apply the pictureone.yaml
8. Apply the picturetwo.yaml
9. Apply the picturethree.yaml
10. Apply the picturefour.yaml
11. Apply the counter-ingress.yaml
12. Apply the ingress.yaml

This will deploy the sample apps. To deploy a different set of sample apps, the same steps can be used, but the files need to be adjusted accordingly.

14.4.1 Available Routes

After the successful deployment, the following routes are available:

- /counterup - Counter Up
- /counterdown - Counter Down
- /pictureone - Solar Panel Raspberry Pi 1
- /picturetwo - Weatherproof Housing for a Raspberry Pi
- /picturethree - Solar Panel Raspberry Pi 2
- /picturefour - Sample Picture

Chapter 15

Test reports

15.1 Test history

The following tables show the test that were carried out and their results.

ID	**T1**
Date	11.12.2024
Input	Tested all requirements manually.
Expected Output	All requirements work as expected.
Actual Output	Use cases: • UC1: OK • UC2: OK • UC3: OK • UC4: OK • UC5: OK • UC6: OK • UC7: Not implemented. (Optional Use case). • UC8: Not implemented. (Optional Use case). NFRs: • NFR1: OK • NFR2: OK • NFR3: OK • NFR4: OK • NFR5: OK • NFR6: OK • NFR7: OK

Table 15.1: Test History: T1

ID	T2
Date	15.12.2024
Input	Check if the frontend is responsive enough to fulfill requirement set in NFR2.
Expected Output	The frontend checks at least every 5 minutes for changes in energy levels. .
Actual Output	If automatic scheduling is enabled the frontend checks for changes in energy levels every minute.

Table 15.2: Test History: T2

15.2 Usability-Test

In order to successfully carry out the Usability-Test, a form was created using Microsoft Forms. This results in a uniform process between the different tests and makes it easy to analyze the results.

15.2.1 Structure

The first questions are general questions about the tester, like their age or current job. Then they are asked to perform certain tasks and rate how easy and intuitive it was to complete them. The test is finished off by asking general questions about the application.

15.2.2 Results

The Usability-Test was performed with three different people between the ages of 16 and 22. Two of them have already completed their apprenticeship, while the other is in the process of doing so. This meant that the target group of people interested in studying computer science at a university of applied sciences was effectively covered. The tests yielded the following results:

Category	Remark
General impressions	The general impressions were very positive.
Using the website	No tester had any problems navigating the website. All of them could easily complete the tasks given to them. They found the website easy and intuitive to use.
Design	The design was well received by every tester.
Feedback	One tester wished for more feedback when a node was not available.

Table 15.3: Summarized results of the Usability-Test

15.2.3 Conclusion

The overall result of the Usability-Test is very positive. The only area identified for improvement is the feedback when a node is unavailable. Currently, it is a handled in a way that when pods cannot be moved onto a node it will not be highlighted in a green color as it usually would. However, given the time constraints and the fact that the other tester did not encounter issues with this functionality, it was decided to not prioritize any changes in this area.

15.3 Usability-Test questions and answers

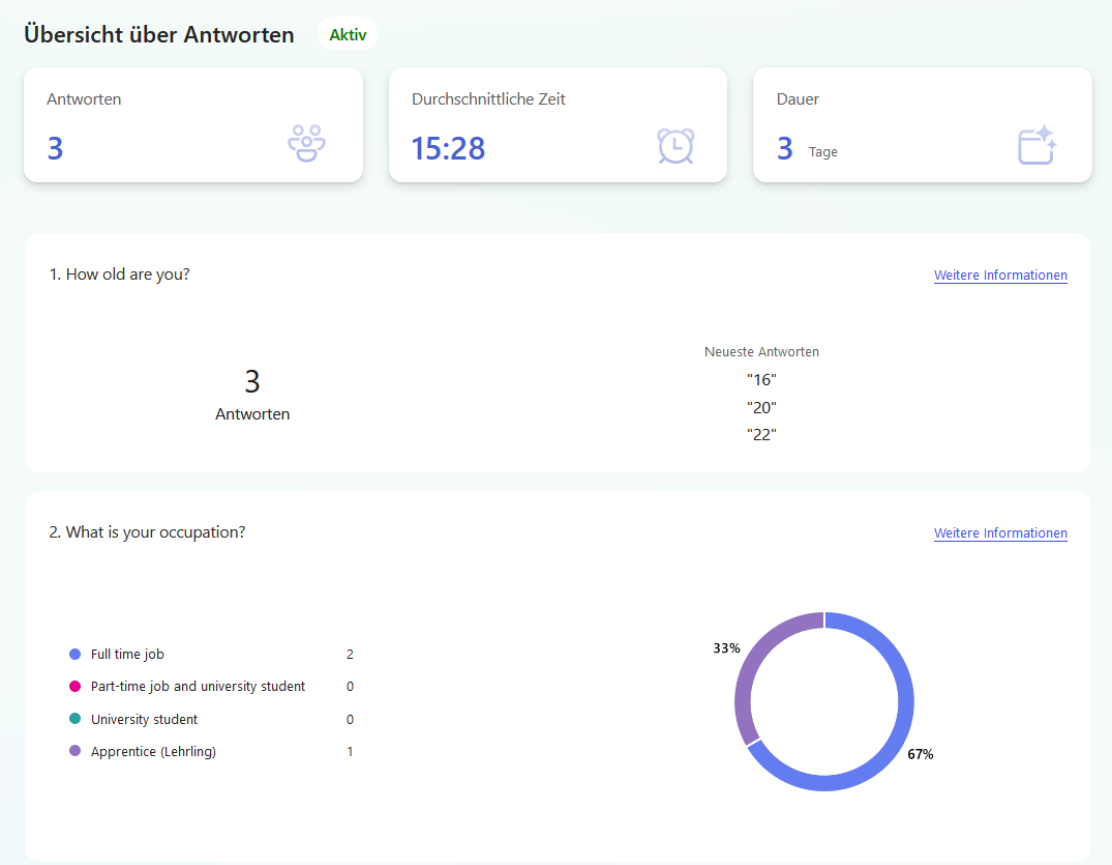

Figure 15.1: Questions and answers 1-2

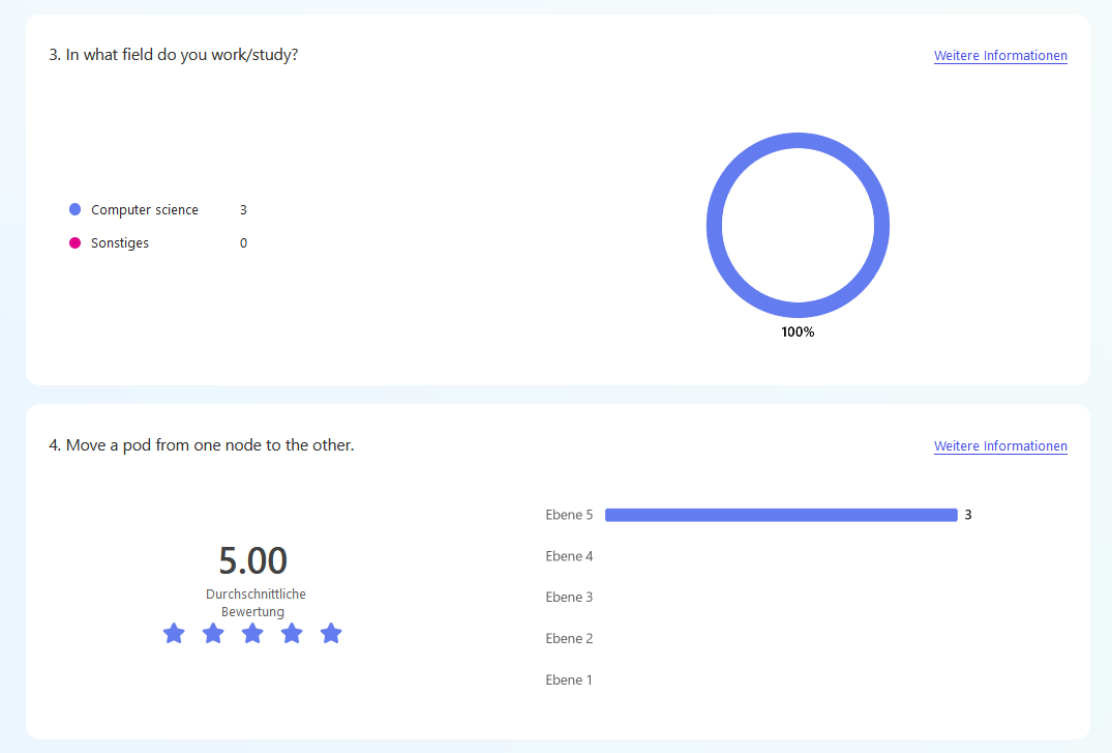

Figure 15.2: Questions and answers 3-4

Figure 15.3: Questions and answers 5-6

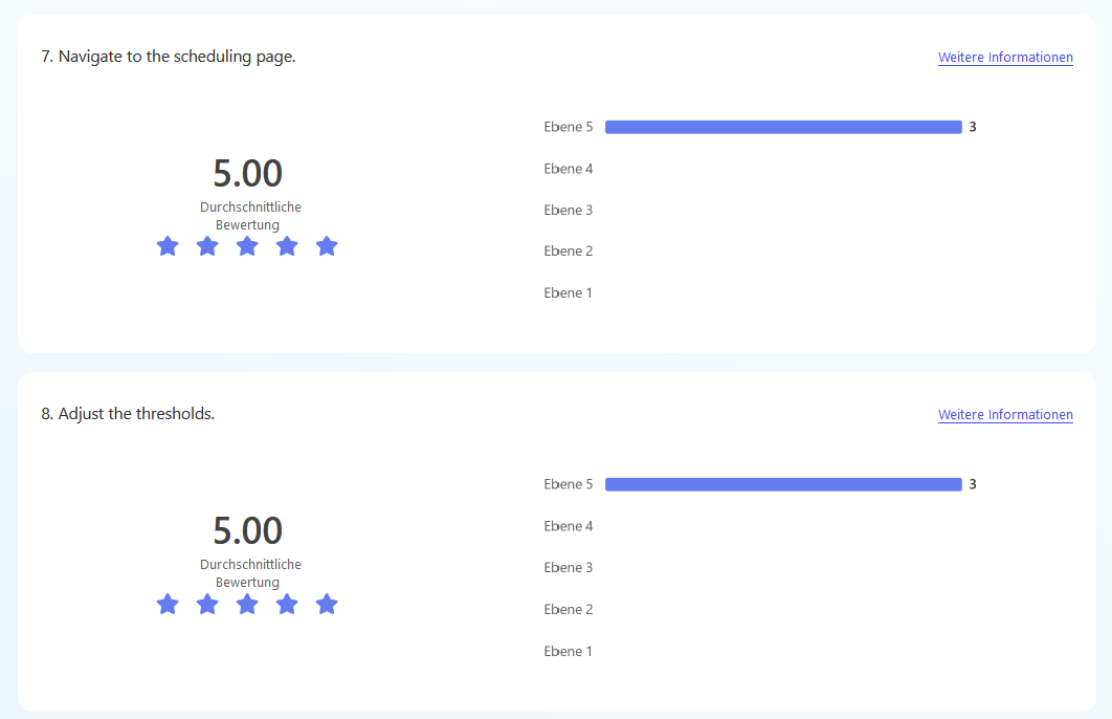

Figure 15.4: Questions and answers 7-8

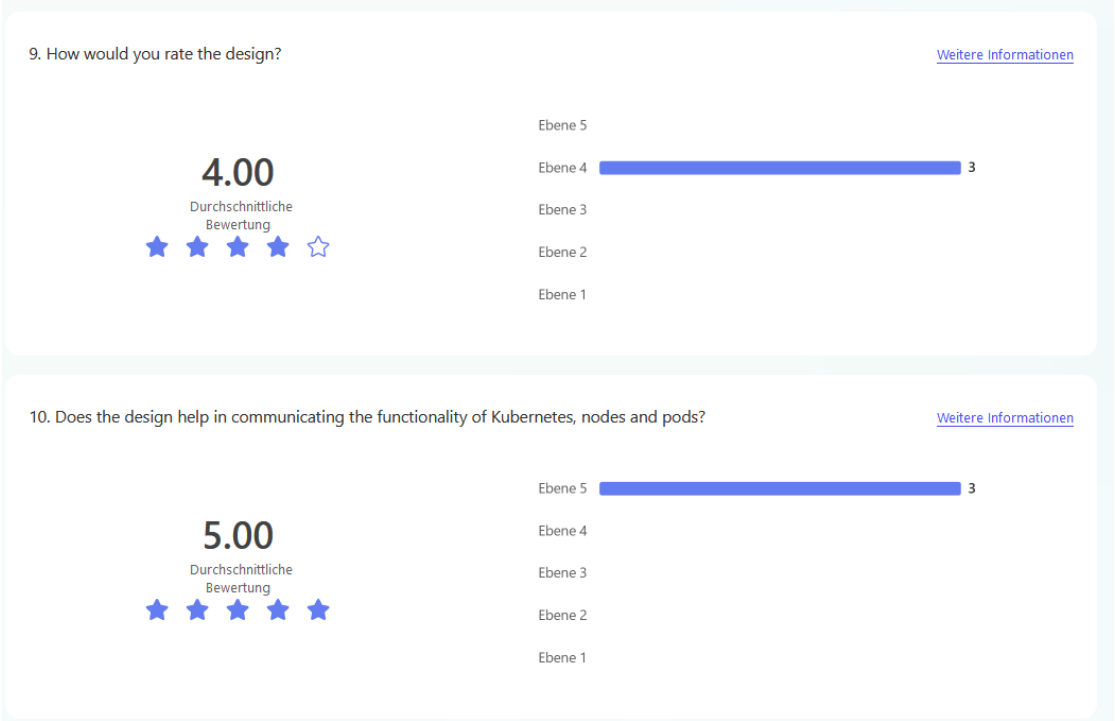

Figure 15.5: Questions and answers 9-10

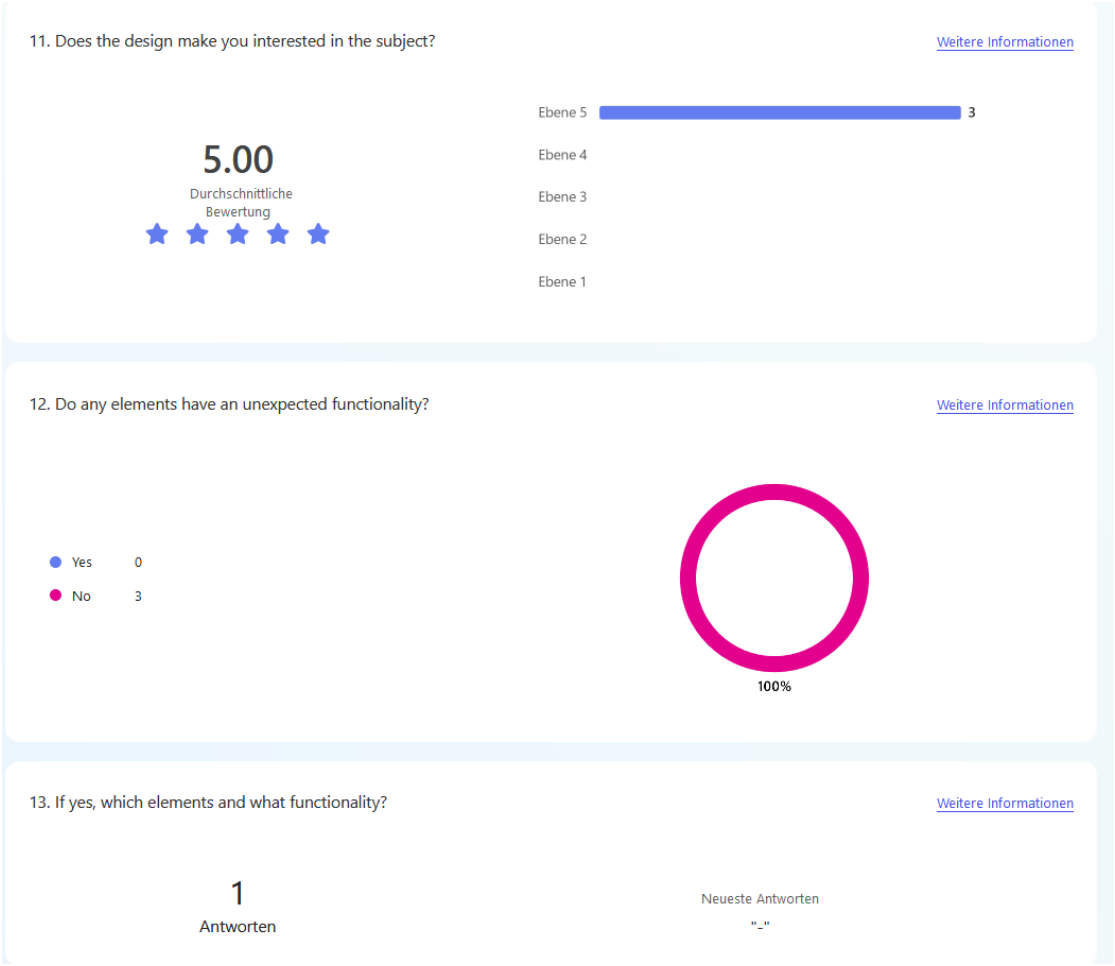

Figure 15.6: Questions and answers 11-13

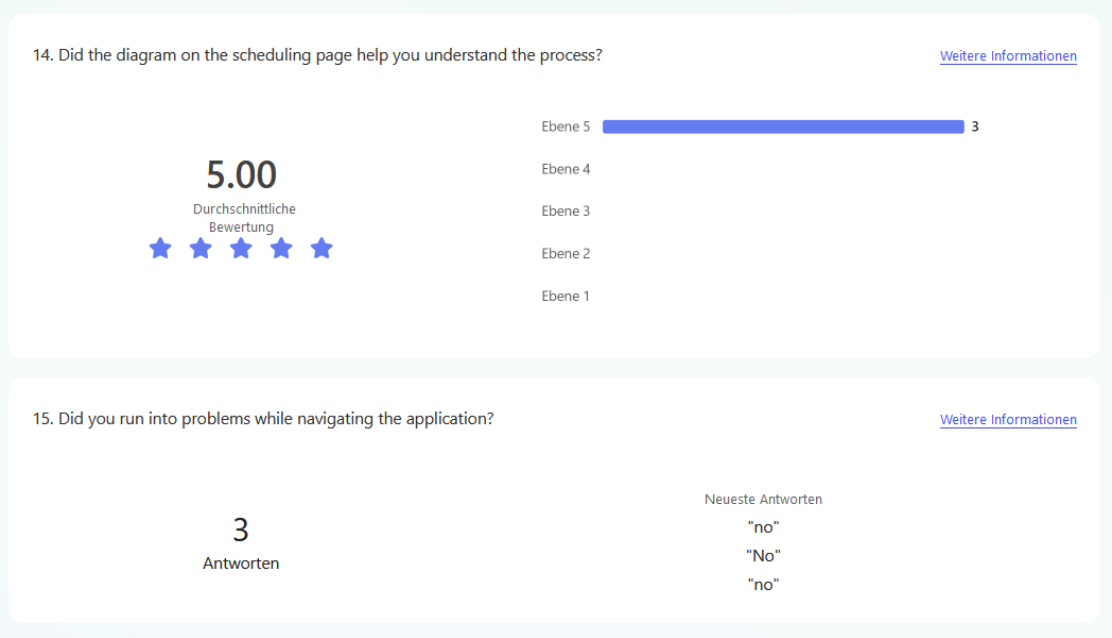

Figure 15.7: Questions and answers 14-15

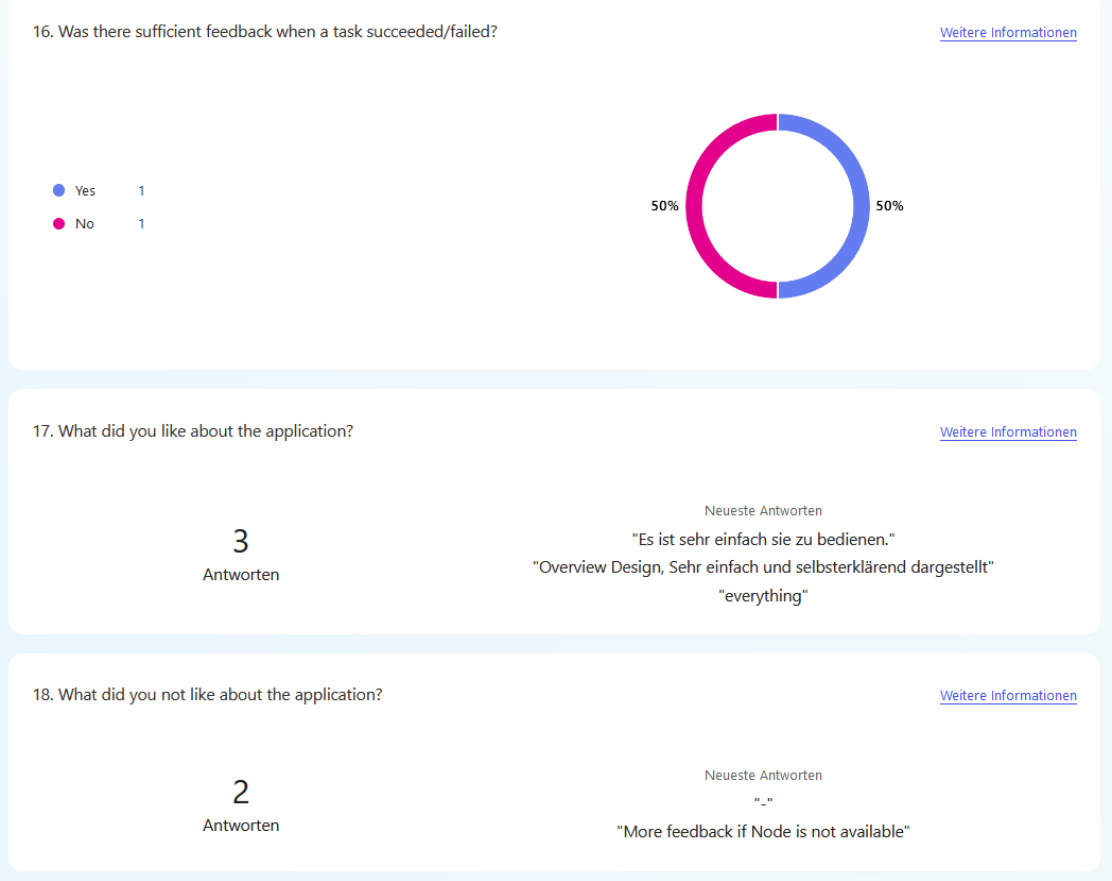

Figure 15.8: Questions and answers 16-18

Chapter 16

API Description

The following list shows all API calls that can be made to the backend.

/nodes

Returns all nodes.
HTTP-Verb: GET
Body: -

/pods

Returns all pods.
HTTP-Verb: GET
Body: -

/namespaces

Returns all namespaces.
HTTP-Verb: GET
Body: -

/nodeInfo

Returns information of the nodes: Energy data, what pods are running on them, if they are schedulable.
HTTP-Verb: GET
Body: -

/nodes

Returns all nodes.
HTTP-Verb: GET
Body: -

/move

Moves pod to specified node.
HTTP-Verb: POST
Body:

- `node: string` Name of the node the pod should be moved to
- `pod: string` Name of the pod to move
- `namespace: string` Namespace of the pod to move

/schedule

Starts the scheduler.
HTTP-Verb: GET
Body: -

/scheduleWithCustomEnergy

Starts scheduler with energy values provided.
HTTP-Verb: POST
Body:

- `energy: {[key: string]: Number}` Energy level of all active nodes

/setPriority

Sets specified priority of a pod.
HTTP-Verb: POST
Body:

- `pod_name: string` Name of the pod
- `pod_namespace: string` Namespace of the pod
- `priority: string` Priority to set the pod to

/setBatteryThresholds

Sets the different battery thresholds, which are used by the scheduler .
HTTP-Verb: POST
Body:

- `minBattery: Number` Threshold where pods get moved off of node
- `killMediumBattery: Number` Threshold where medium priority pods get killed
- `uncordonBattery: Number` Threshold where node gets schedulable

/getBatteryThresholds

Returns the current battery thresholds used by the scheduler.
HTTP-Verb: GET
Body: -