Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

Eduard Andrei Cristea
Jonas Tøsse
Richileu Alphonso Bailey
Torstein Eide

Refactoring and development of Web User Interface for a control system for flood-, bow- and searchlights

Bachelor's thesis in Computer Science Engineering
Supervisor: Arne Styve
May 2023

NTNU
Norwegian University of
Science and Technology

Eduard Andrei Cristea
Jonas Tøsse
Richileu Alphonso Bailey
Torstein Eide

Refactoring and development of Web User Interface for a control system for flood-, bow- and searchlights

Bachelor's thesis in Computer Science
Supervisor: Arne Styve
May 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

ABSTRACT

Luminell AS, a leading manufacturer and distributor of LED floodlights, bow lights, and searchlights for demanding environments, saw a need to update and refactor their web-based user interface for controlling searchlights. This project aimed to give Luminell's clients a better user experience by providing a robust and more capable web user interface. The project focused on enhancing the existing web user interface solution by refactoring the codebase and introducing new features and functionalities that improved the monitoring and control of connected searchlights.

The scope of the project included several additions to the existing solution, such as network settings, debug window, device control and monitoring features, and other functions proposed by the students. The team faced limitations related to the backend and firmware of the searchlight, requiring creativity to overcome these challenges while ensuring the project remained within its scope.

Challenges found in this project include inaccurate estimation of time, somewhat going outside the scope, and much more. The outcome of this bachelor thesis is an updated, dynamic solution that not only improves the monitoring and control of connected devices but also enhances the overall user experience for Luminell's clients. This project demonstrates the potential of computer engineering to optimize marine lighting products and highlights the value of collaborative efforts between academia and industry.

SAMMENDRAG

Luminell AS, en ledende produsent og distributør av LED-flomlys, bauglys og søkelys for krevende miljøer, så et behov for å oppdatere og refaktorisere nettgrensesnittet til Unity-Hub produktet sitt. Målet var å gi Luminells kunder en bedre brukeropplevelse i et ellers tøft miljø, ved å tilby et robust og mer kapabelt brukergrensesnitt. Prosjektet fokuserte på å forbedre den eksisterende web-brukergrensesnitt-løsningen, ved å refaktorere kodebasen og introdusere nye funksjoner og funksjonaliteter som forbedret overvåking og kontroll av tilkoblede søkelys.

Omfanget av prosjektet inkluderte flere tillegg til den eksisterende løsningen, som nettverksinnstillinger, feilsøkingsvindu, enhetskontroll og overvåkingsfunksjoner og andre funksjoner foreslått av studentene. Teamet møtte begrensninger knyttet til "backend" og fastvaren til søkelyset, noe som krevde kreativitet for å overvinne disse utfordringene samtidig som de sørget for at prosjektet holdt seg innenfor sitt omfang.

Utfordringer funnet i dette prosjektet inkluderer unøyaktighet i estimering av tid, gått noe utenfor rammen av prosjektet og mye mer. Resultatet av denne bacheloroppgaven er en oppdatert, dynamisk løsning som ikke bare forbedrer overvåking og kontroll av tilkoblede enheter, men som også forbedrer den generelle brukeropplevelsen for Luminells kunder. Dette prosjektet demonstrerer potensialet til datateknikk for å optimalisere marine belysningsprodukter og fremhever verdien av samarbeid mellom akademia og industri.

PREFACE

About

This is the report for the bachelor thesis "Refactoring and development of Web User Interface for a control system for flood-, bow- and searchlights". This project was concluded in May 2023 in Ålesund. The bachelor thesis has been conducted by:

Eduard Andrei Cristea

Jonas Tøsse

Richileu Alphonso Bailey

Torstein Eide

The project assignment was worked on, in close collaboration with Luminell Norway AS, with Frode Kolgrov as the company representative.

The bachelor project was chosen from a request made by Luminell for further development on their current web-frontend solution, which a group member already had worked on through the summer of 2022.

Motivations for this project were the group member's existing knowledge about this project and challenges, as well as the practical side of this project, as the project encompassed many trials and errors.

Acknowledgements

We would like to thank Frode Kolgrov for the ease of communication, access to services and materials, a space to work, and feedback throughout the project.

We would also like to thank our supervisor Arne Styve at the Department of ICT and Science at the Norwegian University of Science and Technology, for good advice and guidance throughout the entire process.

Assignment

The project client was Luminell Norway AS and our main contact was with Frode Kolgrov. Luminell is a company making light technologies. They are particularly strong on floodlights and searchlight systems. Luminell is using a gateway named "Unity-Hub" which gives external systems the ability to control the searchlights. Unity-hub had a web server with a simple user interface where all connected units could be viewed, and simple configurations could be made. They wanted a more advanced solution where searchlights could be controlled and monitored and relevant information regarding the lights could be viewed. The scope of the project consisted of several additions to the existing solution, and refactoring of the initial solution.

CONTENTS

Abstract i

Sammendrag i

Preface ii

Contents vi

List of Figures vi

Abbreviations ix

1 Introduction 1

1.1 Background . . . 1

1.1.1 The teams contribution . . . 2

1.1.2 System overview . . . 3

1.2 Problem . . . 4

1.3 Requirements . . . 4

1.4 Limitations . . . 6

1.5 Structure . . . 7

2 Theory 9

2.1 Refactoring . . . 9

2.1.1 Principles of Refactoring . . . 9

2.1.2 Benefits of Refactoring . . . 9

2.1.3 Common Refactoring Techniques . . . 10

2.1.4 Terms . . . 10

2.2 Morse code . . . 10

2.3 Client Server Communication . . . 11

2.3.1 HTTP . . . 11

2.3.2 REST API . . . 11

2.4 Multi-Paradigm programming . . . 11

2.4.1 Object-oriented Programming . . . 12
2.4.2 Functional Programming . . . 12
2.4.3 Event-driven Programming . . . 12
2.5 Design patterns . . . 12
2.5.1 Factory pattern . . . 12
2.6 Quality assurance . . . 13
2.6.1 GitFlow . . . 13
2.6.2 Merge requests . . . 15
2.6.3 Code review . . . 16
2.7 Development . . . 16
2.7.1 Version control . . . 16
2.7.2 Agile development . . . 16
2.7.3 Scrum . . . 16

3 Methods **21**
3.1 Materials and Tools . . . 21
3.1.1 Introduction . . . 21
3.1.2 Programming Languages and Frameworks . . . 21
3.1.3 Software Applications and Tools . . . 22
3.1.4 Development and Collaboration Tools . . . 22
3.1.5 Hardware Equipment . . . 22
3.1.6 The types of nodes . . . 23
3.1.7 External libraries . . . 25
3.1.8 Refactoring . . . 26
3.2 Project process . . . 26
3.2.1 Team . . . 26
3.2.2 Supervisor . . . 27
3.2.3 Client . . . 27
3.2.4 Meetings . . . 27
3.2.5 Development process . . . 28

4 Results **31**
4.1 Engineering result . . . 31
4.1.1 Overall results . . . 31
4.1.2 Refactoring . . . 32
4.1.3 Implemented functionality . . . 38
4.2 Frontend Architecture and Technologies . . . 45
4.2.1 Figma . . . 45
4.2.2 Consistency . . . 45
4.2.3 External libraries . . . 48
4.3 Administrative results . . . 51

4.3.1 Collaboration . . . 51
4.3.2 Confluence . . . 51
4.3.3 Jira . . . 51
4.3.4 Scrum . . . 52
4.4 Version control . . . 53
4.5 Constraints . . . 54
4.5.1 Obstacles . . . 54
4.5.2 Time estimation . . . 55

5 Discussion **57**
5.1 General discussion . . . 57
5.2 Engineering discussion . . . 57
5.2.1 Refactoring . . . 57
5.2.2 Page overview . . . 58
5.2.3 Light Controller . . . 59
5.2.4 Morse Module . . . 59
5.2.5 Statistics . . . 60
5.2.6 Settings . . . 61
5.2.7 Network Settings . . . 62
5.3 Administrative discussion . . . 62
5.3.1 Time estimation . . . 62
5.3.2 Work estimation . . . 63
5.3.3 Motivation and teamwork . . . 63
5.3.4 Remote vs on-location work . . . 63

6 Conclusions **65**
6.1 Problem solving . . . 65
6.2 Our contribution . . . 66
6.3 Further work . . . 66

7 Social impact **69**

References **71**

LIST OF FIGURES

1.1.1 Two CL-38 mounted on a ship's bridge 1
1.1.2 SL1 mounted on a rescue ship 2
1.1.3 SL2 in use 2
1.1.4 Reference diagram 3
1.1.5 Overall system layout 4
1.3.1 Project requirement 5

2.5.1 Factory pattern method example [8] 13
2.6.1 Gitflow workflow 14
2.6.2 Feature branches 14

3.1.1 SL1 23
3.1.2 SL2 24
3.1.3 CL-38 24
3.1.4 Unity Reference 25
3.2.1 Project roadmap 28

4.1.1 User-interface with sidebar and controllers for SL1, SL2 and CL38 . 32
4.1.2 API separated within the file structure 33
4.1.3 Instance of a searchlight being defined in JavaScript before refactoring 34
4.1.4 Definition of a searchlight using TypeScript after refactoring 35
4.1.5 Searchlight React Component Pre-Refactor and Post-Refactor . . . 36
4.1.6 Initial file structure 37
4.1.7 Refactored file structure 37
4.1.8 Dashboard / sidebar 39
4.1.9 Light controller SL2 40
4.1.10 Light controller SL1 40
4.1.11 Light controller SL2 with menu open 41
4.1.12 Light controller for the CL models 41
4.1.13 Morse module 42
4.1.14 Statistics page 43

4.1.15 Settings **Settings page** 44
4.1.16 Bridge dropped onto a boat view 44
4.1.17 Slots where lights can be dropped 45
4.2.1 Design guideline colors 46
4.2.2 Design guideline typography rules 46
4.2.3 Design guideline hierarchy rules 47
4.2.4 Design guideline rules for icons, shadows, and border rounding . . . 47
4.2.5 All MUI components used 48
4.2.6 Simplicity and customizability of the morse-converter 49
4.2.7 Circular slider for moving horizontally 50
4.2.8 Chart using ChartJS 50

ABBREVIATIONS

List of all abbreviations in alphabetic order:

- **API** Application programming interface
- **DOM** Document Object Model
- **HMI** Hydrargyrum Medium-Arc Iodide
- **HTTP** Hypertext Transfer Protocol
- **IP** Internet Protocol
- **LED** Light Emitting Diode
- **Mamsl** meter above mean sea level
- **MUI** MaterialUI
- **NTNU** Norwegian University of Science and Technology
- **PCA** Principal Component Analysis
- **REST** Representational State Transfer
- **TCP** Transmission Control Protocol
- **UI** User Interface

CHAPTER

ONE

INTRODUCTION

This chapter will introduce the project background, the project requirements, limitations, and the report structure.

1.1 Background

Luminell is a company with an impressive range of marine lighting products. Luminell has broad expertise in floodlights and searchlight systems. Using a device called Unity-Hub which acts as a gateway, external systems can communicate with Luminell's searchlights. For a couple of years, they have had a simple web-based user interface where lights and other connected devices could be viewed and simple configurations could be made. They wanted an updated, broader, and more advanced solution where lights and devices could not only be viewed but also monitored and controlled. In the summer of 2022, a team member started working on the project as an intern and saw the opportunities and potential for this project to become a bachelor's thesis.

Figure 1.1.1: Two CL-38 mounted on a ship's bridge

Figure 1.1.2: SL1 mounted on a rescue ship

Figure 1.1.3: SL2 in use

1.1.1 The teams contribution

In this project, it is necessary to be precise about which part of the code base is made by the team. To recreate this very project, the Unity Hub is needed. The team has not modified Luminell's existing back end, as this would be outside the

scope of the project. The team did get an already set up web UI, but this needed feature improvement. When refactoring, much of the existing functionality needed tailoring to fit the refactored standards. That way, Luminell's existing front end merely served as a template for layout and structure. Therefore, the great majority of the user interface functionality and logic is made by the team. The figure below illustrates the entire project system, with the part outlined in red being the only part that was within the team's scope. The figure describes Luminell's Unity Hub with Colornet and the Axon protocol as part of Luminell's proprietary technology (fig. 1.1.4).

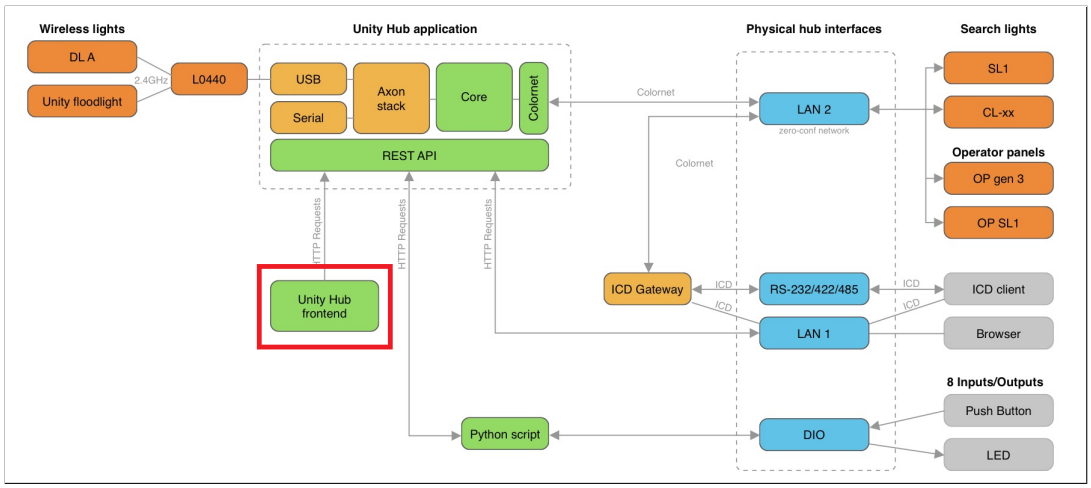


Figure 1.1.4: Reference diagram

1.1.2 System overview

A system delivered by Luminell is any desired searchlight and the Unity Hub. The searchlights can be controlled by either the Luminell operating panel or the web-based UI. The figure below (fig. 1.1.5) illustrates a representation of an example system.

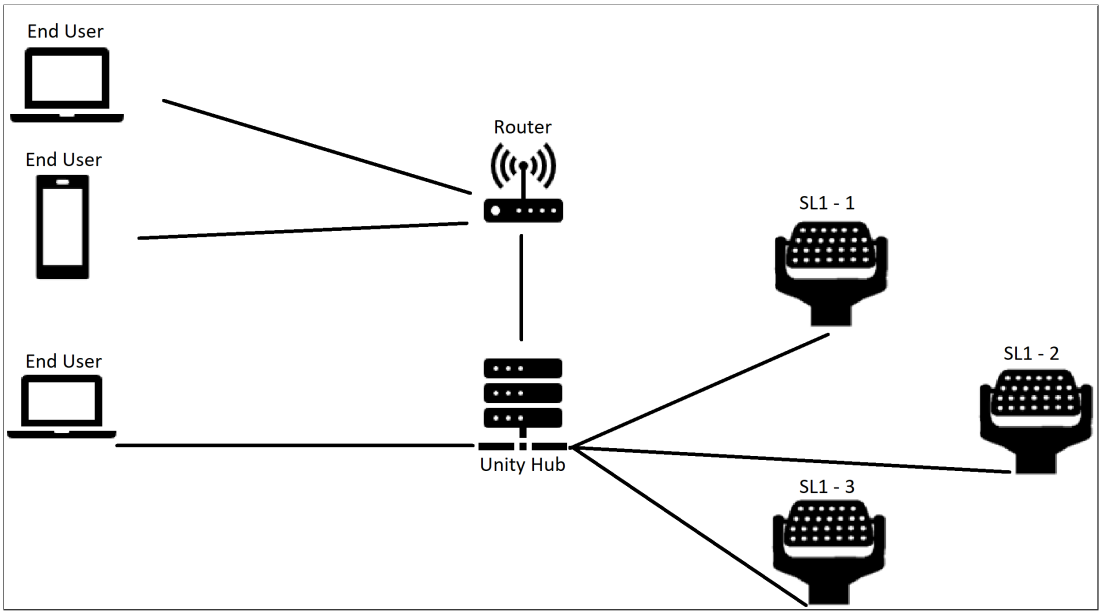


Figure 1.1.5: Overall system layout

1.2 Problem

Continuing development on an existing project takes much time to get into. Especially when the existing solution needs to be updated and is a particular solution with limited functionality. It then becomes a priority to update the technologies and develop new features.

1.3 Requirements

Since the existing solution was outdated, Luminell wanted an updated version with many new features. As mentioned, the Unity-Hub device acts as a gateway for third-party systems to control and monitor all activity from all connected devices, whether these are searchlights, floodlights, or even operating panels. With this in mind, Luminell made a list of requirements, for a bachelor's thesis for further development of the web user interface for Unity Hub. This list had a set of technologies currently being used, as well as a set of additions they wanted to add to the system. With this list of requirements, the team was free to decide on their own project requirement proposal.

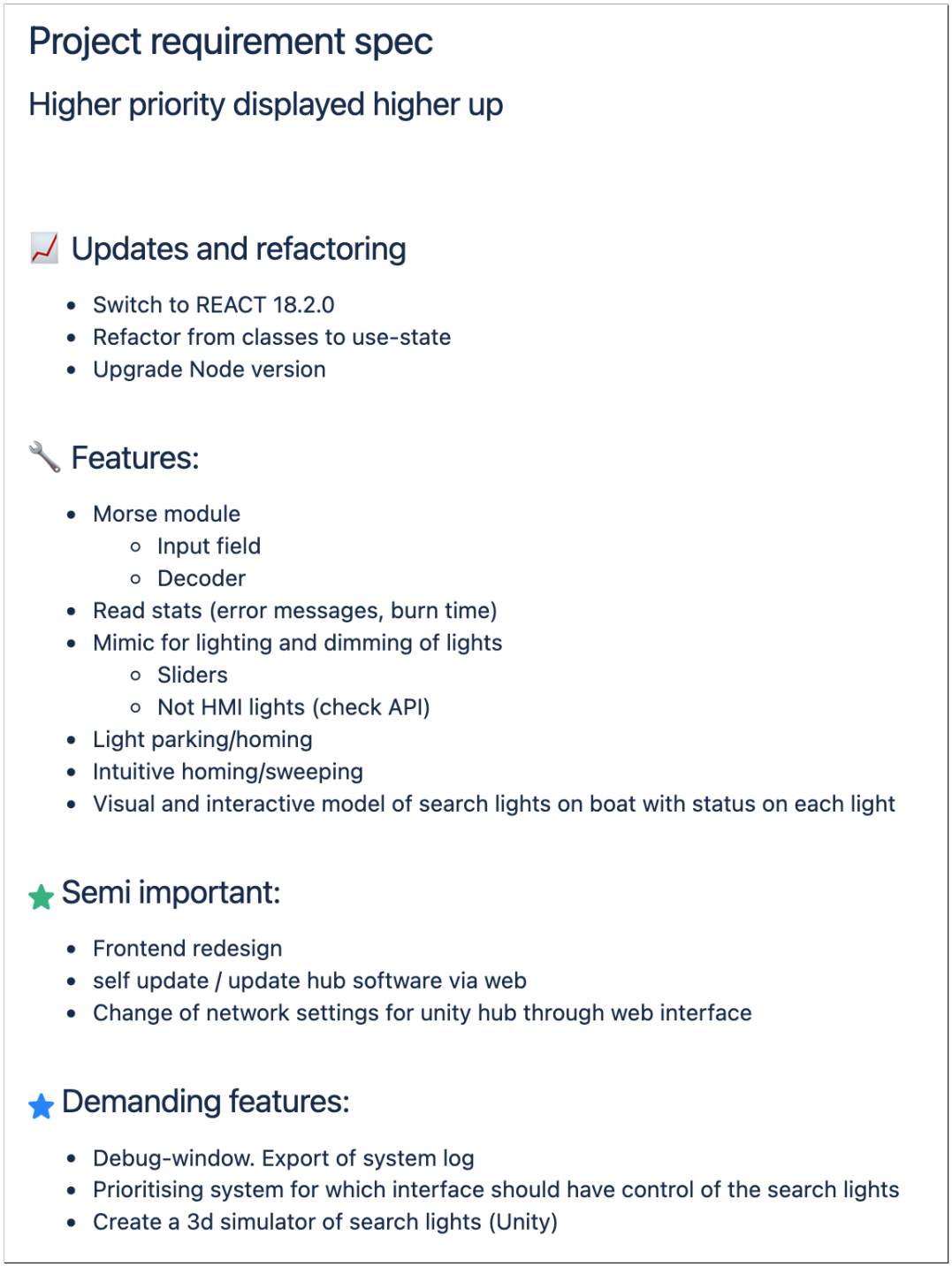


Figure 1.3.1: Project requirement

1.4 Limitations

The team was limited by the backend and the firmware of the searchlight. These limitations include a lack of functionality in the backend and inconsistency in the API documentation. Backend problems can be resolved, however, it is not part of the project scope.

Another limitation has been the course done in parallel with the bachelor's thesis. This course has taken three out of five weekdays throughout the entirety of the course. Leaving little to no time for the bachelor project during the 11 weeks the course lasted.

1.5 Structure

The rest of the report is structured as follows:

Chapter 2 - Theoretical basis: Contains theoretical background information for methods and technologies that have been used throughout the project.

Chapter 3 - Methods: Includes tools and methodologies used and describes how the team planned to approach and have approached the project.

Chapter 4 - Results: Describes the results of the project.

Chapter 5 - Discussion: Here an assessment of the team's results will be discussed. This will be an assessment in regard to limits, changes, and deviations in the project according to the original plans.

Chapter 6 - Conclusion and further work: Gives a conclusion of the project as a whole in regard to the requirements, based on results and discussion.

Chapter 7 - Social impact: Provides an overview on how the project can affect different aspects of our society.

CHAPTER

TWO

THEORY

2.1 Refactoring

Refactoring is a systematic process of improving the structure of existing software code while preserving its external behavior. It is an essential practice in software development aimed at enhancing the maintainability, readability, and modularity of the codebase without affecting the software's functionality. This section of the bachelor thesis provides an in-depth understanding of the theory of refactoring in programming, its principles, benefits, and various techniques employed. [1]

2.1.1 Principles of Refactoring

- Behaviour Preservation: Refactoring should not alter the external behavior or functionality of the software. It must aim at improving the internal structure of the codebase.

- Small Steps: Refactoring should be done incrementally, focusing on small, manageable changes to the codebase. This approach reduces the risk of introducing bugs and makes it easier to comprehend and review the code changes.

- Testing: A comprehensive test methodology should be in place before refactoring to avoid introductions of new bugs or regressions within the code changes.

2.1.2 Benefits of Refactoring

- Improved Code Readability: Refactoring enhances code readability, making it easier for developers to understand, maintain and extend the codebase.

- Reduced Complexity: By simplifying the code structure and breaking down complex functionality into smaller, more manageable pieces, refactoring helps reduce the codebase's complexity.

- Increased Maintainability: A well-structured, modular codebase is easier to maintain and debug, leading to increased efficiency and reduced development costs.

- Faster Development: A clean, well-organized codebase allows for faster development of new features and easier identification of resolution of bugs within a codebase.

2.1.3 Common Refactoring Techniques

There are several refactoring techniques in programming to improve code quality. Some of the most common techniques include:

- Extract Method: This technique involves breaking down a large, complex method into smaller parts, improving code readability and maintainability.

- Renaming: Renaming parts of code such as variables, methods, and components to be a more descriptive, meaning.

- Move Method/Field: Relocating methods or fields to more appropriate places can improve code organization and modularity.

2.1.4 Terms

Terms are often used by programmers to define parts of code.

- Dirty Code: Result of tight deadlines, mismanagement, inexperience, or shortcuts taken during the development process.

- Code Smells: Indicators of problems, often easy to spot and fix, but may show symptoms of problems with the codebase.

- Clean Code: Code that is easily read/understood and maintainable.

2.2 Morse code

Morse code is a method of communication that consists of a set of dots and dashes, transmitted as on/off signals, that represents letters and numbers. The communication method can be transmitted through technology like lights, radio, and telegraph. The speed of the Morse code transmission can vary, but a general implementation is: dots are one unit of time, dashes are three units, the pause between words is one unit, pauses between letters are three units, and the pause between words is seven units [2].

An example of this is: .... . .-.. .-.. — / .– — .-. .-.. -..
That forms the words: "HELLO WORLD"

2.3 Client Server Communication

Client and server communication are communication between a client, often used by a user, and a server, often a backend service in which the interpreter requests from the client and handles them accordingly. Client-server communication is used for inter-process communication. For this to work both sides need to follow the same protocol so they both know what to expect.

2.3.1 HTTP

HTTP or Hyper Text Transfer Protocol is a common protocol used for communications between a client and a server. HTTP clients make a connection with the IP protocol to a specific port on the server. The most used transport protocol is Transfer Control Protocol (TCP).

2.3.2 REST API

The application programming interface (API) serves as an intermediary, enabling applications or services to access resources from other applications or services. RESTful APIs use the representational state transfer (REST) architectural style to support seamless data exchange. Utilizing the HTTP protocol, RESTful APIs allow users to perform various operations. These operations are known as CRUD (Create, Read, Update, and Delete) and are performed through specific request methods. [3]

These CRUD operations correspond to the following HTTP methods: POST for creating new resources, GET for reading or retrieving resources, PUT for updating existing resources, and DELETE for removing resources. Each operation manages the resources within an application or service. These ensure that users can effectively interact with, modify, and maintain data. By implementing the CRUD operations in a RESTful API, developers can create a flexible and scalable system, streamlining data management and promoting interoperability between different applications and services.

2.4 Multi-Paradigm programming

Multi-paradigm programming is a software development approach that leverages multiple programming paradigms to solve problems effectively and efficiently. A programming paradigm is a fundamental style, approach, or methodology used to structure, design, and build software programs. Different paradigms emphasize different aspects of software development, such as data manipulation, code organization, or computation models.

The theory of multi-paradigm programming suggests that no single programming paradigm can address all the complexities and requirements of modern software development. By combining the strengths of multiple paradigms, developers can

create more flexible, scalable, and maintainable software systems.

Multi-paradigm programming advocates for the use of different paradigms based on the specific needs of a project or problem domain. By embracing multiple paradigms, developers can create software systems that are better suited to handle the diverse and evolving challenges of modern computing.

2.4.1 Object-oriented Programming

Object-oriented Programming or OOP for short, emphasizes the organization of code around "objects," which are instances of classes that encapsulate data and behavior. This paradigm promotes modularity, code reusability, and abstraction through inheritance, polymorphism, and encapsulation. [4]

2.4.2 Functional Programming

This paradigm treats computation as the evaluation of mathematical functions, avoiding changing state or mutable data. Functional programming emphasizes immutability, first-class functions, and higher-order functions to enable more predictable and easily testable code. [5]

2.4.3 Event-driven Programming

This paradigm focuses on the flow of control in a program, which is determined by events such as user actions, messages from other programs, or sensor outputs. Event-driven programming is widely used in graphical user interfaces, real-time systems, and server applications. [6]

2.5 Design patterns

2.5.1 Factory pattern

A factory pattern is a design pattern, which leverages a factory-like structure to limit the amount of reoccurring code. It enhances the reusability and flexibility of code. A typical implementation of the factory pattern is to have a base class to be overwritten by derived classes, this makes sure to simplify object creation, hides complexity, and improves readability [7].

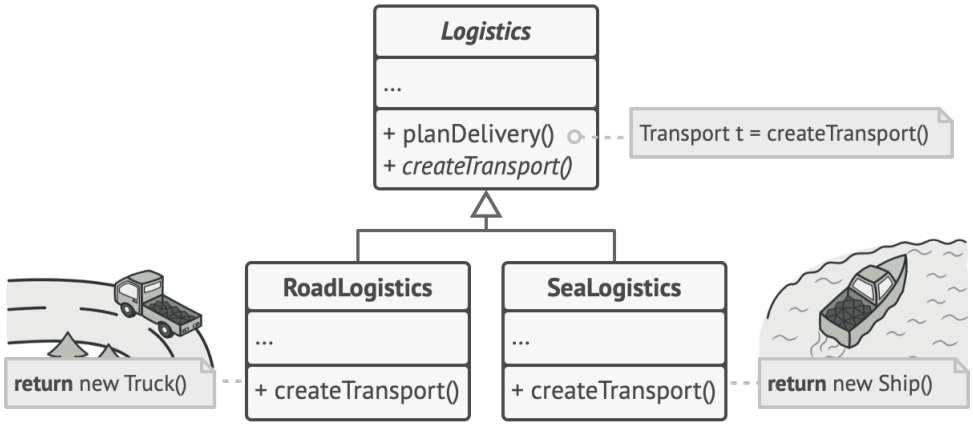


Figure 2.5.1: Factory pattern method example [8]

2.6 Quality assurance

2.6.1 GitFlow

Atlassian describes the GitFlow workflow as an alternative Git branching model that ensures good code quality and project integrity by separating the project into feature branches and two branches that record the history of the project named "main" and "develop" which have an infinite lifetime [9]. With this work methodology developers create feature branches and delay merging with the main branch until the feature is complete. GitFlow introduces a development branch from which all feature branches should be created, this branch will act as a less stable main branch. It will act as an intermediary between the main branch and feature branches, ensuring that the main branch always will be stable and operational. When finally pushing to the main it is convenient to tag the commit with a version number. The development branch will contain the entire history of the project. In addition to the "main", "develop" and feature branches, GitFlow frequently consists of release and hotfix branches.

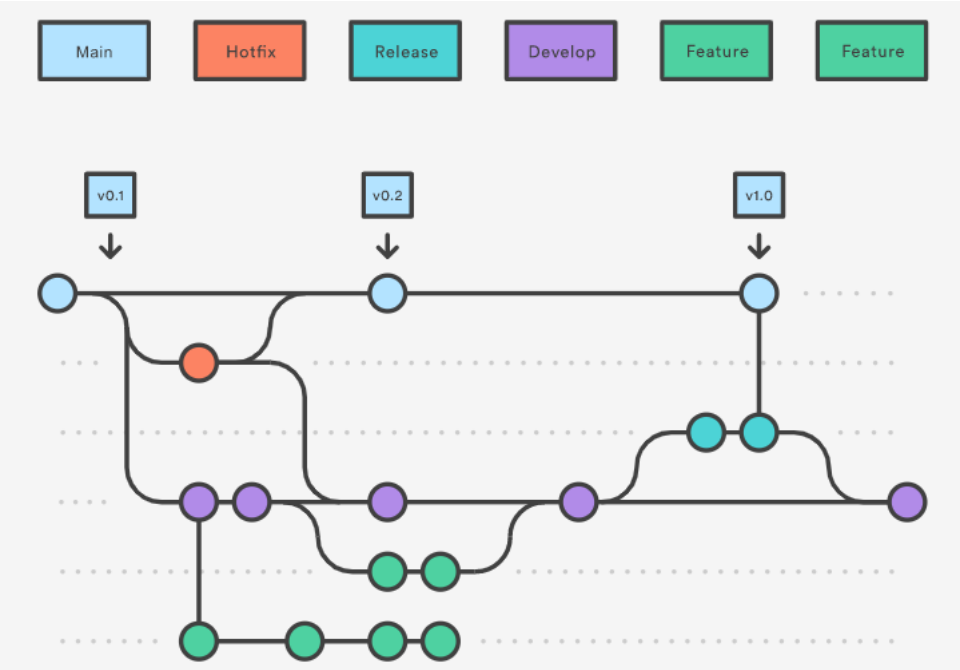


Figure 2.6.1: Gitflow workflow

2.6.1.1 Feature branches

Features branches are branches that are created for developing a certain feature. When creating feature branches they should originate from the develop branch and when done, merged back into the same branch. Feature branches should never interact directly with "main", and should rather be merged into "develop" which, when stable and ready for a new release, should be merged into "main". [9]

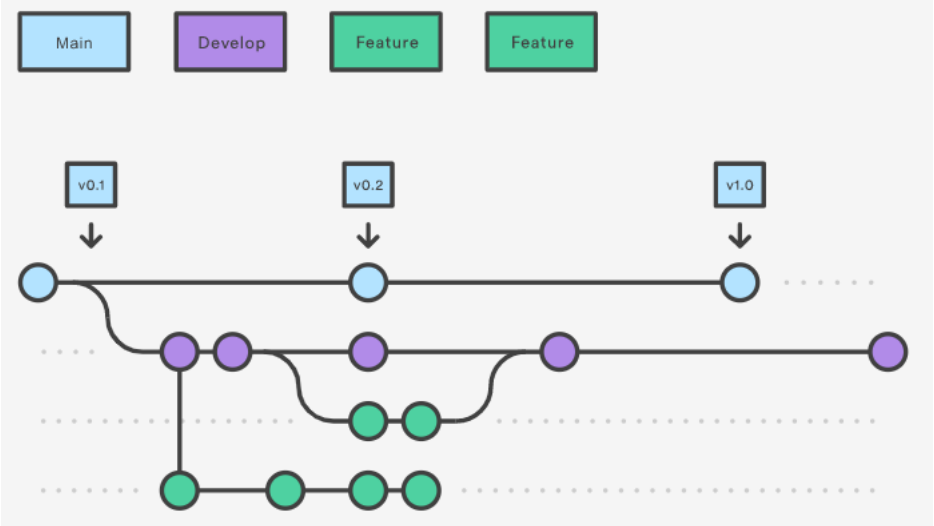


Figure 2.6.2: Feature branches

2.6.1.2 Release branch

The purpose of a release branch is to prepare for a new production release. This branch is forked from development and no new features should be added from this point, rather the branch should only be used for documentation generation, bug fixes, and other release-related tasks. Once the branch is ready, it should be merged into "main" with a release tag. In addition, it should also be merged back into development so it is up to date [9].

2.6.1.3 Hotfix branch

The hotfix branch lies between the "main" and "release" branches. This branch is the only branch that should fork directly from main. Once a fix is complete, the hotfix branch should be merged back into both main and develop, tagged with an updated version number [9].

2.6.1.4 Pros and cons

Pros:

- The main branch is always stable and production-ready, as it only gets updated after the development branch is thoroughly tested and deemed stable.
- Multiple features can be developed and tested simultaneously without interfering with each other, thanks to the use of feature branches.

Cons:

- Due to the increased number of branches and merges, the likelihood of merge conflicts increases, requiring developers to resolve conflicts frequently. This increases the amount of overhead. The process of maintaining multiple branches, merging them, and ensuring they are all up to date can be time consuming and may slow down the development process.

2.6.2 Merge requests

Merge requests can be requests made by a developer to merge their working branch into another branch. These merge requests often consist of a description, reasons for the changes, and links to related issues. Merge requests are often assigned a reviewer that is to review the code for any errors or faults, and can choose to approve whenever they see fit. This process enables collaborative programming and ensures good code quality and stability of code. [10]

2.6.3 Code review

After a developer is done developing a new feature in a feature branch. They naturally want to merge their changes and additions into an upstream branch. Before these changes are merged, it is good practice to perform a code review. A code review is an important tool in software development, as it allows second opinions on the solution and implementation as well as helps identify bugs, logic problems, and other issues with the code. Code reviews are important to ensure good code quality and transparency. In addition to the already mentioned benefits, code reviews also help to share knowledge across the team. When team members make changes they might learn new techniques and solutions. As everyone in the team has the ability to review and offer feedback, this prevents one person from being a single point of failure [11].

2.7 Development

2.7.1 Version control

Version control is a practice used to keep track of and monitor changes made to software code. It is an important tool within agile development as it can help create an overview of changes in code, hence reducing development time and increasing the chances of a successful deployment. Version control lets developers keep track of releases through branches and the merging of these branches. This enforces the concept of agile development as it helps release iterations of development, and if anything goes wrong, version control enables rollback to older iterations. [12]

2.7.2 Agile development

Agile development is an approach to software development that focuses on developing a product through iterations rather than having a big launch. The goal of agile development is to have continuous delivery of high quality and to create close collaboration among team members, as well as flexibility and adaptability within teams. This is so that teams can respond quickly to changes in requirements or priorities. Agile development introduces short, incremental cycles. Each cycle consists of planning, testing, development, and delivery of functionality. [13]

2.7.3 Scrum

Scrum is a framework within agile methodologies that facilitates the structuring of a group's work process and the creation of dynamic solutions for complex problems [14]. The Scrum framework helps the team maintain communication, consistency, quality, and deadlines [15]. To achieve this, Scrum utilizes an iterative and incremental approach, aiming to optimize productivity, predictability, and risk control [14]. The iterative approach involves the team planning, working through, and reviewing sprints, as well as reviewing the product [14]. The incremental approach refers to the group breaking down the problem into smaller tasks. The team then

adds tasks based on the workload they estimate they can handle and the priority of the tasks [14].

Scrum's Six Principles

For Scrum to work, there are certain theoretical processes that it emphasizes, such as Scrum's six principles.

- The empirical process is applied in Scrum to increase adaptability by having the group learn from their experience and plan sprints based on observed results and achievements[16].
- Self-organization: For Scrum to work, it is important that every member of the group is able to engage in self-organization and independent operation [14].
- Collaboration is important for Scrum so that the team can communicate and cooperate on changes and issues that arise during the process and thereby keep everyone on the team updated.
- Value-based prioritization is utilized to organize tasks so that the important tasks and those with short time limits are addressed first.
- Time-boxing is used to designate how much time each task is assigned to help the group keep track of time and prevents wasted time and delays.[14].
- Iterative development allows the team to adapt and manage changes more easily through the understanding that the project will have to be refined several times throughout the project's lifecycle [14].

The Team

A Scrum development team should consist of a small group of workers with the combined knowledge to create the project. The members should be treated equally without any hierarchy [15]. The team should also be cross-functional and take accountability.

Product Owner

The product owner of the team is the one who conceives and owns the idea. They are responsible for managing and organizing the backlogs [15]. It is important that they communicate the product goal to the team.

Scrum Master

The Scrum Master is responsible for facilitating the Scrum process and the communication between the team and the product owner. The Scrum environment they facilitate should also make it easy for stakeholders to inspect and adjust the sprint. This can be done by having a routine where the product

owner orders work for the problem by creating items in the backlog. Then, the Scrum team decides which items they add to the sprint during sprint planning, and concludes with the team and stakeholders inspecting and adjusting the results before the next sprint is planned. To foster this environment, the Scrum Master should help lead, train, and coach the organization in its Scrum implementation. They should also find techniques to facilitate communication between all members of the Scrum team and help the team plan Scrum implementations [15].

Sprints

Sprint Planning

In Sprint Planning, the team comes together to plan which items from the backlog they intend to work on for that particular sprint. The team also plans how they intend to solve the items and who will be responsible for solving them. While planning the sprint, it is important for the team to properly estimate the workload to avoid situations where there is a lack of tasks, resulting in wasted time, and also to avoid excessive workload.

Daily Standup

The Daily Standup is a short meeting that allows each member of the group to update the team on their progress, as well as issues and other changes. This helps detect, communicate, and address issues quickly. Daily Standup should not last long; a common rule is that members should be able to stand during the entire meeting, hence why it is referred to as a Standup. This allows stakeholders, the product owner, and other teammates to provide feedback. The backlog is also updated to reflect any changes in priority, requirements, or changes caused by issues.

Sprint Review

At the end of a sprint, a Sprint Review is held. The team reviews and discusses accomplishments, issues, and changes. During the Sprint Review, the group can check if any changes were made that drastically altered the sprint goal [15].

Sprint Retrospective

The Retrospective is also held at the end of the sprint. During the Retrospective, the focus is on how the group worked during the sprint and how that affected the outcome. If necessary, this allows for changes to be made to collaboration between group members and individual work ethics.

Scrum Artifacts and Items

In Scrum, items represent functions that need to be created or implemented. These functions are defined through User Stories, which have a title and description that explain what the user should be able to do once the User Story is completed. Epics are collections of User Stories that represent a user experience, and Issues are minor tasks broken down from the User Stories. Duties include all other charges that are not related to functions. To keep track of these items, Scrum artifacts are used. These tools help keep the project on track. The three Scrum artifacts are:

Product Backlog

The Product Backlog is the project's to-do list. It includes the project roadmap and requirements. The backlog should be updated whenever achievements are made or obstacles are met and after sprints. A backlog is a build-up or accumulation of prioritized tasks like user stories, changes to functionality, and bug fixes. An accurate and prioritized backlog reinforces the development team to work smart to get the "must have" functionality before the other tasks aren't as important. A backlog is necessary when using an agile workflow because sprint planning relies on the backlog to both scope and size the development tasks.

Sprint Backlog

The Sprint Backlog is a smaller to-do list created for a specific sprint. During Sprint Planning, the development team fills it with items from the Product Backlog and assigns them to team members. It is essential to choose high-priority projects during Sprint Planning; a well-managed backlog will make this easy. Having a good Sprint Backlog helps keep the team updated and focused [15].

Increments

Increments are defined as concrete stepping stones towards the product goal [15]. Every increment must be built upon all prior increments and verified before its addition, ensuring that all increments work together.

User Stories

User stories are a way for an end user or a customer to give an informal general explanation to describe wanted functionality or changes to the software. A user story aims to discuss with the end user to improve the software to the user's needs. A user story consists of four elements, known as "The four Cs," these elements being: the Card, The Conversation, The Confirmation, and The Context.

- The card is meant to establish "the" who, "what," and "the why." It contains who wants the change, what it is, and why it is needed.

- The conversation is vital to establish a dialogue with stakeholders or product owners to establish a shared understanding of what's necessary for a user story to reach a reasonable goal.
- Confirmation is a set of rules for agreeing when a user story is done. The warranty is given when all the criteria are met and the solution's owner formally accepts said resolution.
- The context is, as you might suggest, the context of the user story and how that user story relates to other user stories. The context will be critical when dealing with highly interconnected or somewhat overlapping stories.

Scrum Poker

Scrum poker, or planning poker, is an estimation technique agile teams use. It estimates a project's effort, complexity, or size. This method is used in Scrum development processes to make estimates for user stories or other functions in sprint planning.

CHAPTER

THREE

METHODS

3.1 Materials and Tools

3.1.1 Introduction

This section describes the materials, tools, and technologies utilized in the development of the bachelor thesis.

3.1.2 Programming Languages and Frameworks

The main programming languages and frameworks used in the development of the project were:

- HTML: For creating the structure of the web application. [17]
- CSS: For designing and styling the components and layout of the web application. [18]
- TypeScript: For defining interfaces and types for the data being used and returned. Typescript was used because it provides enhanced code quality by defining interfaces, classes, and other constructs that improve code organization and readability. In addition, it was also used because of its compatibility with JavaScript. [19]
- React: For building the user interface components and managing their state. React was used because it allows for creating single-page applications, this means that instead of re-rendering the entire web page upon interaction, only necessary components are re-rendered. Resulting in a more responsive and interactive user interface, making an overall better user experience. [20]
- Rust: For developing a mock of a searchlight node, implementing a REST API and used to create the service for changing network settings of the Unity Hub. Rust was used as the programming language of choice due to its powerful memory safety guarantees and performance benefits, this makes it an ideal choice for a high-performance REST API. [21]

3.1.3 Software Applications and Tools

The following software applications and tools were employed during the project:

- Figma: For creating wireframes and designing the user interface. Figma was used for its powerful ability to create interactable wireframes in a cloud-based environment, this allows several team members to work together at once. Additionally, Figma provides a user interface that is easy to use. [22]
- Postman: For testing API calls and learning about the searchlight's behavior. Postman was chosen for its capability to simplify the process of testing API endpoints by providing a user-friendly interface to send and receive HTTP requests and display their responses in an easily understandable format. [23]

3.1.4 Development and Collaboration Tools

To facilitate efficient collaboration among team members and manage the development process, the following tools were employed:

- Git: For version control. Chosen for its powerful version control system that makes it easy to track changes made to code over time, and its ability to roll back changes, merge and review code [24].
- GitLab was used for collaborative code management, adhering to the GitFlow workflow model 2.6.1. GitLab was used specifically due to the codebase already being situated on the client's GitLab repository [25].
- Scrum: For project management and agile development methodologies. Scrum was chosen because of its ability to facilitate agile teamwork [26].
- Confluence: For handling administrative aspects of the project. This tool was chosen for its project management ability, as it offers a collaborative and customizable platform for the team to share and organize information. [27]
- Jira: Track sprints and issues. Jira was used because of its compatibility with Confluence and because it provides an intuitive environment for planning, tracking, and releasing software. [28]
- Discord: For instant messaging with team members. Discord was chosen due to every team member already using it on a regular basis. Additionally, discord provides the ability to easily create new channels, making it easy and quick to set up. [29]

3.1.5 Hardware Equipment

Hardware equipment played a crucial role in the development and testing of the project. The main hardware components used were:

- SL1 Searchlight: Loaned to the team by Luminell for development and testing purposes.

- Unity Hub: Served as the server, running a custom Linux-based distribution hosting the Unity Hub application. The Unity hub contains the backend and the REST API, which is necessary for communicating with searchlights through third-party interfaces.

3.1.6 The types of nodes

The reasoning to include some different searchlights designed by Luminell in this chapter (**SL1** 3.1.6, **SL2** 3.1.6, **CL-38** 3.1.6) is to underline the distinction between the different lights. There are important differences in limitations to be considered when designing components in the frontend solution. Furthermore, the team will also describe the Unity Hub in this chapter.

SL1

SL1 is a led searchlight designed by Luminell AS. The searchlight has a single led module, as opposed to the SL2 (Fig. 3.1.1). The searchlight has a servo base with full 360° free range in horizontal movement. In vertical movement, however, the servo has 120 degrees of movement.

Figure 3.1.1: SL1

SL2

SL2 is also a searchlight designed by Luminell AS. The SL2 has two led modules that are oriented vertically. The SL2 has 360° free range horizontal and vertical movement. The searchlight is shown below (Fig. 3.1.2)

Figure 3.1.2: SL2

CL-38

CL-38 is a searchlight with long-range direct lighting because of its HMI technology. Burn time (i.e., the amount of time the light has been on) is a related topic with the HMI lamp technology having a burn time of about 400-500 hours, which, as opposed to LED techonlogy, having a burn time of around 50 000 hours. In contrast to the earlier discussed LED lights, the HMI lamps have a longer startup routine, and if the light are turned on and off in quick succession multiple times, the bulbs could be damaged. The CL-38 has the same base as SL2 and has a 360° free range horizontal and vertical movement. The searchlight is shown below (Fig. 3.1.3)

Figure 3.1.3: CL-38

Unity Hub

The Luminell Unity Hub is designed to be the central gateway unifying the Colornet and Axon protocol by exposing an easy-to-use REST API. It allows the retrieval of system information and the current status of connected nodes. Unity Hub can be referred to as a server. It runs a custom Linux-based distribution that

hosts the Unity Hub application. The diagram below is for reference purposes only. Details might be imprecise or intentionally left out.

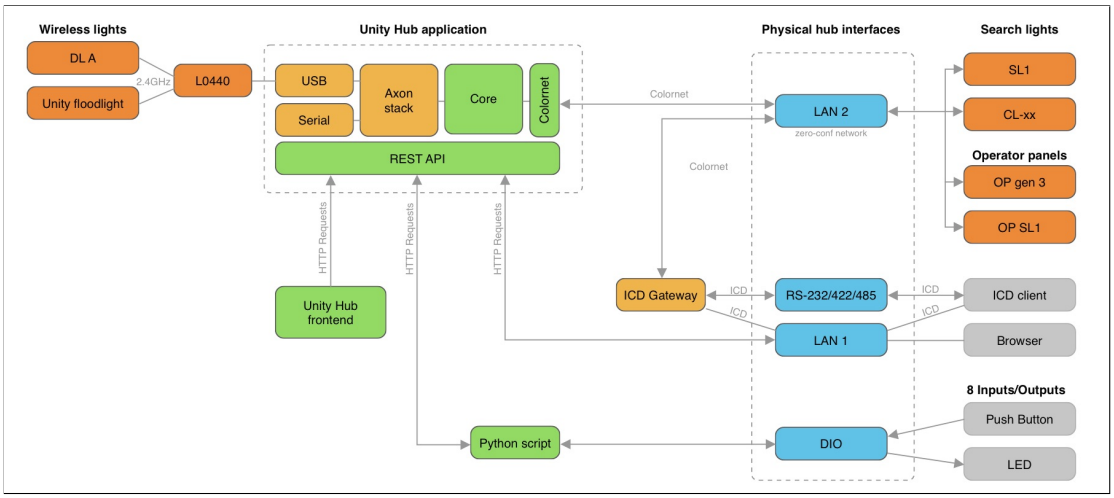


Figure 3.1.4: Unity Reference

Client-node communication

The interaction between the client and the nodes is facilitated through a front-end interface, which presents the user with pertinent information and control mechanisms for light manipulation. This interface is supported by a RESTful API integrated within the Unity Hub central gateway. Communication between the front-end client and the Unity Hub gateway employs the JavaScript Object Notation (JSON) format to transmit data. Subsequently, the Unity Hub gateway executes requisite logic to guarantee accurate translation to and from a node. Nodes within the system represent devices such as, but not limited to, searchlights, operator panels, and wireless lights.

3.1.7 External libraries

Several external libraries were used across the project. Libraries often provide pre-built and tested solutions to common problems, due to this libraries were added to save time on creating components and implementing functionality.

- FontAwesome: FontAwesome provided the front end with free icons that fit well with the theme of the web page. It was chosen due to the vast variety in customizable and scalable icons it introduces [30].
- MaterialUI: Used for its components. Chosen because MaterialUI introduces many pre-built components that are responsive, visually appealing, and customizable [31].
- ChartJS: ChartJS was used to create an interactive and responsive chart for displaying information from lights, It provides an easy-to-use API for creating charts that work across multiple browsers [32].

- Lodash: Used functionality to only send requests in intervals or on events. Was chosen to avoid boilerplate and come up with a proprietary solution that this utility library contains.
- morse-converter: Used to translate plain text into Morse code. This library was chosen due to its easy-to-use nature [2].
- react-dnd: used for implementing drag and drop functionality. Chosen because of its simplicity, customizability, and large community [33].
- Axios: For sending HTTP requests. Axios was chosen for its simple, lightweight way to handle HTTP requests and responses [34].

3.1.8 Refactoring

The team utilized refactoring to update the initial code and make it more easily scalable. Refactoring was used on the following aspects of the project.

- **Frameworks**: To update the frameworks to newer versions.
- **Code**: To fit the standards of the updated frameworks.
- **Styling**: To follow a single naming convention across the entire project.
- **File structure**: To make it easier to navigate when the project scales up.

3.2 Project process

3.2.1 Team

The development team in this project consists of four bachelor students from NTNU Ålesund: Richileu Alphonso Bailey, Eduard Andrei Cristea, Jonas Tøsse, and Torstein Eide. Adhering to the Scrum roles, the team was assigned as the development team (2.7.3). Furthermore, Eide was designated as the "daily Scrum Master" to oversee all team Scrum-related activities on a daily basis.

Roles and task distribution

Aside from being designated as the development team, each member of the team was also assigned a specific role to aid in effective project management and supervision, which extended beyond the Scrum framework. The objective of these roles was to ensure timely submissions, especially the team leader role, which was crucial to ensure every member was motivated and on time with their assignments. These roles were suggested as tightly attached to the academic part of the project, such as the pre-project plan and report.

A draft of the distribution of roles and responsibilities was drafted in a group contract during the commencement of the project. According to the original document, Eide was designated as the team leader and report manager, while Cristea

was appointed as the meeting manager, and Bailey and Tøsse were assigned the roles of document and submission managers. Additionally, all team members were assigned the responsibility of quality control to ensure comprehensive oversight of the project.

3.2.2 Supervisor

Arne Styve is an associate teaching professor at NTNU in Ålesund. Styve's role in this project is that of the Scrum Master and project supervisor. A Scrum Master's role is to ensure that the Scrum framework is followed and should act as a coach for the rest of the team (ref. 2.7.3). A project supervisor's role is to assist the team with technical and theoretical information and give the team advice throughout the project's lifetime.

3.2.3 Client

Luminell Norway AS is the client for this project, with Frode Kolgrov as their representative fulfilling the role of the product owner as per the Scrum methodology (ref. 2.7.3). The team will deliver the final product to Luminell Norway AS.

3.2.4 Meetings

Regular meetings were to be held with both the client and supervisor to ensure the project progressed smoothly. At the conclusion of each sprint, meeting notes would be generated to document the feedback provided by both the product owner and supervisor. These meetings took place at either the NTNU or Luminell's office, either in person or via Teams. Initially planned for a duration of thirty minutes, the meetings were extended when necessary to discuss significant features or challenges encountered by the team.

Mettings with client

In-person meetings with clients were held every other week, to ensure that product owners were up to speed on development and potential difficulties regarding existing software. Important decisions both for design and functionality were also discussed in these meetings. Even though the meetings were held bi-weekly, the client attended the meetings with the supervisor, on teams, whenever they wanted.

Meetings with supervisor

The team held bi-weekly meetings with their supervisor on the weeks when they did not have in-person meetings with the client. These meetings served as an opportunity to discuss the project status, as well as any issues or concerns with the Scrum workflow or work methodology. The team used these meetings to receive guidance and feedback from their supervisor to ensure the project was on track.

3.2.5 Development process

Project setup

This section outlines the initiation phase of the project, namely the project setup phase. During this stage, the project team established a framework for the project by creating Confluence and Jira spaces that accommodated the administrative and workflow aspects of the project. Additionally, a roadmap was developed and approved by the supervisor and client representative, outlining the project's objectives (fig. 3.2.1). The setup phase also involved making Scrum-related decisions such as sprints, sprint reviews, and daily stand-ups, as well as scheduling mandatory physical meetings. Lastly, the team finalized the planning phase by creating a product requirement specification (fig. 1.3.1).

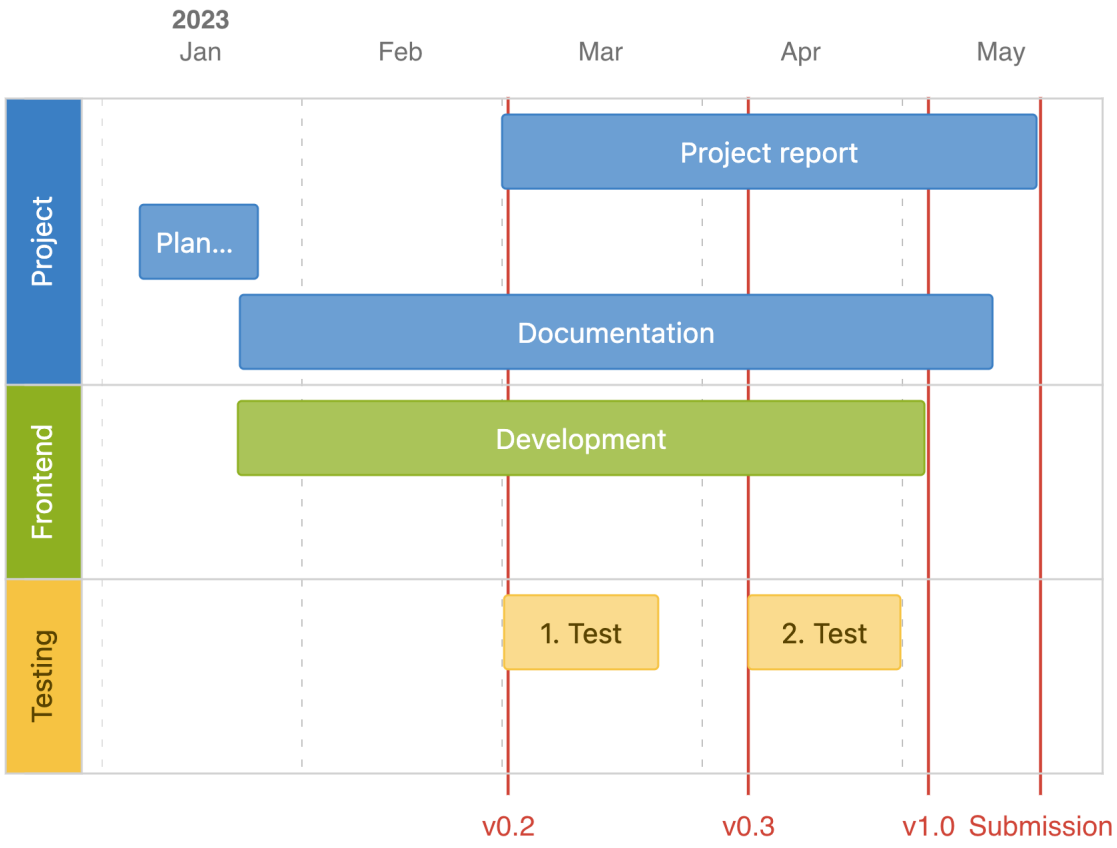


Figure 3.2.1: Project roadmap

Scrum

Scrum has been used as the agile management framework to provide a flexible and iterative approach to the development process and to provide structure to the development throughout the project's lifetime.

Sprints

During the project setup, it was agreed to have weekly sprints. After each sprint, a sprint review meeting with the client and/or Scrum Master was conducted, followed by planning for the next sprint.

Sprint Planning

As the name implies, sprint planning was utilized to prepare for the upcoming sprint. Although this procedure could be carried out in collaboration with the client and supervisor, it primarily involved the development team. This is due to the fact that story points were assigned to issues to estimate the necessary hours for each task and that the client prioritized issues in Jira. Based on these criteria, the team could independently determine the scope of each sprint.

Daily Stand-up

Throughout the sprint, daily standup was employed to maintain regular updates on the progress of both the sprint and the overall project. The daily standup did adhere to a specific schedule; instead, it was conducted just before the shared lunch. The duration of the standup was consistently kept within a five-minute timeframe. Within this brief period, each member presented their progress, issues, and changes that had occurred, facilitating clear communication of individual needs.

Jira

Jira was set up using Atlassian's template for Scrum projects, this template provided a backlog, issue board, reports, and a roadmap. This space was set up by the project supervisor and Scrum Master. Jira was used to create issues, which could be categorized as user stories, tasks, improvements, bugs, or epics. Each issue was required to have a summary, version tag, description, and label. The summary provided a brief overview of the issue, while the description provided more detailed information, including the completion criteria for the issue.

The backlog was populated with the created issues, which were ordered based on their priority. The higher a task was on the list, the higher it was prioritized. This allowed the client to adjust the priority of the issues by changing their order within the backlog.

Confluence

Confluence was configured to match the Jira space after using the Scrum template, it was also based on a development process template. Inside this space, a main page was constructed. This page included an introduction to the project, with information about the project, team, and issues from Jira. Additionally, pages for meeting notes, retrospectives, status reports, files, and decision logs were added.

These pages included a button for creating a new corresponding page with a template, as well as a list of all pages related to their parent.

Quality assurance

The team adopted the practice of utilizing merge requests to maintain high code quality. Essentially, when a feature branch is prepared to merge with another branch, a merge request is created (ref. 2.6.2). Subsequently, the merge request is assigned to one or more team members who review the changes made, commonly referred to as a code review (ref. 2.6.3). The reviewer evaluates the modifications, provides feedback, and comments, and ultimately approves or closes the merge request. In the event that the merge request is closed or changes are requested, it's essential to provide comments that outline the criteria for approval.

Furthermore, GitLab pipelines were employed to verify that each version of a branch was compiled correctly. Pair programming was utilized, ensuring a higher level of consistency and code quality.

CHAPTER

FOUR

RESULTS

4.1 Engineering result

4.1.1 Overall results

In this project, the team has developed a web-based application for Luminell AS to serve the Unity Hub (ref. 3.1.6). The application has been developed by using React and Typescript. The overall solution is a refactored and improved solution upon Luminell's web-based user interface that represents all searchlights connected through Luminell's Unity Hub. The final result included versions of React and Node that were up to date. Additionally, much of the existing code was refactored from class components to react functional components. All this resulted in a user interface with a navigational sidebar, used for navigating through the different pages and functionality.

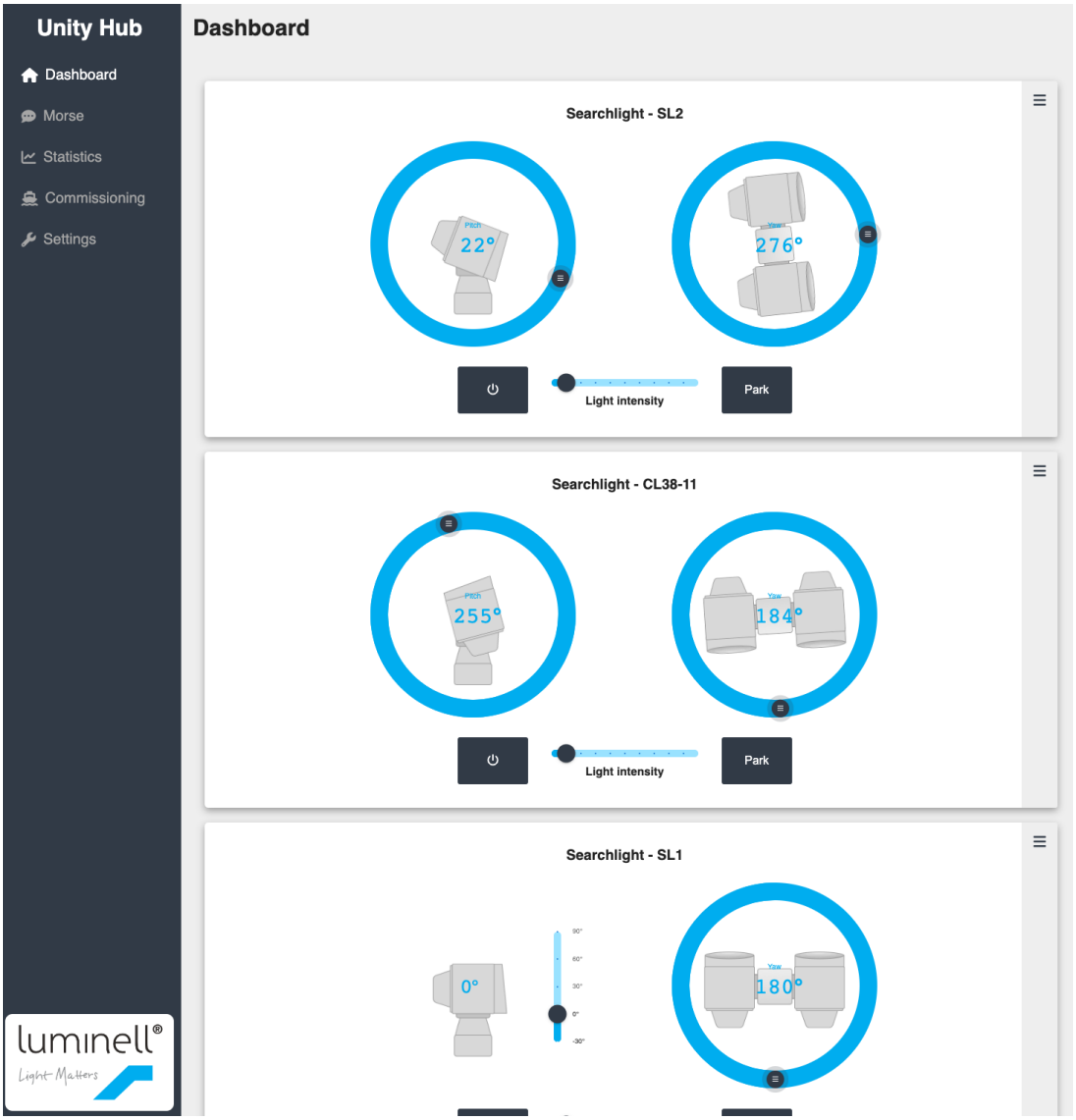


Figure 4.1.1: User-interface with sidebar and controllers for SL1, SL2 and CL38

Furthermore, extra features were added. These features include a morse module, an error -and statistics page, dimming of lights, parking, sweeping, surveillance, and network settings. Looking at the requirements defined in the list of project requirements, it is clear that not all requirements were met. The lack of met requirements is a result of both lack of time and features missing from the backend.

Although not all requirements were met, the solution is updated, robust, scalable, and well-documented. These are all factors that make the project easier to continue development on. The following resulted in the client being satisfied with our development.

4.1.2 Refactoring

Refactoring the codebase resulted in higher cohesion and lower coupling, as opposed to the early iteration which the team took over. Core functionality which would be used in multiple parts of the code has now been moved separately from

the part of the code which is responsible for displaying the DOM. API calls to specific parts of the REST endpoints are now defined and no longer dangling with new definitions in different parts of the codebase.

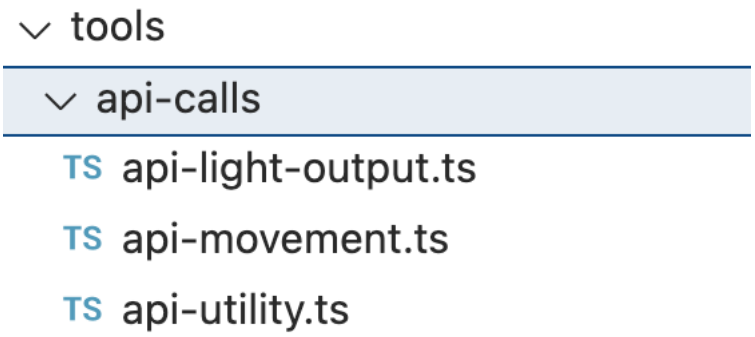


Figure 4.1.2: API separated within the file structure

Before embarking on the development of new features and components, the team prioritized refactoring the existing codebase. This decision aimed to ensure a thorough understanding of the logic and reasoning behind the original design choices. The team meticulously analyzed the codebase, applying industry best practices to enhance its quality and documenting the processes involved.

When refactoring, the team quickly identified that the existing code was outdated and required a significant overhaul. The solution was using outdated versions of NodeJS and React. Class component states were used instead of the more efficient and simpler state hooks with functional components. Furthermore, the team preferred to use TypeScript instead of JavaScript. Refactoring played a crucial role in the initiation of the project and took much longer than anticipated, with a total of 51 hours spent across three sprints to upgrade, switch to TypeScript, and refactor from class component states to state hooks.

However, during the sprint planning process, the team underestimated the amount of time the refactoring would require. This experience highlighted a significant challenge in project management with Scrum - that things often do not go as planned, and careful sprint planning is necessary to avoid delays and ensure project success in the future. This would help shape how the team would plan their upcoming sprints, and would later lead to the introduction of story points in Jira.

In addition to addressing "dirty code", the team also tackled "code smells" during the refactoring process (ref. 2.1.4). These are indicators of deeper issues within the code that may not necessarily cause problems but could indicate sub-optimal design or coding practices. To ensure the stability and safety of the refactored codebase, the team conducted tests after each change, verifying that the modifications behaved predictably and did not introduce new issues. This rigorous testing process further contributed to the overall improvement of the software's quality and maintainability. The approach facilitated more predictable and higher-quality software development.

Node and React

When the team would begin to upgrade the node and react versions. It was decided that it would be easier to upgrade node and react versions on their local development environments and poke at the existing project, and then create a new React project using the "create-react-app" TypeScript template.

JavaScript to TypeScript

Using TypeScript the team has set constraints on the definition of a searchlight. JavaScript is known for its "loosely typed" nature, indicating that it has the capability to automatically convert data to the expected data type whenever an operator or statement requires it. This behavior can be malicious to systems that require precision and safety, in this case, a searchlight is a critical component on a ship and must be treated and guarded from unpredictable behavior. The figures below show the result of conversion and refactoring of how a searchlight is represented.

```
class SearchlightInfo extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      axonAddress: "",
      axonMac: "",
      axonStatus: [],
      cameraSettings: [],
      colornetAddress: "",
      dio: "",
      insyncSettings: [],
      location: [],
      name: "",
      nodeType: "",
      restrictedAreaSettings: [],
      searchlightStatus: [],
      surveillanceSettings: [],
      sweepSettings: [],
      trackingEnabled: "",
      upsideDownEnabled: "",
    };
```

Figure 4.1.3: Instance of a searchlight being defined in JavaScript before refactoring

```typescript
export interface Searchlight {
  axonAddress: number;
  axonMac: string;
  axonStatus: AxonStatus;
  cameraSettings: CameraSettings;
  colornetAddress: string;
  dio: number;
  insyncSettings: InsyncSettings;
  location: Location;
  name: string;
  nodeType: number;
  restrictedAreaSettings: RestrictedAreaSettings;
  searchlightStatus: SearchlightStatus;
  surveillanceSettings: SurveillanceSettings;
  sweepSettings: SweepSettings;
  trackingEnabled: boolean;
  upsideDownEnabled: boolean;
}
```

Figure 4.1.4: Definition of a searchlight using TypeScript after refactoring

Class Component State to State Hooks

A big part of refactoring from an older version of react to a new one, is that of refactoring from class component states to state hooks. State hooks provide a simpler and more concise syntax compared to class components, by letting developers manage state directly within functional components, eliminating the need for constructor functions and binding methods. The team handled this refactoring in a systematic way (ref. 2.1.3). This approach did not work for all components, however, and some needed to be handled with more care or even recreated completely. This resulted in functional programming, instead of object-oriented.

```
class Searchlight extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      hAngle: 0,
      vAngle: 0,
      leftLightStatus: 0,
      leftLightUsage: 0,
      rightLightStatus: 0,
      rightLightUsage: 0,
      isDragged: "",
    };

    this.passHorizontal = this.passHorizontal.bind(this);
    this.passVertical = this.passVertical.bind(this);
    this.handleChange = this.handleChange.bind(this);
    this.checked = this.checked.bind(this);
    this.getNodeInfo = this.getNodeInfo.bind(this);
    this.lightOff = this.lightOff.bind(this);
    this.lightOn = this.lightOn.bind(this);
    this.handleIsDragged = (isDraggedState) => {
      this.setState({ isDragged: isDraggedState });
    };
  }

  componentDidMount() {
    this.getNodeInfo();
    this.interval = setInterval(() => this.getNodeInfo(), 5000);
  }
  componentWillUnmount() {
    clearInterval(this.interval);
  }
```

```
export default function SearchLightContainer() {
  const [searchlights, setSearchlights] = useState<Searchlight[]>([]);

  // Gets the searchlights from the backend
  useEffect(() => {
    const getSearchlightsAndUpdate = async () => {
      const searchlights = await searchlightService.getSearchLights();
      setSearchlights(searchlights);
    };

    getSearchlightsAndUpdate();
    let interval = setInterval(() => {
      getSearchlightsAndUpdate();
    }, 3000);

```

Figure 4.1.5: Searchlight React Component Pre-Refactor and Post-Refactor

CSS

When the team started the refactoring process there was no apparent structure to the styling, as much of the styling was inside of a single style sheet, and there did not seem to be a systematic naming convention for the class names. When refactoring started, the single stylesheet was split into multiple ones with names corresponding to the component it was manipulating. After this job was done, a global stylesheet was established. This sheet would contain global styles (for buttons, etc.), unit resets, and variables that were set according to rules set in the design guidelines. Additionally, a set of IDs would be set in this sheet, these would represent the style of different content-related components, like content

cards, wrappers, and headers. This would be changed in the future due to the nature of ids.

When establishing a structure and standard for styling, it was established that the project was to contain a styling folder that contained style sheets for each component and page, this would prove efficient for scalability and navigation. These sheets were to be named the same as the file they were manipulating. With this established, a naming convention was established. Class names should start with the name of the component it wishes to manipulate followed by a hyphen ("-") and a description of the part of the component that should be manipulated, according to standard practices within the industry [35].

File structure

During the refactoring process, the team decided to revamp the entire source folder of the initial solution. Originally the source folder contained two folders, one for images and one for components, whereas, in the component folder, both pages and components were located. It was decided early to split the source code into folders for assets, components, error handling, model, pages, styles and tools.

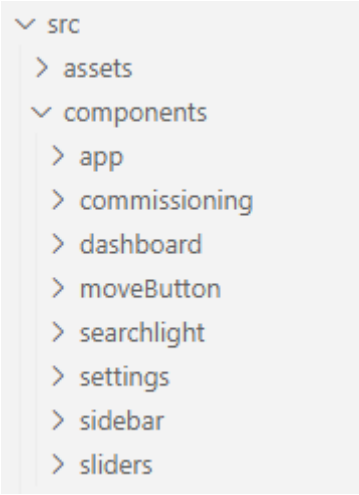


Figure 4.1.6: Initial file structure

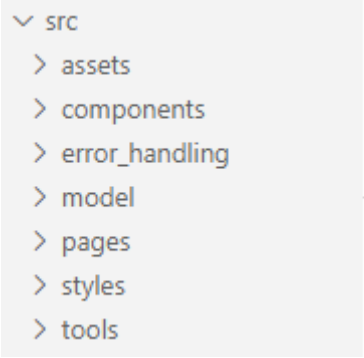


Figure 4.1.7: Refactored file structure

When the project becomes more complex, it becomes more difficult to navigate, this was done in an attempt to keep order and to make scalability easier.

Summary

All things considered, the refactoring process was to be carried out systematically, with the team focusing on the core functionality and streamlining the code design for simplicity and maintainability. Any "dirty code" result of shortcuts, inexperience, or tight deadlines was then identified and replaced with cleaner, more comprehensible code until it was deemed as "clean code".

4.1.3 Implemented functionality

Page overview

Figure 4.1.8 gives an overview of the entirety of the web application. On the launch of the application, the dashboard loads by showing you the connected searchlight and the corresponding light controllers (fig. 4.1.11). This is a scrollable page that dynamically adds a controller for every consecutive searchlight.

On the left of the figure, it shows the sidebar, consisting of the Unity Hub headline, and the following buttons for the different pages: Dashboard, Morse, Statistics, Commissioning, and Settings.

The prior pages will be presented below.

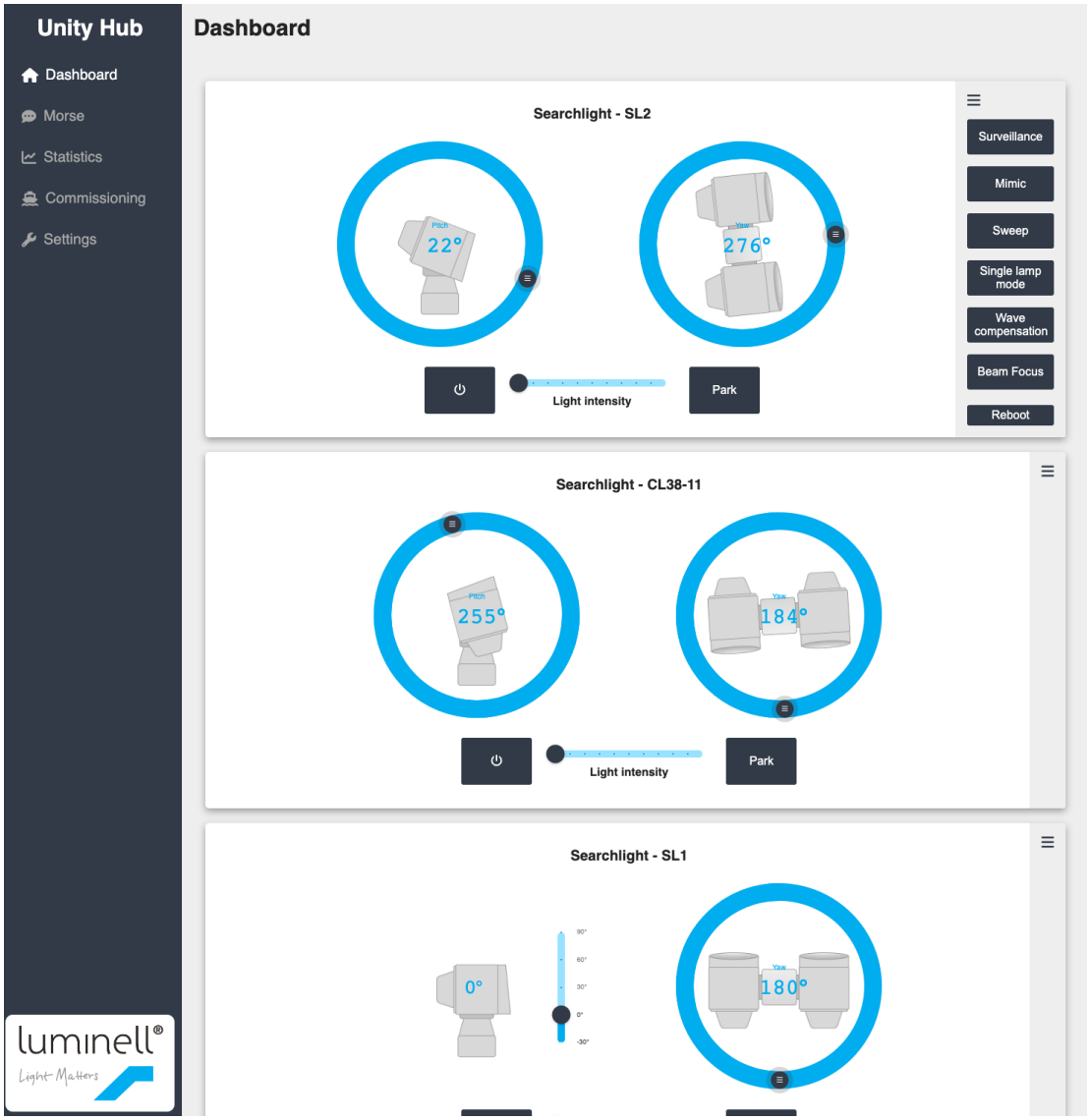


Figure 4.1.8: Dashboard / sidebar

Light Controller

The Light Controller, aptly named, serves as the functional controller for the searchlights. The device features an array of controls including an on/off button, an intensity adjuster, and a park button. Additionally, the controller incorporates adjustable sliders, whose functionality varies based on the specific light under control. As introduced in the method chapter, certain searchlights have a restricted range of motion (ref. 3.1.6); a factor that was thoughtfully considered during the creation of these controllers.

The Light Controller comes in two distinct models. The first model allows unrestricted movement (fig. 4.1.9), equipped with a pair of circular controllers that offer comprehensive 360-degree control in both the horizontal and vertical planes.

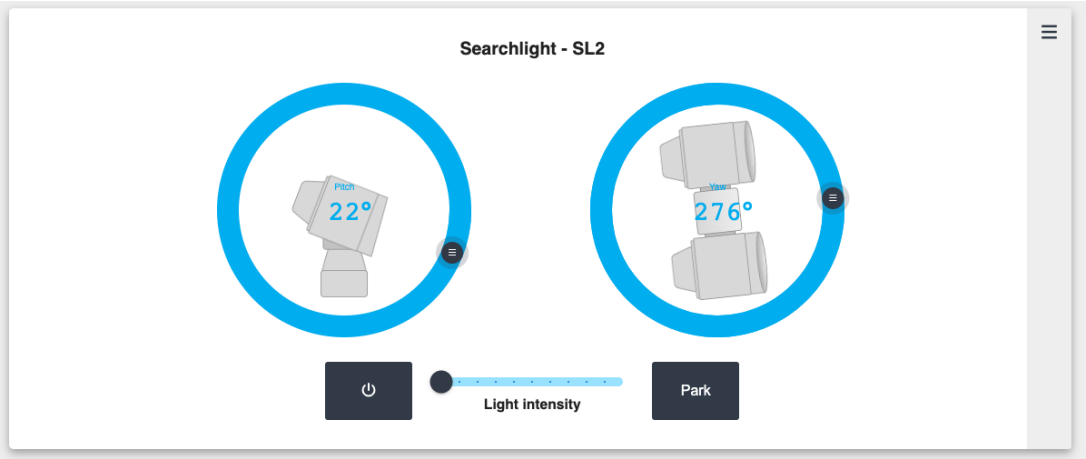


Figure 4.1.9: Light controller SL2

The second variant maintains the same circular controller for horizontal motion but employs a different mechanism for vertical movements. This adaptation is due to the SL1 searchlight's limited movement range of 120 degrees, hence the inclusion of a vertical adjustment slider (fig. 4.1.10).

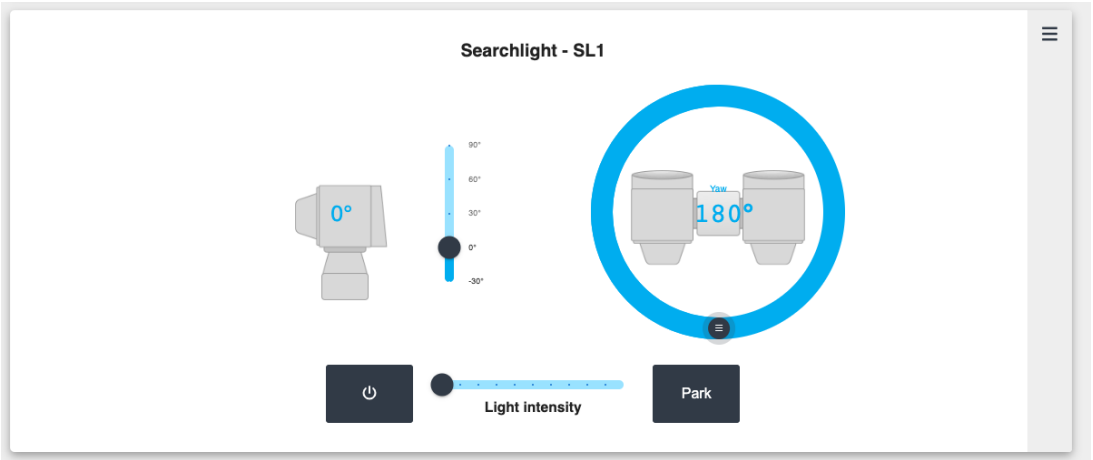


Figure 4.1.10: Light controller SL1

Both controller models incorporate an icon in the top right corner resembling a slide drawer. Upon pressing, this expands to unveil the additional functionalities of a searchlight. The available options in the menu consist of surveillance, mimic, sweep, single lamp mode, wave compensation, beam focus, and reboot.

Figure 4.1.11: Light controller SL2 with menu open

The CL(-38) controller is the same as the SL2 controller, except, this controller should not have the option to adjust light intensity. The reason for not fixing this problem in time for submission is that there needed to be a way to distinguish between SL2 and CL models. This fix to the backend was applied at the end of the project, with no time to implement it in the web UI.

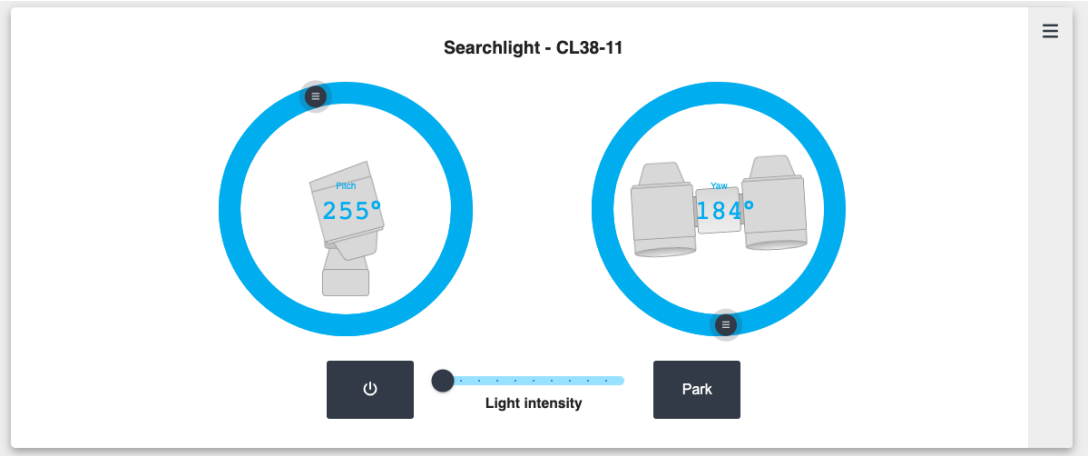


Figure 4.1.12: Light controller for the CL models

Morse Module

In the event of electronic communication failure for vessels traversing the open sea, alternative methods of communication with other ships or land become increasingly essential. One such approach involves the transmission of Morse code via searchlights. The figure below illustrates two primary functionalities incorporated within this system: a text-to-Morse code module and a Morse code-to-text module. These modules were developed after the theory of Morse code (ref. 2.2).

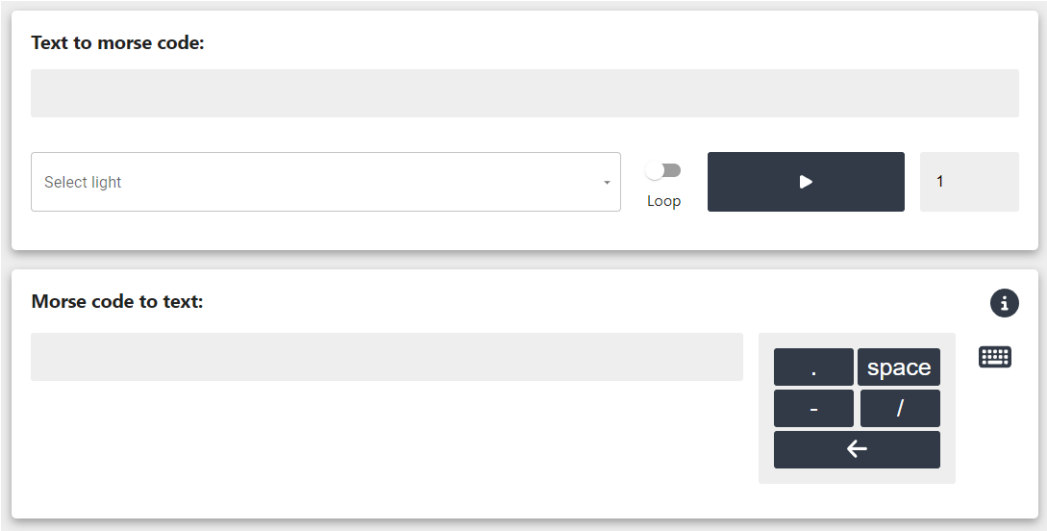


Figure 4.1.13: Morse module

The text-to-Morse code module is specifically designed to facilitate the conversion of desired textual information into Morse code for transmission through the vessel's searchlights. This module allows users to input the required text, select the appropriate searchlight(s) for transmission, determine whether the message should be looped, adjust the playback speed, and ultimately initiate the Morse code transmission.

Conversely, the Morse code-to-text module provides a means of decoding incoming Morse code messages into textual form. This process necessitates manual input from the operator and a basic understanding of Morse code principles. This component facilitates an input field followed to the right by a keyboard button, which reveals a set of buttons to input received Morse code. The empty whitespace below shows the deciphered Morse code in textual form. By utilizing these two modules, vessels can maintain vital lines of communication, even under challenging circumstances.

Statistics

In case of an error concerning a searchlight, the searchlight's error gets added to a designated error array. The error is subsequently displayed in the table as shown below. The error array is structured to assess whether a particular error already exists for a given searchlight. The error implementation consists of sending API requests to check the status of the searchlight. Then there is relatively simple logic: sleep until the searchlight should have done what was requested, and conversely, check if the searchlight status is the same as requested. Then the error creation is handled by a class called "ErrorFactory."

Additionally, a separate window is incorporated to display the burn time for individual searchlights. By providing this information, customers can effectively predict the approximate duration before a bulb replacement is necessary. This feature enhances the overall user experience and contributes to proactive mainte-

nance and improved system reliability.

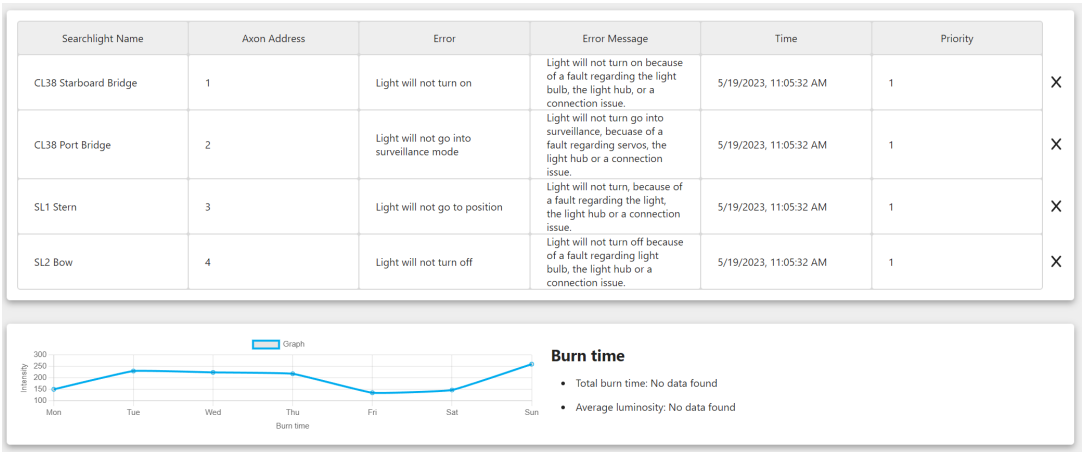


Figure 4.1.14: Statistics page

Settings

The settings page is a page with information about the different searchlights, as well as network settings for changing the Unity Hub's IP, gateway, and subnet mask. Starting on the top of the page; it shows connected devices, starting with the name of the device, address, and node type. This table is there to easily be able to identify how many devices are connected, and which ones. If the user rather wants to check out specific information or change some settings, then the table below is the designated table to look up. Here the user can adjust settings like the position of the searchlight, and modes like tracking and upside-down mode. Lastly the box at the left bottom displays three input boxes for inputting new network settings. This is an advanced setting, requiring some understanding of networks.

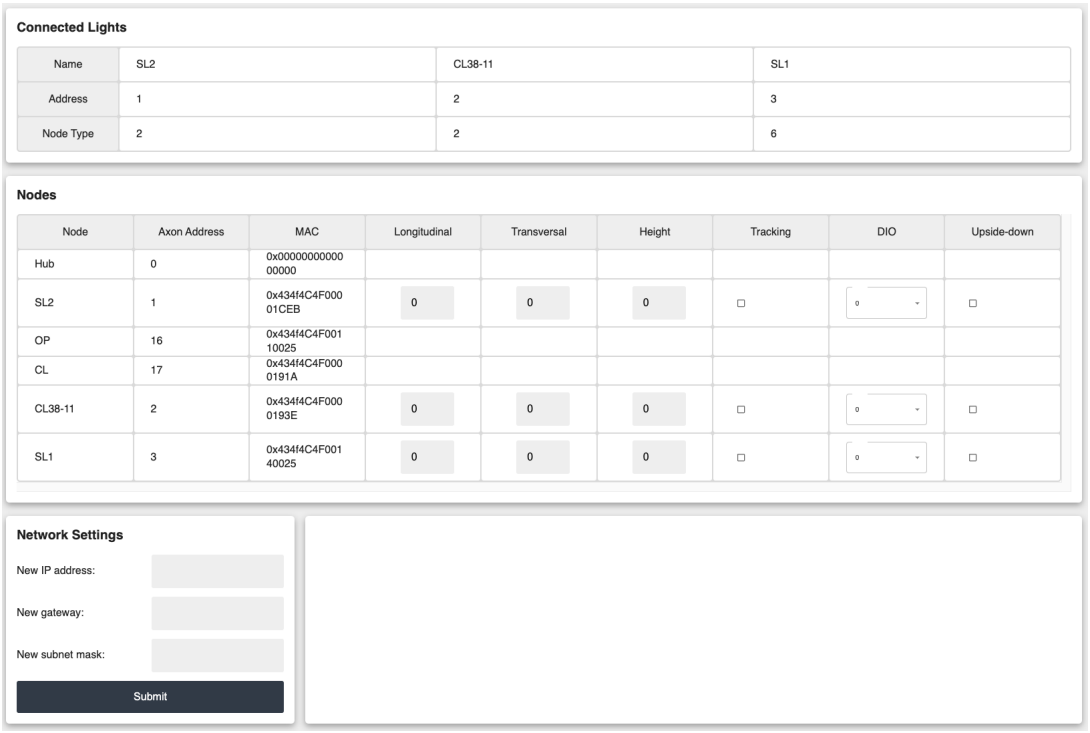


Figure 4.1.15: Settings **Settings page**

Drag and Drop

Drag and drop logic was implemented, but not finished. Within the commissioning page located in the sidebar, users should be able to view an overview of their vessel. Additionally, they should be able to choose the location of their bridge, as well as their searchlights, in order to make the overview fit their specific vessel. The logic behind dragging components into a droppable slot was implemented, however, the team did not find the time to fully implement this feature.

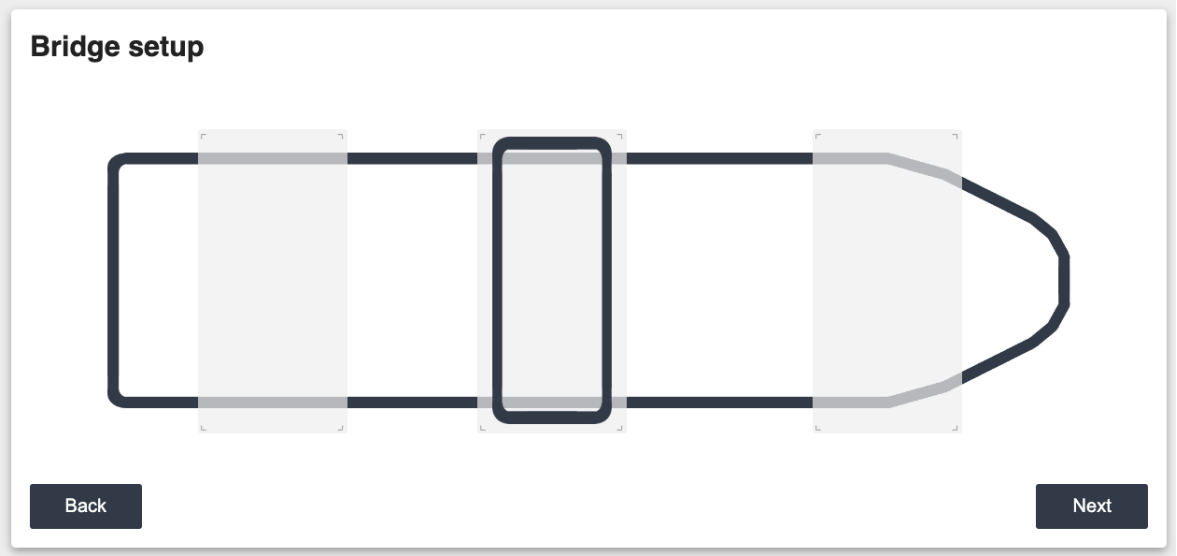


Figure 4.1.16: Bridge dropped onto a boat view

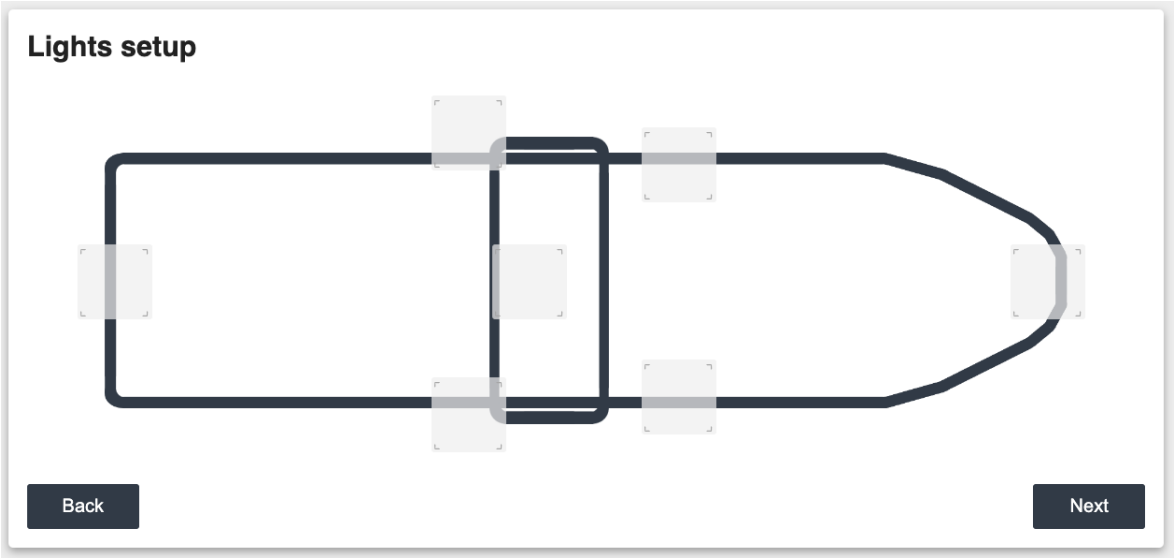


Figure 4.1.17: Slots where lights can be dropped

4.2 Frontend Architecture and Technologies

4.2.1 Figma

The project's wireframes, based on the outlined features, were created using Figma. The wireframes underwent several iterations throughout the project. Initially, rough sketches were presented to the client for feedback. Once the sketches and feedback were approved, the team worked on designing them in greater detail. Throughout the project, the sketches were modified to align better with the website's overall theme and ambiance. This was a favorable decision as it resulted in an improved user experience.

4.2.2 Consistency

To ensure design consistency the team would create and follow a design guideline. This design guideline contained a set of rules that were to be followed when working on the web UI. In this guideline rules for the following points could be found: theme, color scheme, typography, hierarchy, icons, border rounding, and shadows. With the implementation of this guideline, all components, pages, etc. would be consistent and would have the same feel. The rules defined in the design guideline would be added to the global stylesheet as variables.

Colors

The colors defined in the design guideline were "main", "accent" and "client logo color". The main and accent colors were to be used. The main colors with the "client logo color" were used for elements that should be more noticeable. The colors were decided from the existing solution, as well as from Luminell's logo.

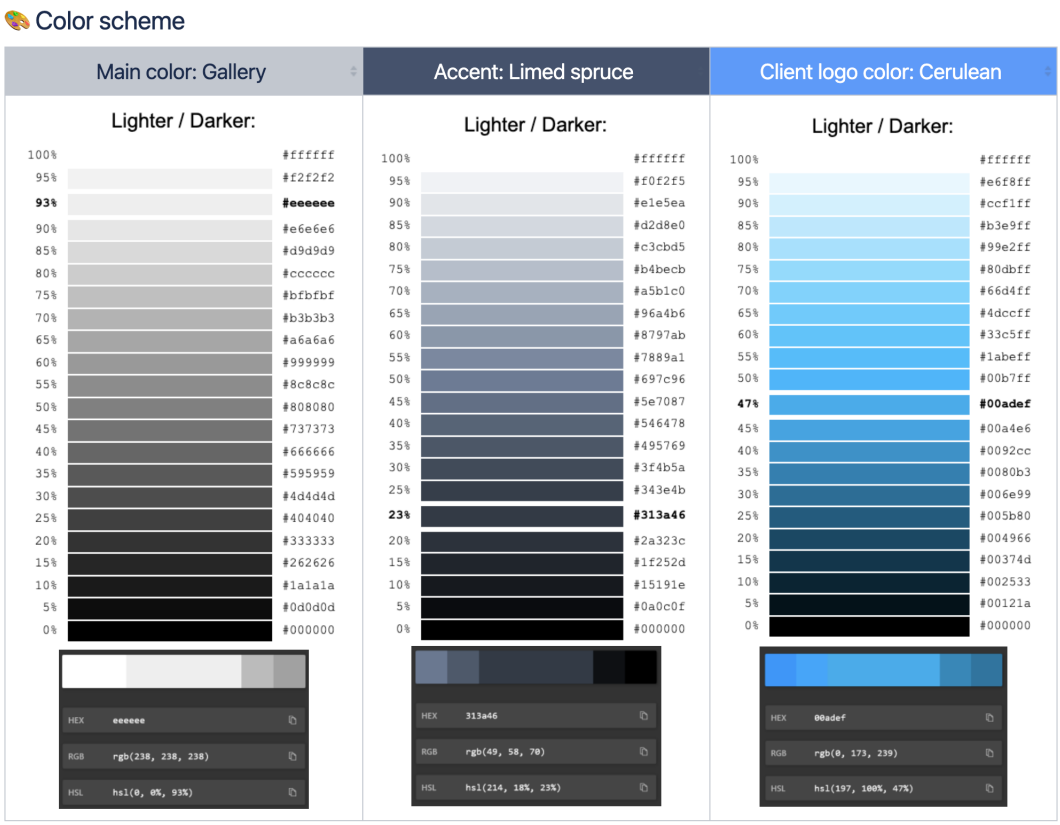


Figure 4.2.1: Design guideline colors

Typography

The typography section defined every aspect related to typography. This included font sizes in multiple units, font family, and a set of rules for header text and body text.

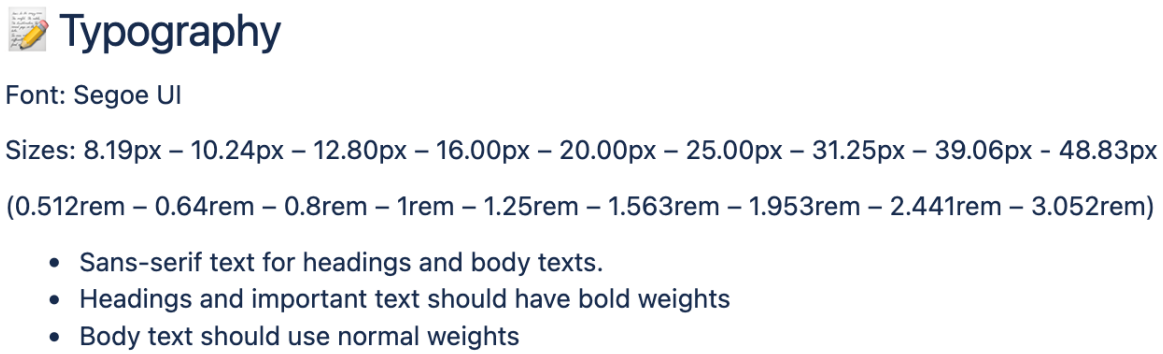


Figure 4.2.2: Design guideline typography rules

Hierarchy

The hierarchy of the front end would be described loosely in this section of the design guideline. Here the main components of the website would be described,

with what they should contain and how they should work. Additionally, this section would contain a set of spacing rules that should be used across the entirety of the front end.

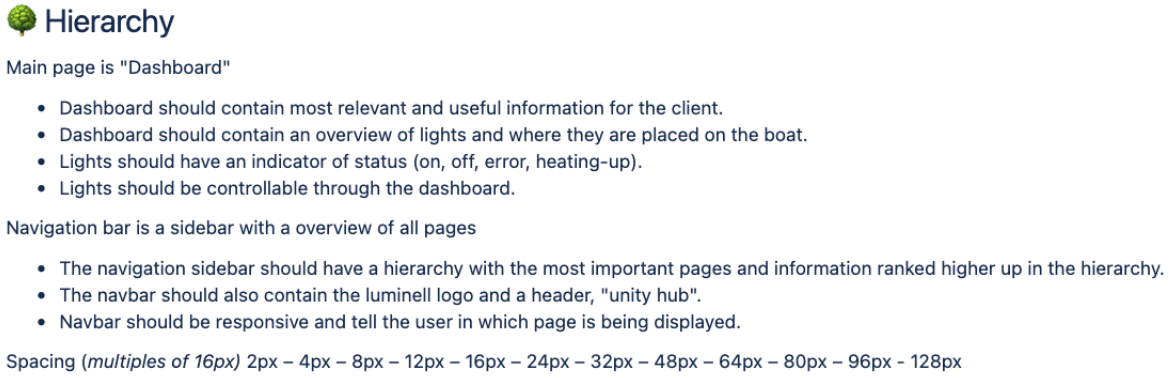


Figure 4.2.3: Design guideline hierarchy rules

4.2.2.1 Icons, border rounding and shadows

Rules for use of border-rounding, shadows as well as icons were defined in the design guidelines as well, these rules were short, and just specified the specifics of the values that should be used on borders and on shadows. Additionally rules for the use of icons, as well as the type of icons were described.

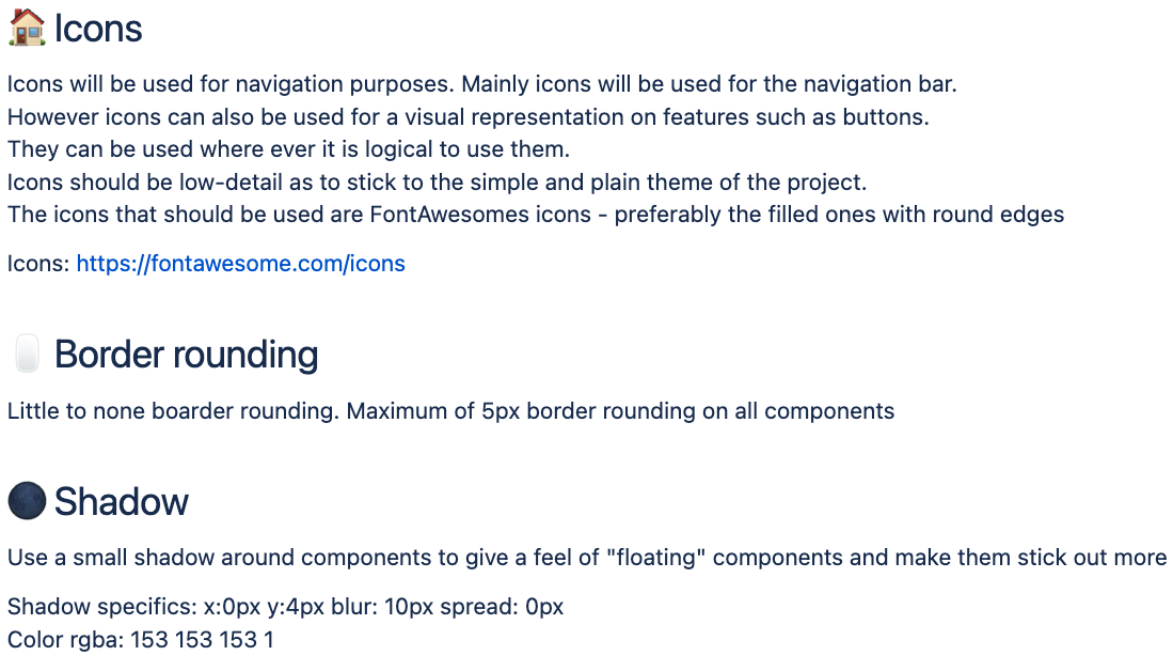


Figure 4.2.4: Design guideline rules for icons, shadows, and border rounding

4.2.3 External libraries

Several external libraries were used to reach the point the project is at today. Libraries were used to provide circular sliders, morse translation, and multiple frontend components like icons, provided by FontAwesome, sliders, and switches, provided by MaterialUI (MUI).

MaterialUI

While most components represented are custom-made, many components were gathered from external libraries such as MaterialUI, and styled to fit the theme of the website. MaterialUI provided the team with responsive and visually appealing components that were customizable. The components used were styled to fit the theme of the website, with respect to the design guideline (ref. 4.2.2).

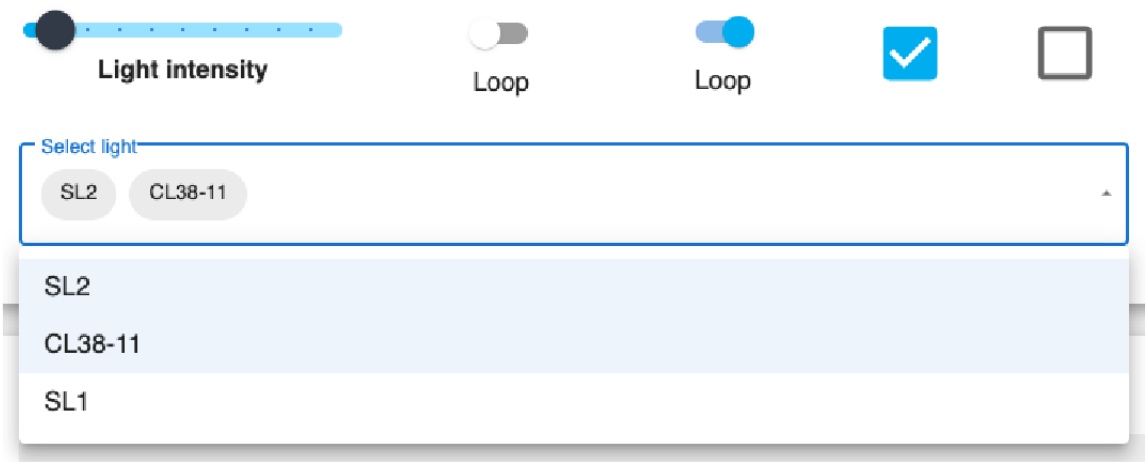


Figure 4.2.5: All MUI components used

Morse-Converter

When developing the Morse module, the team needed logic that could convert text to Morse code and vice versa, as well as logic for transmitting lights according to the Morse code. To save time, it was decided to use an external Morse converting library that offered the required functionalities. The team specifically chose this library for its simplicity and customizability, as it allowed the team to easily convert from one to another, as well as change characters for dots, dashes, spaces, separators, and unknown characters.

```
const morse = require("morse-converter");

const encoded = morse.encode("Morse");
// -- --- .-. ... .

const decoded = morse.decode("-.-. --- -.. .");
// CODE

const custom = morse.encode("is great! ñ", {
  dot: "o",
  dash: "a",
  space: "_",
  separator: "|",
  unknown: "#"
});
// oo|ooo|_|aao|oao|o|oa|a|aoaoaa|_|#
```

Figure 4.2.6: Simplicity and customizability of the morse-converter

React-circular-slider

The react-circular-slider library is an important part of the light controllers. As lights are able to move 360 degrees in the horizontal direction and some models are able to move 360 degrees in a vertical direction, it was necessary to implement a circular slider that could display the current degrees of the light, in terms of its point of origin. To implement this feature a circular slider library was used and styled to fit the theme given in the design guidelines.

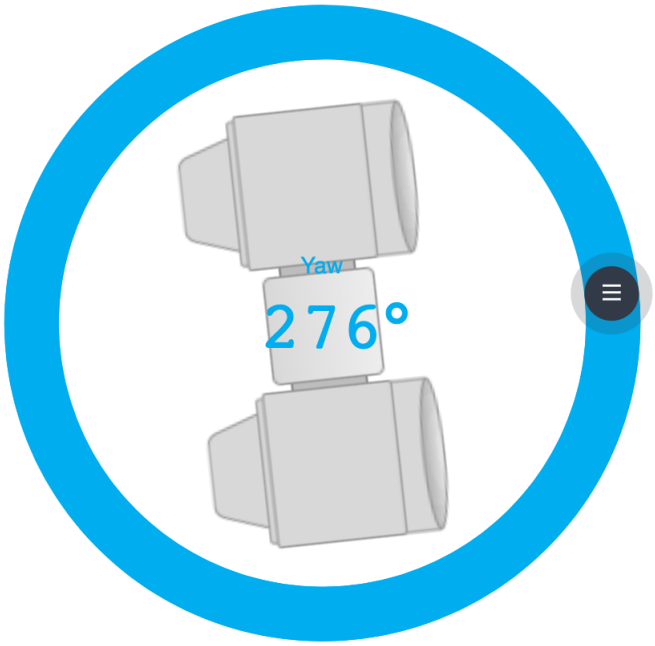


Figure 4.2.7: Circular slider for moving horizontally

FontAwesome

FontAweomse introduced many icons to the project. It was used to save time so that the team would not need to draw icons themselves. It was decided that FontAwesome's classic solid icons would be used, this was because they fitted best with the theme of the website.

Lodash

The Lodash library was used primarily for its Debounce and Throttle functions. The functions implement techniques to control how many times we allow a function to be executed over time.

ChartJS

ChartJS was used to save time when creating a chart component for displaying information about searchlights over time. It was chosen for its simplistic way of implementing charts and for its versatile chart types.

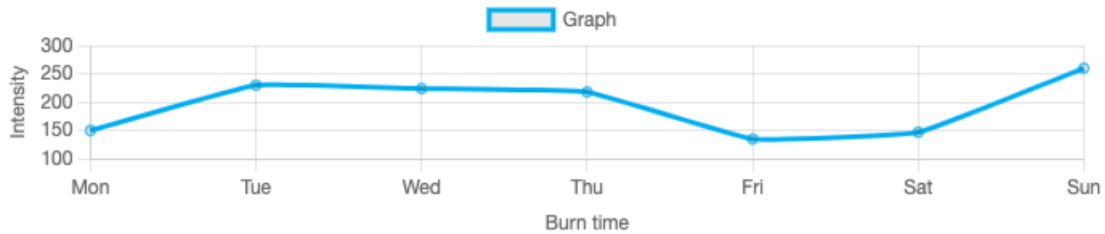


Figure 4.2.8: Chart using ChartJS

Axios

The Axios library was used to handle all the HTTP requests. The tool was chosen for its promise-based HTTP client and easy-to-use interface. Even though the project did not have extraordinarily complicated requests,

react-dnd

React-dnd was used to ease the implementation of a drag-and-drop to an unfinished and unreleased iteration of the commissioning page. The team managed to implement draggable and droppable components.

4.3 Administrative results

4.3.1 Collaboration

For collaboration, the team used tools such as Confluence and Jira to keep track of the project's administrative and development aspects, as mentioned in the method section 3.2.5. Confluence and Jira proved to be powerful tools for collaboration among team members, supervisor, and client. The supervisor Styve (Scrum Master), had access to the project space and could follow the team's progression, and give well-suited advice. The client also had access to the space and could add user stories and prioritize issues based on the need of the different functionalities. The benefits of having the administrative documents in Confluence and issue tracking, with time writing, aided the team in their collaboration. Story points in Jira were another collaboration tool that the team employed some weeks into the project; this assisted the group in improving time estimation.

Aside from the collaboration tools, the team employed co-working practices to overcome issues. The group also employed mandatory on-site working days to improve collaboration and communication; this proved successful since code collaboration seemed harder on platforms like group calls.

4.3.2 Confluence

Confluence was used to create a general overview and to organize the project. Supervisor Styve provided the team with a Confluence space which was set up in the planning phase of the project. Styve has been very helpful and strict with the use of Confluence to ensure the good and correct use of the platform. The project space contains general information, project files, decision logs, and product requirements given to the team by the client. This space also contains meeting notes from sprint reviews, retrospectives, and status reports.

4.3.3 Jira

In Jira, the backlog was filled with issues during the planning phase and during the lifetime of the project. These issues could be tagged as epics, story points, tasks,

improvements, or bugs. These issues were then given a summary, description, version tag, priority, and story point. The description included clear and specific criteria that determine when the issue is considered completed. Additionally, all descriptions should contain a statement that reads "this issue is complete when..." to provide a clear indication of the completion status. Story points are just fictive numbers, representing a unit that is defined by the team. In this project this unit represented hours. This means that the story points were estimated hours an issue would use to be complete. Priority was set to medium by default. However, client representatives can update and set the priority in the backlog. Issues that were higher up in the backlog were considered more important, and their priority would be set accordingly.

4.3.4 Scrum

Using Scrum, the team has worked agile and structurally throughout the project's lifetime. By following the framework, the team has had regular sprint reviews, meetings, retrospectives, and status reports that have been documented. The Scrum roles include a team, product owner, and Scrum Master. The product owner for this project has been the client, Luminell Norway AS, and their representative, Frode Kolgrov. Kolgrov has been very active during the project, giving feedback and essential to prioritizing issues within the project's backlog. The project supervisor, Arne Styve, was the Scrum Master, providing input on how the team practices Scrum and its workflows.

Sprint

The sprint duration was established during the project's initiation in a startup meeting with the client representative and the Scrum Master (ref. 3.2.5). It was agreed that sprints would last for one week, and at the end of each sprint, a meeting would be held with all parties involved to discuss the latest sprint. This frequent feedback loop increased team collaboration, communication, and productivity as potential issues were identified and addressed more quickly. Additionally, with more opportunities to receive feedback and adjust plans, the team was better equipped to respond to project requirements or priorities changes. Every alternate sprint would entail an additional meeting with the team and the supervisor/Scrum Master to discuss Scrum theory and practice, ensuring the correct use of the methodology. By having more frequent sprints, the team had more opportunities to learn and improve Scrum practices through regular retrospectives.

Daily stand-up

Daily stand-ups were held standing-up every day, orally at the campus or through calls on Discord if members could not attend physically. Initially, stand-ups were held at a point in time when all team members saw fit; however, this raised the issue of forgetting to conduct the meetings. To fix this, it was decided to have the daily stand-up meetings right before lunch breaks (11:45). This resulted in much fewer deviations. Additionally, a discord chat channel was created where each team member would shortly write what they discussed on the stand-up, marked

with a date. Stand-up meetings were held standing up.

During the meetings, team members could request feedback, assistance, pair programming, and tips. In extreme cases, when a member was unable to resolve a problem, they could also request that another member assume responsibility for the task.

Scrum poker

The team utilized Scrum Poker to estimate the time needed for issue completion. They employed a mobile app with cards containing numerical values representing hours. Team members would open the app and select the card representing the hours they believe the issue would require completion. Once everyone had chosen their card, they would reveal them simultaneously. The final estimated time would then be determined by taking the average values on the cards.

Meetings

As previously stated, meetings were held at the conclusion of each sprint. These meetings took place at either the NTNU or Luminell's office, with Frode Kolgrov attending whenever possible, either in person or via Teams. The team generated meeting notes to document the feedback provided by both the product owner and supervisor. While the meetings were initially planned for a duration of thirty minutes, they were extended when necessary to discuss significant features or challenges encountered by the team.

Meeting with client

Meetings with the client Kolgrov were organized every Friday at 9:30 AM. The scope of the meetings was to show an overview of the progress that the team has done. Features would be unveiled and a demo would be shown. Thereon feedback from the client would be noted and added as an action item in the Confluence meeting notes. Meeting notes were taken by Eide. The meeting notes are comprised of a date, attendees of the meetings, goals for the meeting, discussion items, and action items. The client is technically knowledgeable of the product, this project is based upon and would provide insight into the documentation and API. Discussions around technical challenges would occur at the end of the meetings so that ideas and solutions could arise. Some meetings would last for well over an hour due to the accumulation of feedback and discussions of the searchlights.

4.4 Version control

Git was used for version control during the project. The team utilized parts of the GitFlow model (ref. 2.6.1) to structure our use of version control. A development branch was used to keep track of the project's history and from it, feature branches were created. These feature branches were named after their corresponding Jira issues. When the features these branches were producing were complete, a merge

request into the development branch would be created and executed as detailed in the quality assurance section in Chapter 3 (ref. 3.2.5). While not used by the team directly, the master branch contained the code for the initial version of the project. The release branch was not used due to an underestimation of the time required to implement the features in the first release and as such the features for the subsequent releases were also not completed in time. Due to this lack of releases, no merges were pushed to the master branch, and due to the lack of releases and merges to the master branch hotfix branches were not necessary.

4.5 Constraints

4.5.1 Obstacles

Overcommitment

Time proved to be a significant obstacle. Initially, the team planned sprints based on what they believed they could accomplish during each iteration. However, this approach resulted in an overestimation of their capabilities as issues often persisted across multiple sprints. When this happened it lead to a delay in the completion of tasks and subsequently the overall project.

Backend

The backend turned out to be a big constraint in terms of time. Although the team had the freedom to work with the backend as needed, it was not originally part of the project scope. The backend contained the REST API and all its endpoints and was documented in the client's repository, making it readily accessible to the team.

However, during the course of the project, the team discovered that the backend posed a significant constraint in terms of time. While testing the system, they noticed that the backend did not always function as described in the documentation. For example, when requesting the current mode of the searchlights, the team found that the expected values were not returned. Upon investigation, it became apparent that the documentation specified the modes as "string" values, whereas in reality, each mode was assigned a numerical value. These numerical values were not defined in the documentation, requiring the team to refactor existing code to accommodate them. Additionally, the team needed to identify which numerical values corresponded to which modes. To address this issue, the team communicated the problem to the client, who subsequently updated the documentation to include the numerical mappings for each mode.

Another issue that arose with the backend was the omission of certain functionalities from the documentation. While most of the backend functionalities were well described, some were completely left out, including those that were critical to the initial solution. As a result, the team had to search through old code to find the missing functionalities, which were not documented in the standard format followed by the rest of the endpoints.

A further challenge that the team faced was related to the missing "model" of the searchlights. Since different searchlights operate differently, it was important to distinguish between them to avoid using them interchangeably. Specifically, there were two main types of searchlights, LED and HMI, with significant differences in their functionality. LED lights could be turned on and off instantly and their intensity could be changed, while HMI lights required time to power up and were not dimmable, but were more powerful. It was therefore critical to identify the model of each searchlight to ensure that HMI lights were not treated as LED lights, which could cause the HMI bulb to burn out. Although the documentation indicated that the "model" was included with each searchlight, it was not actually present, rendering it impossible for the team to distinguish between the models and meet all the requirements.

Overall, these challenges related to the backend of the project significantly impacted the team's progress and required considerable effort to overcome.

4.5.2 Time estimation

Time estimation began during the pre-project planning and continued throughout the project within the sprint planning. The overall time estimation was successful, with all sprints concluded without significant alterations and the majority of them finishing without excessive leftover tasks. Having an extra subject that spanned the first three days of every week, made it harder to accurately gauge the volume of work the team could accomplish within a week. However, the biggest issue with time estimation arose from the team being "time optimists" and underestimating the time required for some tasks, especially the refactoring and user story 29 regarding error messages. To address this, story points were introduced from sprint 8 to assist the team in better-assessing task time requirements. The empirical process was also employed to mitigate this, resulting in some sprints with fewer tasks to allow the team to concentrate on resolving the challenging tasks. This is reflected in the fact that most user stories were completed within the predicted time frame, taking between one to three sprints. Furthermore, the team could have improved by promptly removing deferred issues from the sprint backlog.

CHAPTER

FIVE

DISCUSSION

5.1 General discussion

The team is happy with their result. the final solution is a refactored and upgraded/updated version of the initial solution, with added features. Although the solution is considered good and well-built, there is the issue of not meeting all requirements given in the project requirement specification. Most of the more important requirements were met, such as updating node and react versions, as well as refactoring. However, there is the case of missing functionality, due to lack of time within the team and due to the obstacles discussed in the previous chapter (ref. 4.5.1).

5.2 Engineering discussion

5.2.1 Refactoring

One of the major hurdles for this project was understanding the design choices that have been done by the previous developers. From the team's perspective, the codebase was a mess, code duplication, using the same variable in multiple parts of the code, not reusing components, and unnecessary updates of components were among the issues the team had found with the codebase. This made it extremely difficult for the team members to start coding and precious time was spent in figuring out the structure.

From classes to use-state hooks

The project was initially started in 2019 using an older version of React. At that time class components were still commonly being used, currently, this is considered as a legacy API that no longer receives updates. Class components are still supported by React, but the maintainers and developers of the library do not recommend using them in new code. It is recommended to define components

as functions instead of classes. The team still decided to pursue the challenge of migrating the components from a class to a function using use-state hooks. This proved to be quite effective in learning and understanding how the previous codebase was written and what it actually did. Moreover, it simplified the codebase to be more concise without requiring an unnecessary boilerplate. It is only a matter of time until class components will no longer be supported, as use-state hooks are much better in terms of performance, readability, and longer lifetime support.

Challenges

During the refactoring phase, the team did not have a searchlight available from week 8 through week 9, which slowed down the development and refactoring. During this time a mock of a searchlight was made to facilitate the development. Although the behaviour was not completely mocked the the core functionality was the same. This mock was drafted and coded quickly in the programming language Rust, which one of the previous members was familiar with, mitigating downtime. Lack of documentation and poor API documentation has affected how fast the team was able to code. The code inherited was lacking or had poor documentation of its components and logic.

Refactoring benefits for product owner

The product can be considered as up to date, mitigating vulnerabilities, exploitations, and bad practices within the React library. Readability has increased and the time to code has been dramatically decreased. New developers can have a much clearer overview. This decreases time and costs for the product owner if further development is to be continued.

Value of refactoring

While this practice can sometimes be viewed skeptically, given that it doesn't directly contribute to the introduction of new features or functionality, the value of refactoring lies in its indirect benefits such as improvement of the health of the codebase, enhances developer productivity, reduces errors, and ensures the software's long-term sustainability and adaptability. The team deemed the codebase hard to work on, as it was hard to expand on. The high coupling made it a substantial task for the developer to process the flow at which events would happen. It's an investment that pays off in the long run and should be a consistent part of any development strategy. All in all the team is very happy with the result of refactoring.

5.2.2 Page overview

As mentioned earlier in the thesis, Luminell had an existing web app with a basic layout. The team was free to redesign the web app as they wanted, but since the current solution had simplistic and easy-to-understand qualities, they just improved upon that design. The design guidelines were followed consciously (see dashboard figure 4.1.8 and design guidelines figure 4.2.1). The team utilized

contrast to distinguish the importance of components, and therefore improve the visibility across the application. Using the icons in the sidebar is an effort to abide by some design principles, like utilizing visual elements to aid the understanding of the menu item and make an aesthetically pleasing design. The team chose to separate the different sections using colors and shadows to represent cards, as seen in fig 4.1.8.

The team is very happy with the page overview. The consistency across the entirety of the web application is done well, resulting in a tidy look. Furthermore, the use of colors to indicate important content such as buttons and sliders made it easier to see what is more important for the users.

5.2.3 Light Controller

The implementation of the light controller was a crucial aspect of the project. Considerable effort was put in to ensure that the controller functioned as intended, by the team. The controller computes the desired input provided by the user and translates it into the output angle for the specific searchlight. In doing so, the controller must account for different searchlight types, such as SL1 or others, in order to calculate the accurate angle for the lights. Moreover, the controller refreshes every three seconds to obtain the updated position for the searchlight.

Upon reflection, this solution exhibits limitations, as the controller appears unresponsive and uneven in performance. Alternative approaches, such as utilizing web sockets instead of API calls, may have yielded improved outcomes. This would be far outside the team's scope and would be a big overhaul of Luminell's codebase. Nonetheless, the team exerted their utmost effort to optimize the controller's accuracy within the constraints of the given framework.

The decision to minimize the majority of buttons for the searchlight was made deliberately to emphasize only the most crucial functions and avoid unnecessary complexity in the controller interface. The primary controller should incorporate sliders for movement, an on/off toggle, a light intensity slider, and a park option(as discussed in the results). Upon a single click, all other buttons for the searchlight are revealed on the right side. This approach proved to be successful, as evidenced by the client's favorable reception of the team's implementation.

5.2.4 Morse Module

The Morse module, as described in the results, works as expected. It is a robust system with a logical interface, and the team is very happy with the module. Though there was used a library for the translation from Morse to text, and vice versa, there was an extensive amount of time put in to ensure the team met the product owner's standard.

Despite the team's satisfaction with the Morse module, the question lingers: was this truly a good solution? Overall, the outcome is regarded as robust and of high

quality, with the only drawback being the morse converting library that was utilized. The specific library was initially chosen for its straightforward approach to Morse translation. However, its lack of maintenance posed a problem. The library had not been updated for two years prior to conducting this thesis. Typically, such a prolonged period without updates is viewed unfavorably. Nevertheless, in this case, since Morse code is unlikely to undergo significant changes in the near future, the team did not perceive it as a significant concern.

5.2.5 Statistics

Error messages

The logic for the error message display that the team created, is in the frontend. This is not a good idea when coding because the systems logic should be in the backend and the frontend should only have code for the UI. The reason the error message logic was made in the frontend was that the backend was out of the scope of the bachelor project. This implementation was only added because Luminell wanted to have error handling for its searchlights. This solution is not recommended as it could have security flaws, but this web application will never connect to the wide web, so the security threat is not a big concern. While having the logic in the frontend was not the best solution. The error logic itself was well implemented using Factory Pattern making adding more errors in the future easy.

The issue regarding the error message display is that the team failed to properly implement the communication between the error message logic and the error message display. The team managed to make a component that dynamically displayed the errors, but the passing of information between the files was subpar and could be improved by using other technologies such as Reacts Context API. However, the best solution would be to move the error message logic to the backend and create a database. Then modify the display code to retrieve the errors from the database and refresh periodically.

Due to not having a database, the error messages are not persistent. Though the team could store data on the website(Client-side storage), this could lead to sub-optimal performance, and data storage would be individual for each client, so the team decided not to store the data. When Luminell overhauls the codebase, the database could be easily implemented.

When opening the statistics page, the error message table appears empty. The team decided that the error table was to be visible even though there were no errors. This visibility ensures that the user understands where to look if there is a failure in the system. Moreover, showing a reference to the error table if an error occurs using the light controllers would be an excellent addition to improving the user experience.

5.2.5.1 Burn time

The burn time display's functionality was also limited due to the absence of a database. Initially, the team aimed to develop a display, featuring a graph that would present information regarding burn time, gathered from the database. However, upon discovering that incorporating databases fell outside the project's scope, the team shifted its focus to devising a simple JSX component capable of accepting the desired information as parameters. This approach allowed for a more simplified presentation of the burn time information, albeit with reduced functionality compared to the original design.

In future iterations of the project, the goal is to develop a burn time card file that utilizes the chart card component. This burn time card would feature an interface termed "burn time info", which would contain a date, a data list, and a labels list. The date property would facilitate access to historical burn time data. Additionally, the file would incorporate a function to retrieve data from the database using the date as the key. Upon rendering, this function would be called retrieving the burn time information corresponding to the present.

The team's proposed structure for the collection in the database, intended to store burn time information, is as follows:

- A collection named burn time info contains several date-based collections.
- Date collections each encompassing Burn time log documents.
- Burn time log documents, created each time a light is turned on, turned off, or experiences an intensity change. These documents would include fields for the light's name, its axon address, the intensity level it was set to, and the timestamp of the intensity change.

To facilitate the logging of burn time information, a "log burn time information" method would be implemented, which would be invoked each time a light is turned on, off, or undergoes an intensity change. This method would record the light intensity in the daily data list and register the date in the daily labels list. Additionally, separate methods would be developed for calculating both the average and total burn time based on the information retrieved from the database.

Furthermore, a calendar would be integrated to enable users to access historical burn time data by selecting a specific date. Upon selection the retrieve data function would be invoked using the given date, thereby providing the user with the relevant burn time information.

5.2.6 Settings

The settings page, as seen in figure 4.1.15, is composed of 2 tables of information for the searchlights. As well as the network setting for the Unity Hub, as mentioned in the project requirement spec(1.3) The page is essential to the operators, with the capability to change the position, upside-down mode, and so forth.

This page is finished and considered done, as all the requested functionality has been added. The choice of having two tables, one for displaying connected lights, and one for displaying all nodes with input fields, is considered a good solution. This is due to the fact that the main focus of the application is the controlling of searchlights, hence the connected lights are displayed at the top to display all the lights that are connected to the system. Additionally, the table below displays a table with all nodes (searchlights, operating panels, and Unity Hub). This table is interactive to some degree as it allows users to change the positioning of devices as well as check the "tracking" and "upside-down" attributes of the lights.

5.2.7 Network Settings

The product owner wished for a way to be able to change the network settings on the Unity Hub via the web UI. This is quite hard to achieve using Node.JS as it is considered dangerous and complex to change a system file which directly affects the network interface configuration of the device, as it leads to system instability, loss of inactivity, and security risks. The initial way of creating this feature was to use Node.JS to write directly to the network configuration file by accessing the file system of the Unity-Hub and parsing it and then replace with the newly desired configuration. This would require a great expenditure of time, and so the team searched for other methods to do so. This feature would fall out of the scope, however, it was highly requested so the team decided to use the Rust programming language and leveraged it's process builder standard library to write a service. Upon searching the team found the standard library "std::process::Command" while developing the mock. This allowed us to change said network interface configuration in a safe and easy way by leveraging the existing Linux commands which the team thought was a brilliant way of solving this challenge, but introduced a new dependency. The Unity-Hub must run the Rust service written to be able to change the network settings on the device via the UI. Instead of creating something proprietary and unproven, the team used the existing Linux command feature that every developer or network administrator is familiar with instead, which the team thought of as a great success and solution.

5.3 Administrative discussion

5.3.1 Time estimation

INGA2300 - Engineering Systems Thinking

A major challenge the team encountered when estimating time, was the inclusion of the subject Engineering Systems Thinking. This subject had lectures scheduled on Mondays and Tuesdays. Wednesdays were also the only days the team had access to teaching assistants for help with assignments and team tasks in the subject. Consequently, the team was left with only two working days per week dedicated to the bachelor project. Considering the examination periods, this subject extended up to the 20th of March, thus occupying a substantial portion of the semester. The significant time requirement by this subject made estimating the total vol-

ume of work the team could feasibly accomplish, hard. During the pre-planning phase the team had to attempt to estimate the approximate amount of time that INGA 2300 would require. The limitation of only having two-day sprints made, estimating the volume of work the team could realistically complete, even harder. This posed a particularly challenging issue when attempting to approximate the time required to resolve problems and bugs, and further estimate how these bugs and constraints would impact the planned roadmap.

To minimize the issue of this course occupying too much time, the team could have utilized the Wednesdays more efficiently, by coming to an agreement with the other team members to spend less time on the course on Wednesdays and rather spend time on the bachelor project.

5.3.2 Work estimation

The Error Message and Statistics page increment was a step in the process that the team eventually discovered that we had wrongfully estimated. Consequently, sprints following sprint 6 needed to account for this wrongful estimation. From that point forward the team always considered the time required to complete item WFLUMINELL-29 and its sub-items as well as whether the incompletion of these items would create constraints for other items. To address this issue a conservative approach was adopted, with the team adding slightly fewer tasks than they believed could be completed within the sprint. Additionally, other team members either assisted with item #29 upon completing their tasks or added new tasks to the sprint, or contributed to the report ensuring progress would not be hindered by the problem item. A concrete example of the empirical approach taken to address item #29 is the planning of sprint 13. During this sprint's planning, the team prioritized the completion of item #29 and consequently included fewer items in Sprint 13, with the intention of adding more tasks if the sprint backlog items were completed ahead of schedule. Due to item #29 taking longer than initially estimated, the sprint review often led the team to plan fewer tasks for the subsequent sprint, prioritizing the completion of the problematic issue while avoiding the assumption that it would be resolved quickly.

5.3.3 Motivation and teamwork

Re-delegation of tasks rarely occurred due to the standup meetings. This can be attributed to the meeting's ability to enable other team members to offer quick tips on technology usage or research, as well as allocating time to pair programs allowing tasks to be completed more quickly. As previously mentioned, convening daily to discuss the progress, enabled the team to quickly adapt to changes.

5.3.4 Remote vs on-location work

Two mandatory office days were agreed upon at the beginning of the bachelor project and as the final month approached all five weekdays were mandatory. These mandatory office days further lowered the threshold, regarding requesting

help, as a member could now simply ask for help from their colleague on the other side of the table. Certain problems require more than one teammate and office days made it easier to work together during those days.

CHAPTER

SIX

CONCLUSIONS

6.1 Problem solving

In the process of implementing the proposed system, the team faced a number of challenges due to certain constraints imposed by the backend (ref. 4.5.1). Despite the fact that not all requirements were ultimately achieved, the team demonstrated resourcefulness and adaptability in overcoming numerous obstacles that emerged throughout the project's lifecycle.

One of the main issues was related to inaccuracies and gaps in the backend's documentation. This issue was most apparent in the case of the searchlights' current modes, where the expected string values were not being returned. Instead, they were replaced by numerical values which had not been defined in the existing documentation. To overcome this problem, the team engaged in intensive communication with the client to bring the issue to their attention and seek guidance. As a result, the documentation was updated to include the numerical mappings for each mode.

Another significant challenge was related to missing functionalities from the backend documentation. This issue was resolved by the team by painstakingly going through old code to locate the missing functionalities. Although this task was time-consuming and wasn't originally planned for, the team's determination and collaborative efforts led to the identification and understanding of these functionalities, which were then incorporated into the project.

A further hurdle encountered was related to the missing "model" of the searchlights, which posed a risk of incorrect usage and potential damage to the HMI lights. In response to this, the team spent a significant amount of time devising a solution to distinguish between the searchlight models, ensuring that they would be used correctly and safely.

Through these experiences, the team honed their problem-solving skills, demonstrating the ability to adapt and innovate in the face of unanticipated challenges.

6.2 Our contribution

The team's contribution to this project was substantial and multi-faceted. It was clear from the onset that the project would demand an extensive understanding of various technologies, a knack for problem-solving, and a profound commitment to collaborative work.

The team effectively took on the task of managing and organizing the project, using project management tools such as Confluence and Jira. This ensured clear communication within the team and with the client, facilitating the tracking of the project's progress and management of sprints and tasks.

On the technical front, the team navigated through the challenges imposed by the backend. Despite numerous hitches, including inconsistent and missing documentation, the team demonstrated exceptional problem-solving skills to rectify these issues. This often involved diving deep into the backend code, working closely with the client, and applying a great deal of creativity and persistence.

Moreover, the team actively contributed to identifying and addressing the client's requirements and effectively translating these into a functional system. Despite not being able to meet all of the requirements due to the backend constraints, the team managed to deliver a functional, user-friendly system that added value to the client's operations.

6.3 Further work

While the project has achieved significant milestones, there are several areas of potential further work due to the evolving nature of the client's requirements and the constraints encountered during the project.

Backend Documentation

One of the main challenges faced during this project was the missing and inconsistent backend documentation. Further work could involve a thorough review and update of this documentation to ensure that it accurately reflects the backend's current functionalities and responses. This would facilitate future developments and minimize the time required for debugging and problem-solving.

Differentiating Searchlight Models

The problem of differentiating between the searchlight models was not fully resolved during the project. Future efforts could focus on finding a robust solution to this issue, perhaps by working with the backend team to include the model information in the searchlight data.

Completing Remaining Requirements

Due to the constraints and challenges faced, not all requirements were fulfilled.

Controller

The current state of the controller is good and they work as intended, however, they have a tendency to feel inconsistent, as it received feedback while operating the controller. This results in the sliders moving around under operation.

Further development on this component would include making the controllers feel consistent and making them not receive information and update during them being operated. Additionally, the buttons for toggling functionality are updated upon receiving information from the lights, resulting in a delay in the updating of the button. Further, these buttons should be updated on click.

Commissioning page

The commissioning page is a page that does not have any features. In later iterations of the project, the team would like to finalize the requested features like drag and drop, to be able to customize the user's vessels.

Refactor from id to components

The solution currently uses global IDs to represent content-related components, such as headers, cards, and wrappers. This is not the correct use of IDs as they should be unique across the entire project. Furthermore, these IDs should be removed and swapped with components or class names.

Implement model restrictions

As it stands there is currently no separation between different models of searchlights in the application. This is problematic as LED and HMI lights should be handled differently. Further, a restriction in features should be implemented between the different models. Such as HMI lights not being dimmable and not being able to transmit Morse code.

Mimic, homing, and sweeping

Mimic, homing, and sweeping are all features represented in the list of requirements (1.3). This logic should be implemented, such as defining sector sizes, sweeping locations, and the logic behind lights mimicking others.

CHAPTER

SEVEN

SOCIAL IMPACT

Social aspects:

The user interface for controlling ship searchlights can significantly enhance safety and security onboard, especially at night. It can help avoid accidents and facilitate communication with other ships, increasing social trust and cooperation among vessels.

The searchlights will be more predictable, therefore leading to better maintenance. Improved maintenance will also lead to improved security for the crew on vessels since the operators can, in a higher sense, trust these searchlights to work when a situation occurs (ex., a dangerous encounter with another ship or "man overboard"). This higher trust is because the app reports burn time for the HMI lights, so the crew may have a set of bulbs sent when the vessel is docked.

Environmental aspects:

If the app improved over time and then better allowed for automated or optimal searchlight control, it could save energy, thus contributing to environmental sustainability. Improved sustainability is critical when considering the marine sector's electrification, especially if ships become all-electric, and therefore need to save as much as possible. Today this is not a critical implementation since most vessels still use fossil fuels, with a dynamo charging a battery.

Light pollution is a relevant subject to the project. In a fundamental evaluation, enhancing the user interface does not increase the existing issue of light pollution. This is because the operating panel responsible for managing these lights is already in active operation; thus, the level of light pollution remains constant. The light pollution will be significantly higher for other vessels that may use searchlights with less intuitive control methods.

Economic aspects:

The application can lead to significant cost savings regarding maintenance. When a light bulb change is on order, the planned maintenance could include changing several other components on the feedback from the burn time and error modules. The improved safety and communication capabilities could enhance operational efficiency, potentially reducing delays and downtime, thus positively impacting the ship's productivity.

REFERENCES

[1] Refactoring Guru. *Refactoring*. URL: https://refactoring.guru/refactoring (visited on 05/08/2023).

[2] Wikipedia contributors. *Morse code — Wikipedia, The Free Encyclopedia*. [Online; accessed 05-May-2023]. 2023. URL: https://en.wikipedia.org/wiki/Morse_code.

[3] Wikipedia contributors. *Representational state transfer — Wikipedia, The Free Encyclopedia*. [Online; accessed 20-May-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=1155276189.

[4] Wikipedia contributors. *Object-oriented programming — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-May-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Object-oriented_programming&oldid=1154120068.

[5] Wikipedia contributors. *Functional programming — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-May-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Functional_programming&oldid=1153372920.

[6] Wikipedia contributors. *Event-driven programming — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-May-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Event-driven_programming&oldid=1153047477.

[7] John Hunt and John Hunt. "Abstract factory pattern". In: *Scala Design Patterns: Patterns for Practical Reuse and Design* (2013), pp. 155–161.

[8] Refactoring Guru. *Factory Method*. URL: https://refactoring.guru/design-patterns/factory-method (visited on 05/08/2023).

[9] Atlassian. *GitFlow Workflow*. Apr. 2023. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (visited on 03/16/2023).

[10] GitLab. *Merge requests*. [Online; accessed 20-May-2023]. 2023. URL: https://docs.gitlab.com/ee/user/project/merge_requests/.

[11] GitLab. *What is a code review?* URL: about.gitlab.com/topics/version-control/what-is-code-review (visited on 03/27/2023).

[12] Atlassian. *What is version control?* [Online; accessed 20-May-2023]. URL: https://www.atlassian.com/git/tutorials/what-is-version-control.

[13] interaction foundation. *What is Agile Development?* URL: https://www.interaction-design.org/literature/topics/agile-development (visited on 04/07/2023).

[14] Naveen Kumar Singh. "*Scrum theory, principles, and values*". In: *Elsevier* (Oct. 2021). URL: https://medium.com/agilemania/scrum-theory-principles-and-values-9a19d01c895c.

[15] UAGC Staff Member. "*What is scrum?*" In: (Dec. 2021). URL: https://www.uagc.edu/blog/what-is-scrum.

[16] Hiren Doshi. "*The Three Pillars of Empiricism (Scrum)*". In: (Dec. 2016). URL: https://www.scrum.org/resources/blog/three-pillars-empiricism-scrum.

[17] HTML. *HTML.* [Online; accessed 16-May-2023]. May 2023. URL: https://html.spec.whatwg.org/multipage/.

[18] Bert Bos. *Cascading Style Sheets home page.* [Online; accessed 16-May-2023]. May 2023. URL: https://www.w3.org/Style/CSS/.

[19] Microsoft. *TypeScript is JavaScript with syntax for types.* [Online; accessed 16-May-2023]. 2023. URL: https://www.typescriptlang.org/.

[20] Meta Open Source. *React the library for web and native user interfaces.* [Online; accessed 16-May-2023]. 2023. URL: https://react.dev/.

[21] Rust. *Rust - A language empowering everyone to build reliable and efficient software.* [Online; accessed 16-May-2023]. 2023. URL: https://www.rust-lang.org/.

[22] Figma. *Nothing great is made alone.* [Online; accessed 16-May-2023]. 2023. URL: figma.com.

[23] Postman. *Build APIs together.* [Online; accessed 16-May-2023]. 2023. URL: postman.com.

[24] Git. *git –fast-version-control.* [Online; accessed 16-May-2023]. 2023. URL: https://git-scm.com/.

[25] GitLab. *Software. Faster.* [Online; accessed 16-May-2023]. 2023. URL: about.gitlab.com.

[26] Scrum. *Welcome to the Home of Scrum!* [Online; accessed 16-May-2023]. 2023. URL: scrum.org.

[27] Atlassian. *Accomplish more together.* [Online; accessed 16-May-2023]. 2023. URL: https://www.atlassian.com/software/confluence.

[28] Atlassian. *All your work in one place.* [Online; accessed 16-May-2023]. 2023. URL: https://www.atlassian.com/software/jira/work-management.

[29] Discord. *IMAGINE A PLACE...* [Online; accessed 16-May-2023]. 2023. URL: discord.com.

[30] Dave Gandy. *Take the hassle out of icons in your website.* [Online; accessed 16-May-2023]. 2023. URL: http://fontawesome.io.

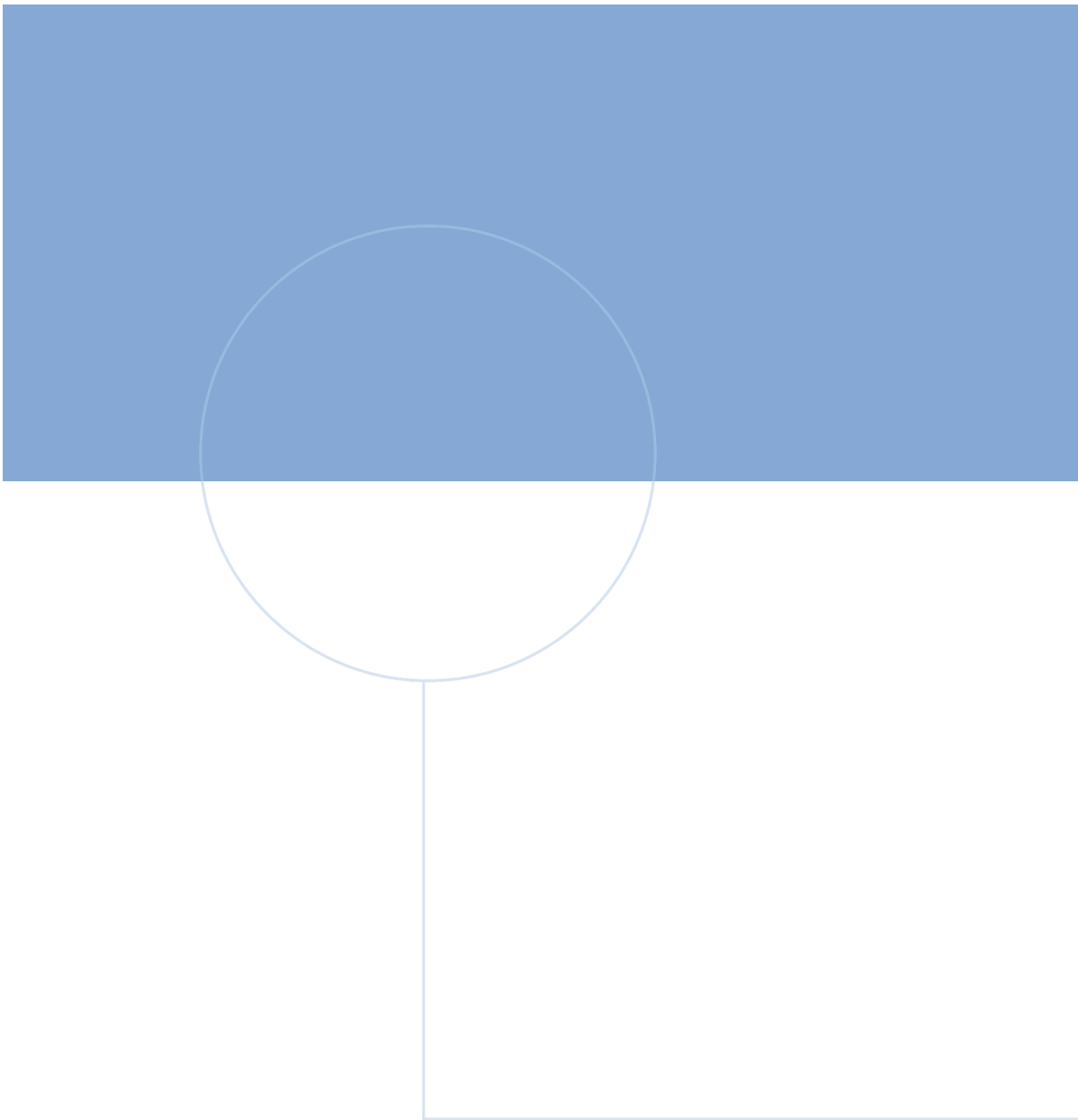
[31] Material UI. *Move faster with intuitive React UI tools.* [Online; accessed 16-May-2023]. 2023. URL: https://mui.com/.

[32] Chart js. *Simple yet flexible JavaScript charting library for the modern web.* [Online; accessed 16-May-2023]. 2023. URL: https://www.chartjs.org/.

[33] React DnD. *Drag and Drop for React.* [Online; accessed 16-May-2023]. 2023. URL: https://react-dnd.github.io/react-dnd/about.

[34] John Jakob "Jake" Sarjeant. *Promise based HTTP client for the browser and node.js.* [Online; accessed 16-May-2023]. 2023. URL: https://axios-http.com/.

[35] MDN Web Docs. *Organizing your CSS.* [Online; accessed 20-May-2023]. URL: https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Organizing.



NTNU

Norwegian University of
Science and Technology