Karl Johan Alvestad
Helene L. Rasmussen
Lasse Raaum
Henrik Rian

OpenBridge Design System Implementation in maritime applications

Bachelor’s thesis in Electrical Engineering - Automation and Robotics
Supervisor: Ottar L. Osen
Co-supervisor: Lars Ivar Hatledal, Eirik Fagerhaug
May 2023

Bachelor’s thesis

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

NTNU
Norwegian University of
Science and Technology

Karl Johan Alvestad
Helene L. Rasmussen
Lasse Raaum
Henrik Rian

OpenBridge Design System Implementation in maritime applications

Bachelor's thesis in Electrical Engineering - Automation and Robotics
Supervisor: Ottar L. Osen
Co-supervisor: Lars Ivar Hatledal, Eirik Fagerhaug
May 2023

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

NTNU
Kunnskap for en bedre verden

Department of ICT and Natural Sciences

IELEA2920 - Bachelorthesis Automation

OpenBridge Design System Implementation in maritime applications

Karl Johan Alvestad (10002)

Helene L. Rasmussen (10025)

Lasse Raaum (10017)

Henrik Rian (10030)

Supervisor: Ottar L. Osen

Supervisor: Lars Ivar Hatledal

Supervisor: Eirik Fagerhaug

May 2023

Preface

This bachelor thesis is written by four students from Electrical Engineering - Automation and Robotics at NTNU Ålesund. The project was conducted in collaboration with Kongsberg Maritime, started in January, and was completed in late May. Our motivation to choose this task came from an interest in the maritime industry, a desire to expand our knowledge about the field, and the opportunity to combine it with what we have learned over the past three years.

We extend our sincere gratitude to the following:

- Kongsberg Maritime for the opportunity to collaborate on this project.
- Our supervisors, Eirik Fagerhaug, Lars Ivar Hatledal, and Ottar L. Osen, for their invaluable advice, constructive feedback, and guidance throughout the duration of this project.
- The staff at the Ocean Industries Concept Lab for their insightful feedback, which proved instrumental in shaping the trajectory of our project.
- Our informants for the research and feedback to the project.
- Our family and friends for their support and valuable feedback during the course of this project.

Abstract

This thesis aims to evaluate the potential of the OpenBridge Design System (ODES) as a design framework for maritime applications, both current and future. The task was conducted as an assignment by Kongsberg Maritime, aiming to evaluate the feasibility of integrating OpenBridge into their existing systems, which involves seamless integration of software and hardware. Additionally, the research aimed to evaluate the effectiveness of the OpenBridge standard in traditional user interfaces (UI) and human-machine interfaces (HMI), while also exploring possible avenues for implementing OpenBridge in maritime applications.

As part of the assigned tasks from Kongsberg Maritime, the group accomplished various outcomes. These included designing aspects using Figma, developing an application using Electron, and conducting an evaluation of OpenBridge. Notably, the most comprehensive design created in Figma was subsequently transformed into an Electron application, serving as a demonstration of how real-time data from a Programmable Logic Controller (PLC) could be presented within an Electron app, following the guidelines and elements provided by OpenBridge. Additionally, the group performed a comparative analysis between the OpenBridge system and existing systems, focusing on aspects such as visual aesthetics, user experience, and workflow.

The findings of the project indicate that OpenBridge offers a viable solution for creating a recognizable and secure system at sea. Its use of well-established symbols and elements contributes to a more user-friendly experience on ship bridges for novice operators. In contrast, existing systems in the present day often prioritize uniqueness and brand logos to capture clients' attention and differentiate themselves, sometimes neglecting the importance of prioritizing a user-centered safe system experience. The developers of OpenBridge are actively involved in expanding its capabilities through the development of extensions that further enhance the OpenBridge framework.

Terminology

Abbreviations

- ODES - OpenBridge Design System
- GUI - Graphical User Interface
- HMI - Human-Machine Interface
- UI - User Interface
- HCI - Human-Computer Interface
- PLC - Programmable Logic Controller
- IAS - Integrated Automation System
- CDP - Control Design Platform
- ACK - Acknowledge message
- GPS - Global Positioning System
- IDE - Integrated development environment
- npm - Node Package Manager
- IEC - International Electrotechnical Commission
- WCAG - Web Content Accessibility Guidelines
- SOLAS - International Convention for Safety of Life at Sea
- URL - Uniform Resource Locator
- SVG - Scalable Vector Graphics
- XML - Extensible Markup Language
- PNG - Portable Network Graphics
- MCS- Machinery Control System
- PMS - Power management system
- AMS - Alarm and monitoring system
- IBS - Integrated bridge system
- CMS - Cargo management system
- API - Application programming interface

Contents

List of Figures

List of Tables

1 Introduction

This chapter introduces the projects background. Including an introduction of the client and their problem formulation for this project. Lastly, a description of how the group approached the problem formulation.

1.1 Background

The maritime sector plays an important role in Norway, and it is a big industry in the northwest part of Norway. In 2018, the maritime industry employed around 84 000 people and generated a total value of 89 billion NOK [14]. Modern ship bridges as a workplace are usually composed of a large number of systems supplied by many providers, all of which are typically built using very different user interfaces (UI) philosophies [27]. Bridges, according to the OpenBridge project, have become cluttered due to human error, inefficient operations, and an increasing requirement for training. Standardization of the UI decreases the margins associated with these issues, allowing the industry to increase productivity and quality, without increased cost. Suppliers must develop and maintain numerous system variations aimed at individual suppliers or ship vendors [2]. The OpenBridge project initiative was formed to address all of these issues above by standardizing the UI in the maritime sector.

1.2 Kongsberg Maritime

Kongsberg Maritime is a global leader in maritime technology. Their solutions include marine automation, safety, maneuvering, navigation, and dynamic positioning, as well as energy management, deck handling, propulsion systems, and ship design services [24]. They have one of the most comprehensive product lines in the world, from the engine room to the working deck.

Through the Kongsberg Maritime problem formulation, it stands to gain a groundwork of understanding how to utilize OpenBridge in new and existing systems, giving older equipment a fresh and modern look. The firm stands to gain a collective knowledge base of how user experience (UX) and UI systems can be improved and how they can impact the safety of their constructions.

1.3 Problem Formulation

The project client from Kongsberg Maritime offered some suggestions about what they wanted for the project, namely a solution for the duties listed below with an emphasis on UI (Contact log L).

Integration of OpenBridge Design System with AIS systems, K-chief and Acon

- Implementing the OpenBridge Design System in an Acon (IAS) to K-Chief One Project, tying together the software and design principles to an actual hardware system.
- Evaluating the OpenBridge standard towards traditional User- and Human Machine-interfaces with regards to ease of use, safety, scalability, reliability, security, and more.
- Explore future possibilities for implementation in a broad application of maritime applications.

The group devised a suitable goal in collaboration with the client and supervisors. They were aiming to provide a comprehensive comparison of existing software and the OpenBridge design system. Examine the new design guideline, OpenBridge, and compare it to the existing systems. The client wanted an analysis of OpenBridge to determine whether it might be utilized to design and develop a marine UI in the future.
The group had a lot of leeway in approaching the project. A constraint agreed upon by the

group, supervisors, and adviser was that the design created in Figma may be less advanced in the programming section. The programming portion was thus limited to verifying whether the OpenBridge parts could handle live data, utilizing only a few pieces and not attempting to program the entire Figma design.

1.4 Approach

A meeting was convened by the research group, attended by both the client and supervisors, during which the project, ideas, and thoughts were formally presented. The group was accorded significant autonomy from the project's inception in determining the approach and resolution to the challenge at hand. The undertaking encompassed a broad array of novel systems, applications, and information. In the initial phase, the group's objective revolved around acquiring knowledge and establishing a comprehensive overview of the project's milestones, thereby fostering a deeper understanding of the project's scope and significance.

The project was logically subdivided into five distinct milestones to facilitate systematic progression:

- Milestone 1: Research and Analysis - This stage involved acquiring essential expertise in OpenBridge, software systems, design principles, programming methodologies, and the maritime industry.
- Milestone 2: OpenBridge Design and Integration - Leveraging the capabilities of Figma, the group undertook the task of developing a system adhering to OpenBridge's guidelines and incorporating its fundamental components.
- Milestone 3: Communication and App - The group focused on constructing an Electron application capable of facilitating seamless communication while integrating real-time data from a PLC into the developed app.
- Milestone 4: Usability Evaluation - An integral aspect of the project involved conducting an evaluation of the user experience associated with employing OpenBridge, encompassing its ease of use, efficiency, and overall usability.
- Milestone 5: Future Possibilities - This final milestone encompassed a thorough investigation and assessment of prospective applications and possibilities that could be realized through the utilization of OpenBridge's guidelines, offering insights into its potential future developments.

1.5 Structure of the report

The report is structured as follows:

Chapter 2 - Theoretical basis

- This chapter provides an introduction to the theoretical basis employed to acquire the knowledge necessary for the successful completion of this project.

Chapter 3 - Materials

- This chapter presents a comprehensive description of the materials utilized in this project, including various programming languages and software employed.

Chapter 4 - Method

- This chapter outlines the techniques employed by the group to design using OpenBridge in Figma and subsequently implement it in an Electron app.

Chapter 5 - Results

- This chapter showcases the results achieved throughout the project. It encompasses a detailed description of the Figma design, the programmed Electron app, and an evaluation of OpenBridge.

Chapter 6 - Discussion

- This chapter engages in a comprehensive discussion of the project's findings, including an exploration of the results, the decisions made, and an overall assessment of OpenBridge.

Chapter 7 - Conclusion

- This chapter concludes with a concise summary of the results and experiences garnered by the group throughout the project.

2 Theoretical basis

This chapter provides a comprehensive overview of the theoretical concepts this project builds upon. It covers important topics such as Integrated Automation Systems (IAS) and their relevance to the project. Additionally, it introduces the OpenBridge project, highlighting its objectives and significance. The chapter also delves into the theory of User Experience (UX) and its relevance to the project's design considerations. Furthermore, it explores various programming languages and frameworks employed in the project, shedding light on their features and benefits. Lastly, the chapter discusses companies that have successfully implemented OpenBridge design system guidelines into their systems and provides examples of the products where these guidelines have been incorporated.

2.1 Integrated automation system (IAS)

The IAS is an integrated marine automation system that can control and monitor various operations and functions on board a modern vessel. It integrates many automation and control systems into one single platform, enabling seamless coordination and communication across various subsystems such as machinery control system (MCS), navigation, Power Management System (PMS), Alarm and Monitoring System (AMS), Integrated Bridge System (IBS), Cargo Management System (CMS) and Safety and Emergency Systems [7] [17]. An IAS's major goal is to improve the efficiency, safety, and dependability of ship operations by automating and integrating numerous procedures. It substitutes manual processes with automated methods, lowering staff labor and the danger of human mistakes. The system transmits real-time data and feedback on the state and operation of several ship systems [19]. This section provides an overview of the two IAS systems Acon and K-chief, that Kongsberg Maritime offers.

2.1.1 Acon and K-Chief

Acon and K-chief are distinct systems that perform similar functions as control and monitoring of IBS, PMS, MCS, and more. Acon was developed by Rolls Royce, and the company was later bought by Kongsberg. Acon is no longer being supported, therefore it is no longer available for purchase L. K-chief was developed by Kongsberg and is the IAS system they currently offer.

2.2 Ocean Industries Concept Lab (OICL)

Ocean Industries Concept Lab (OICL) is a research group dedicated to developing knowledge that will aid the process of user-centered innovation in the maritime sector. The OICL is a part of the Institute of Design at the Oslo School of Architecture and Design. The group works closely with top industry actors on various projects and is founded on design methods such as industrial, interface, graphic, and service design, and it works on projects in close collaboration with key industry players [3].
The most recent project the OICL has had is the OpenBridge design system, which aims to provide uniform user interfaces for all systems aboard ships [3]. OpenBridge is the first open marine design system of its sort, and you can learn more about it in the section that follows.

2.2.1 OpenBridge Design System (ODES)

OpenBridge design system is a design guideline for optimizing the design, readability, and approval of maritime interfaces for equipment. It is an open-source library based on modern UI design ethics, that has been customized to maritime environments and laws. The primary purpose of OpenBridge is to standardize integration frameworks due to the contemporary issues in maritime workplaces encompassing both design and technical implementation. It is designed to support all modern maritime bridges aboard ships as well as land-based UX for maritime operations [2].

2.2.2 OpenBridge's Goal

In this subsection, the goals OpenBridge wants to realize are listed [2].

- Safe and efficient workplaces with consistent design across all systems regardless of supplier.
- Efficient technical integration that will allow maritime systems to be installed on all OpenBridge compatible ships bridge systems.
- A component-based approval system that works within current regulations

2.3 User experience (UX)

UX refers to the overall experience a person gets when utilizing a product, application, system, or service. It focuses on understanding and addressing users' needs, expectations, and emotions to create meaningful and enjoyable experiences [39].
The concept of UX is frequently used interchangeably with terms like "User interface (UI)" and "usability". Usability and UI design are key parts of UX design, yet they are subsets of it. A UX designer is concerned with the entire integration and acquisition process of a product, including design, usability, and function. This subsection describes the theory behind the various interfaces and the user experiences they offer. Human-machine interface (HMI) and human-computer interface (HCI) are interfaces used to operate a device, whereas GUI is the produced digital screen that controls the device [35].

2.3.1 Human machine interface (HMI)

HMI is a user interface that enables communication between automated systems and operators. Enabling two-way communication between humans and machines, interaction, information sharing, and machine control. An HMI's purpose is to build a user-friendly interface that facilitates communication and enables efficient control of machines [16].

2.3.2 Human computer interface (HCI)

HCI is a UI that allows a person to control a computer. Computers are not machines; they are meant to process and store data. Machines are intended to exert mechanical force. Touchscreens and keyboards are examples of HCI [16].

2.3.3 Graphical user interface (GUI)

A graphical user interface (GUI) refers to a control system that enables users to execute commands on a computer or electronic device by utilizing visually represented icons and other digital images. One commonly encountered example is the touchscreen functionality found in smartphones and tablets. These mobile devices incorporate touch-sensitive screens that display images, allowing users to interact with the displayed elements to operate the device. For instance, to access the internet on a tablet, a user can simply tap the icon representing the mobile browser application. A GUI can be described as an interface that is digitally rendered and presented on a screen [15].

2.4 Runtime Environment, frameworks & other technologies

This subsection gives an overview of the Runtime Environment, framework, and other technologies that are used in this project. In the sections that follow, you will learn more about the Node.js Runtime Environment, the Electron framework, and other technologies such as Figma, NPM, and JSON.

2.4.1 Node.js

Node.js is a server-side JavaScript development runtime environment. Node.js is for the development of building scalability, and network applications with JavaScript [4].

2.4.2 Electron

Electron is a framework specifically designed for the development of cross-platform desktop applications utilizing HTML, CSS, and JavaScript. This framework empowers developers to employ familiar web development tools to create applications that seamlessly run on diverse operating systems, including Windows, Mac, and Linux. Notably, Electron combines the functionalities of a web browser within a desktop application environment.

In addition to its cross-platform compatibility, Electron also prioritizes data security. By encapsulating an application and migrating it to the Electron framework, data remains securely stored locally within the system [37].

Furthermore, Electron facilitates high-performance application development. It significantly reduces production time and offers flexible development options by providing a unified code base for all targeted platforms. This approach alleviates challenges associated with native application development and enables developers to leverage a single code base for both web and desktop applications. Consequently, Electron offers enhanced reusability, making it a more user-friendly and convenient framework compared to alternative options. The motto "Code once, distribute everywhere" aptly encapsulates Electron's ability to streamline development and deployment processes [13].

2.4.3 Figma

Figma is a cloud-based tool used by designers to create user interfaces, prototypes, and real-time feedback. Figma gives the power to create high-fidelity designs with easy sharing functions for collective feedback, which makes it popular among UX/UI designers. Figma has a variety of features, including vector editing tools, layout and grid systems, theme components, and plugins [38].

Figma's key feature is its ability to enable real-time collaboration. Multiple group members can work on the designs simultaneously, and each member's edits are always visible to others. Figma facilitates collaboration even if the design team is not in the same physical location. Figma allows for the creation of interactive prototypes to test and validate design concepts with users.

The versatility of Figma is further amplified by its export capabilities. Design elements crafted within the Figma environment can be exported as PNGs or interactive SVGs. This flexibility enables the utilization of interactive images in coding applications, ensuring seamless integration and scalability within various software development contexts.

2.4.4 Node Package Manager (npm)

The Node Package Manager (npm) is a fundamental JavaScript package manager that is integrated within the Node.js environment. Functioning as a centralized repository, npm facilitates the maintenance of reusable code packages. By leveraging npm, developers can conveniently access and employ code packages contributed by other developers in their respective projects. This streamlined approach reduces development time by mitigating the need for extensive custom code authorship. Notably, the utilization of Node.js enables the utilization of JavaScript for both front-end and back-end development, enhancing language consistency throughout the entire application [1].

2.4.5 JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is a widely used and lightweight format for exchanging data between different systems. It provides a simple and intuitive syntax that can be easily understood by humans and processed by machines. JSON is based on a subset of the JavaScript programming language, which ensures its compatibility with popular programming languages like C, Java, JavaScript, Perl, and Python. This language-agnostic nature of JSON makes it a versatile choice for data interchange [21].

2.4.6 Scalable Vector Graphics (SVG)

Scalable Vector Graphics (SVG) represents a web-friendly file format specifically designed for vector graphics. By employing SVG, designers can create graphics that retain their quality regardless of the size or resolution of the output. In contrast, raster-based formats like Portable Network Graphics (PNG) are composed of static pixels, which impose limitations when scaled. SVG employs mathematical equations to describe the shapes and curves of an image. As a result, SVG graphics can be freely scaled without compromising details or clarity as shown in Figure 1. This attribute makes SVG an ideal choice for various graphical elements such as logos, icons, and charts that demand adaptability and flexibility. SVG supports animation capabilities through the utilization of CSS or JavaScript [33][34].

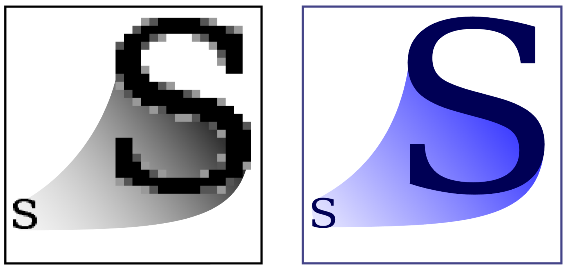


Figure 1: PNG vs SVG

2.5 Companies that have implemented OpenBridge

OpenBridge is used by over 170 companies from all around the world that have registered to use the guideline [31]. This subsection will provide an introduction to three companies, as well as their products.

2.5.1 CDP Studio

CDP Studio is a modern software platform for developing complex control and automation systems. The software is produced by the independent Norwegian software company, CDP Technologies. The intention of CDP is to simplify control system development so that companies can create next-generation industry solutions [9].

CDP Studio is a development tool that includes the OpenBridge guidelines. Designing and developing OpenBridge UIs in CDP Studio is fast and efficient. The use of a no-code editor with the OpenBridge library significantly reduces the complexity and knowledge required to design UIs in accordance with the standard [30].

2.5.2 Alphatron Marine

Alphatron Marine is a supplier of integrated bridge solutions and a producer of unique complementary products. Their software Conning and Docking are applications of their powerful all-in-one software suite AlphaMINDS (Multifunctional Information Navigation & Docking System).

The AlphaMINDS software is based on their automation and connecting platform Lynx, designed as maritime SCADA software for monitoring and controlling all equipment onboard a vessel. Integrated with Lynx is the OpenBridge design guideline [26].

2.5.3 SEAM AS

SEAM AS is a supplier of zero-emission solutions to the maritime industry. Their goal is to develop and deliver zero-emission solutions, securing the transition to green shipping. SEAM aims to make voyages environmental and operation efficient, predictable, and risk-free, by developing, delivering, and implementing cost-reductive and future-ready technology and systems [12].

The bridge solution e-SEA Bridge was the first bridge solution based on the OpenBridge design system. e-SEA Bridge is a bridge solution that integrates what the captain needs to remotely control and monitor while sailing, in easily accessible control units connected to the captain's seat.

2.6 Regulations

Regulations play a crucial role in the maritime sector, ensuring the provision of safety, security, and environmental protection measures. This subsection aims to provide an overview of three fundamental regulations that hold relevance for the OpenBridge guideline, IEC 62288, WCAG 2.0, and SOLAS.

2.6.1 IEC 62288

IEC 62288 is an international standard developed by the International Electrotechnical Commission (IEC), which is titled "Maritime navigation and radiocommunication equipment and systems - Presentation of navigation-related information on shipborne displays - General requirements, methods of testing and required test results". The standard provides guidelines for the presentation of navigation-related information on shipborne displays, including electronic charts, radar images, and other navigational information [18].

2.6.2 WCAG 2.0

WCAG 2.0, stands for "Web Content Accessibility Guidelines 2.0," constitutes a technical standard disseminated by the World Wide Web Consortium (W3C). It aims to furnish comprehensive guidelines for enhancing the accessibility of web content, particularly for individuals with disabilities. This standard outlines directives pertaining to the perceptibility, operability, comprehensibility, and robustness of web content. It covers a broad spectrum of considerations, including the provision of text alternatives, the design of content conducive to keyboard and mouse navigation, and the facilitation of seamless user experiences for individuals with disabilities [36].

2.6.3 SOLAS

The International Convention for the Safety of Life at Sea (SOLAS) represents an internationally recognized maritime convention that establishes minimum safety standards for ship construction, equipment, and operation [32]. Initially adopted in 1914, this convention has undergone various revisions and amendments over the years, with the most recent iteration being SOLAS 2020 [20]. Its overarching objective is to safeguard human life at sea, promote maritime safety, and mitigate potential hazards through a comprehensive framework of regulations.

3 Materials

This chapter describes the materials used to complete this project. The project is mostly made up of software, and the Tables below list the program software, libraries, and hardware that were utilized, as well as a description of how they were used.

3.1 Software

Table 1 shows an overview of the software used in this project.

Softwares		
Software	Version	Description
OpenBridge design system	4.0/5.0 BETA	Guideline built on modern principles of user interface and workplace design adapted to maritime context and regulations
Figma		Collaborative interface design tool
CDP Studio	4.11	Software platform for developing comprehensive control and automation systems
JavaScript		Programming language
Electron	23.1.4	Open-source software framework for building desktop applications
e!Cockpit	1.11.1.2	Wago PLC programming IDE
Visual studio code	1.78.1	IDE
npm	9.5.0	Open source code repository for JavaScript
Git	2.40.0	Version control system
GitHub		Collaboration tool

Table 1: Overview of programs used

3.1.1 Libraries

Table 2 lists the libraries used to achieve tasks.

LIBRARY			
Library	Version	Program	Usage
Nodemon	2.0.22	npm	Used to continuously update electron for every save without restarting the software
Modbus-serial	8.0.11	npm	Library used for Modbus communication from node package manager
openbridge-css	0.2.2	npm	OpenBridge's online CSS for fonts/styles within its designs
openbridge-web-components	0.2.2	npm	OpenBridges early design for components to implement into clients' systems
jsmodbus	4.0.6	npm	Javascript Modbus library used for establishing communication with Modbus

Table 2: Overview of libraries used

3.2 Hardware

Table 3 shows the hardware that was used in this project. The PLC was used to check the compatibility of Electron, using live data through Modbus.

HARDWARE		
Hardware	Version	Description
Wago PLC	750-8214	PLC used to check compatibility

Table 3: Overview of hardware

4 Method

This chapter contains in-depth explanations of the techniques used to solve the project milestones. First how the project was organized is described, focusing on how the project was planned and executed. Next, a description of the method used to develop the various designs in Figma, followed by how the design was programmed in Electron and the application.

4.1 Project Organization

The project team consisted of four students who alternated between the roles of project leader and secretary. This arrangement was implemented to ensure that each member had additional responsibilities and tasks. The project leader's primary duties encompassed organizing, disseminating information, and supervising meetings with supervisors and the client. The secretary assumed organizational responsibilities such as maintaining biweekly updates (see appendix D), managing the gantt diagram (appendix B), and compiling meeting notes (appendix E). Despite the existence of a designated project leader, all decisions were made collectively by the group, reflecting their shared and individual accountabilities for the project's outcome.

Regular meetings were scheduled biweekly to facilitate effective communication and collaboration. During the meetings, the group presented problem-solving ideas and showcased Figma designs, engaging in discussions regarding potential solutions with supervisors and the client. Prior to project initiation, a preliminary-project plan was formulated (see appendix A), accompanied by the creation of a gantt diagram and a description of the key milestones, as introduced in Chapter 1.4. These initiatives aimed to provide a holistic overview of the entire project. Figure 2 presents a visual representation of the three key parties involved in the project.

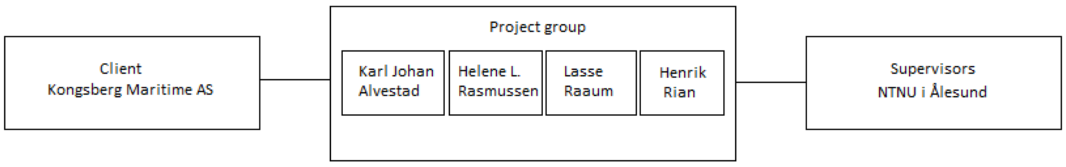


Figure 2: Organization chart

4.2 Design in Figma

This section begins with an introduction to the utilization of OpenBridge in Figma, covering the steps involved in creating an OpenBridge UI within the Figma platform and exploring the potential opportunities it presents. Furthermore, later in this chapter, the process of generating the project's designs will be discussed.

4.2.1 Using OpenBridge in Figma

Figma provides access to the most recent OpenBridge component libraries as well as the ability to design and develop UI prototypes. The OpenBridge library is obtained via the OpenBridge website, which offers a direct link to its Figma library [29]. This library is open source, and several contributors can post comments and submit requests about the various components.

To utilize the OpenBridge library in your own drafts, you can duplicate it and incorporate it into your design process. By doing so, you gain access to the components and can start creating UI sketches that adhere to the OpenBridge guidelines. See Figure 3 for an illustration of the OpenBridge library in Figma.

To fully utilize Figma's collaborative features, it is recommended to upgrade to a Professional account. This upgrade allows you to create a team and collaborate with other team members on the same project, enhancing the collaborative design and development process.

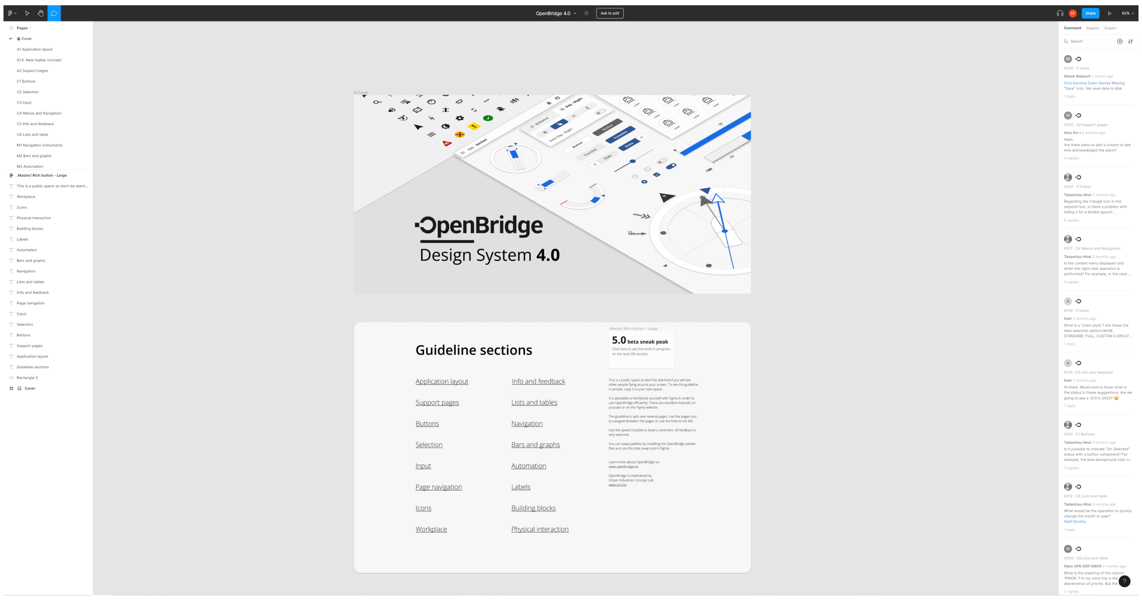


Figure 3: The OpenBridge library in Figma

4.2.1.1 Adding OpenBridge components in Figma

When adding pages to your Figma project, you start with an empty page where you can begin adding components. Figure 4 illustrates this initial empty page where you can start building your UI.

The library is located on the left sidebar. The components library is organized into different categories such as buttons, instruments, menus, bars, and more. These categories make it easier to find the specific components you need for your design. Figure 5a showcases the categorized library in the sidebar.

The library offers a search bar, allowing you to narrow down your search and find the desired components. This makes it efficient to locate specific elements for your UI design.

The library supports a drag-and-drop function. By clicking and holding the mouse button over a component, you can drag it to the desired location in your design. This drag-and-drop functionality simplifies the process of integrating components into your project. Figure 5b provides an example of the search bar and drag-and-drop function.

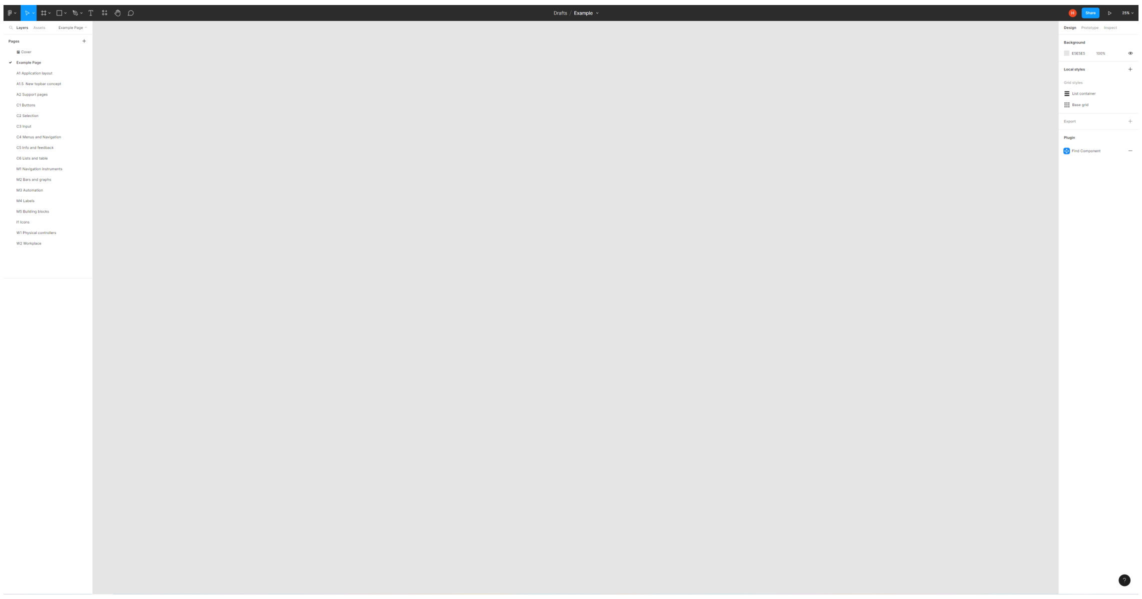
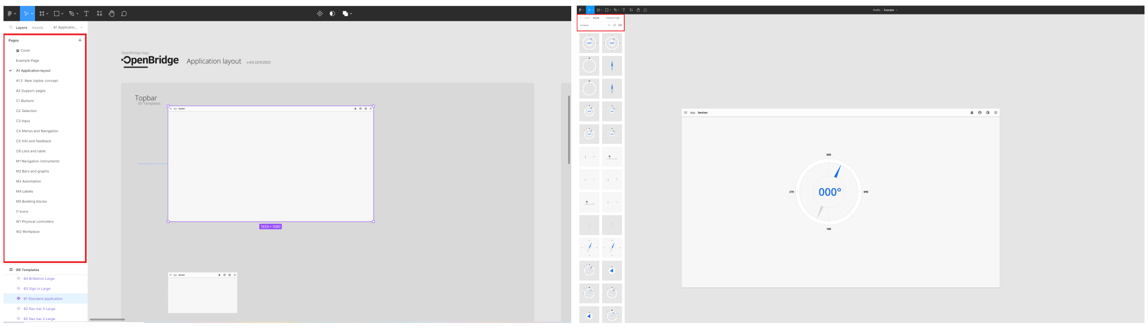


Figure 4: Empty page in Figma



(a) Copying components from library pages (b) Search for components in the assets search bar

Figure 5: Illustration of finding OpenBridge components in Figma

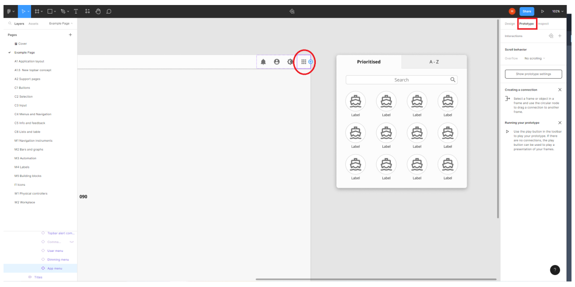
4.2.1.2 Testing the OpenBridge UI in Figma

In Figma, you have the capability to test your UI design by setting up interactions between different components. This allows you to evaluate the usability and flow of your UI. The prototype function in Figma facilitates this process.

Using the prototype function, you can select the desired component and define its interaction with other components by setting up the desired action. For example, in the case shown in Figures 6a and 6b, clicking the app menu icon in the top bar triggers a pop-up menu to appear.

Figma offers various types of interactions to choose from, as depicted in Figure 6b. You can select the appropriate interaction that matches your intended behaviour.

By running the prototype, you can test and experience the interactions in action. Figures 7a and 7b provide visual examples of how the interactions can be tested during the prototype evaluation process. This allows you to assess the functionality and UX of your UI design.


(a) Setting an interaction for the app's icon

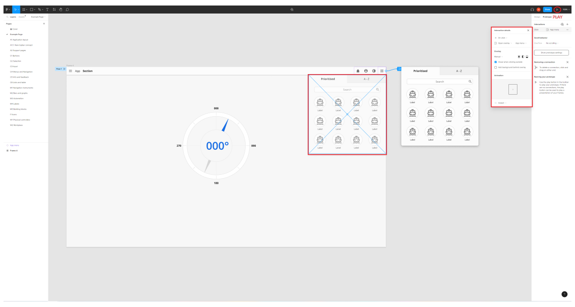
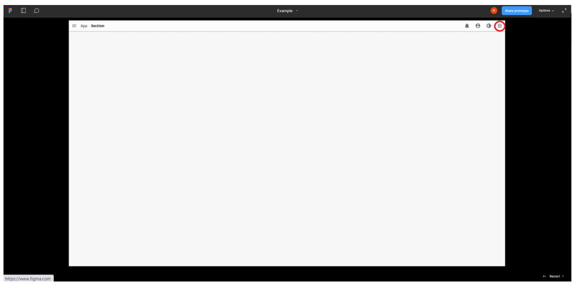

(b) Connecting the interaction to a desired component. In this case a Pop-up menu. And then set the desired interaction

Figure 6: Setting interactions between components in Figma


(a) Running the prototype, and clicking the icon

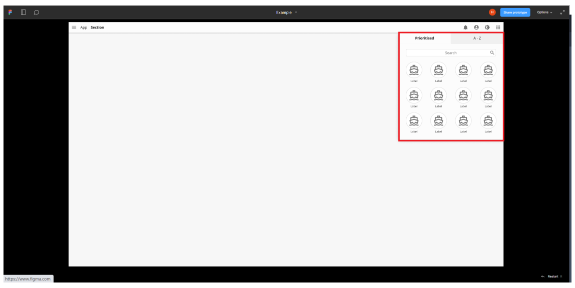

(b) When the icon is clicked the pop-up menu is revealed

Figure 7: Illustration of running a prototype in Figma

4.2.1.3 Creating customized components in Figma using OpenBridge

The OpenBridge library offers building blocks that can be utilized to construct components, as illustrated in Figure 8. These building blocks serve as the foundation for creating customized components tailored to specific needs.

By using these building blocks, you can design and develop components that align with your desired specifications and requirements. These customized components, although not included in the official OpenBridge library, can be created using the building blocks provided.

Furthermore, if you believe that your customized components could benefit the broader OpenBridge community, you have the option to submit them as suggestions to the OpenBridge developers. They can then evaluate and potentially incorporate your suggested components into future updates of the library, expanding its functionality and versatility.

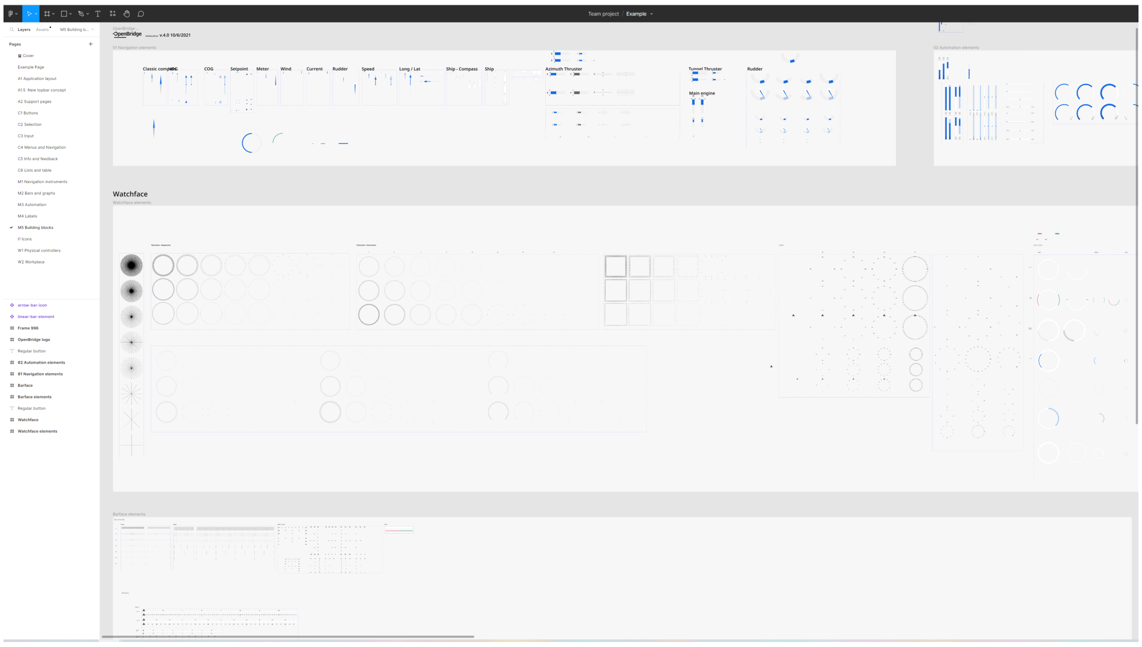


Figure 8: Building blocks library in Figma

4.2.1.4 Changing palettes of the UI in Figma

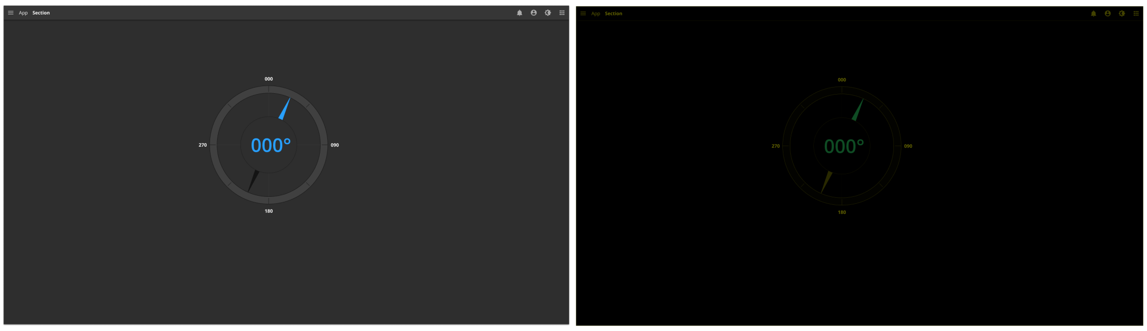
In Figma, it is possible to modify the colour palette of the UI to visualize different colour schemes, such as dusk and night mode. The OpenBridge website offers Figma files that include various colour styles for different palettes, which can be accessed through the provided link [1].

To apply these palettes to your design, Figma provides a "swap library" function, which allows you to replace the existing colour styles with the desired palette [2]. By utilizing this function, you can switch between different colour palettes to visualize how they impact the UI.

Figure 9 showcases an example of a palette swap.

[1]Figma library and palettes library, Link: https://www.openbridge.no/figma/current-release

[2]How to change libraries in Figma, Link: https://help.figma.com/hc/en-us/articles/4404856784663-Swap-style-and-component-libraries



(a) Palette swapped to dusk (b) Palette swapped to night

Figure 9: Palettes swapped in Figma

4.2.2 Design process of UI in Figma

The design process for creating a new interface begins with conducting research and analyzing existing interfaces. As the group had limited experience in designing UI for maritime contexts, we utilized the Acon and K-Chief interfaces as templates for reference, as shown in Figure 10. These templates provided a starting point for designing new interface drafts in Figma, incorporating the OpenBridge library. The process of developing drafts is illustrated in Figure 11.

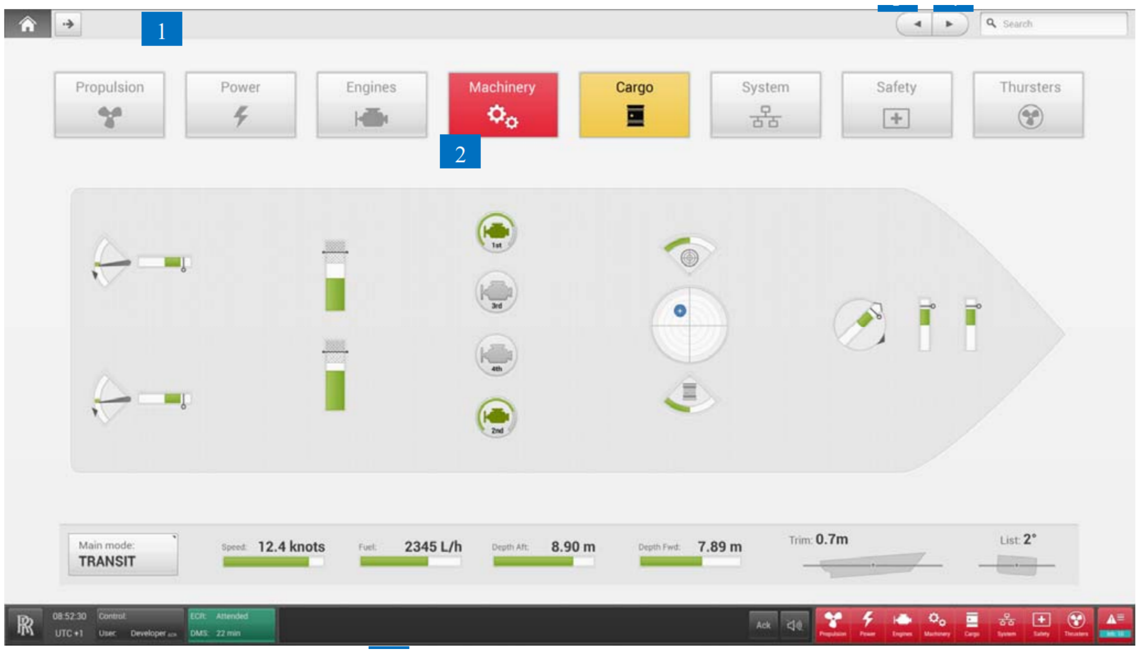


Figure 10: One of the templates provided by Kongsberg Maritime. This is specifically an overview page of an IAS system

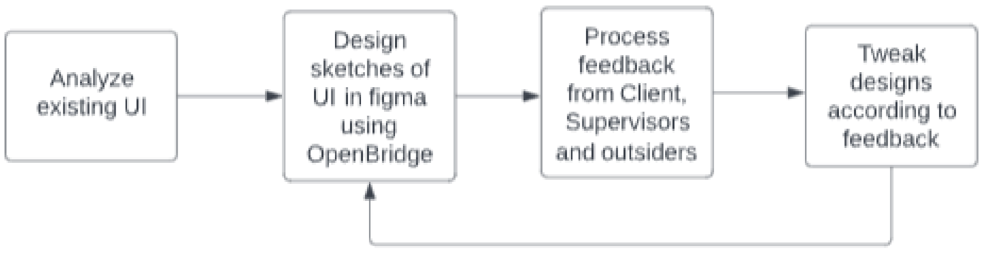

Analyze existing UI
Design sketches of UI in figma using OpenBridge
Process feedback from Client, Supervisors and outsiders
Tweak designs according to feedback

Figure 11: Flow chart of the design process in Figma

The provided interfaces served as valuable tools and guides throughout the design process in Figma. During the analysis of these interfaces, the group focused on gathering specific information to inform our own design decisions. The following key points were particularly sought after by the group:

- General layout: Understanding the overall structure and arrangement of elements in the interfaces.
- Necessary information: Identifying the essential information that users need to access and interact with.
- Always available information: Determining which information should be readily accessible to the user at all times.
- Examining how alarm-related information is presented to the user, ensuring effective alert management.
- Colour usage: Exploring how colours are employed in the interface design, including colour schemes and the representation of different states or modes.
- Navigation methods: Investigating how users navigate through the interface, such as through menus, buttons, or other interactive elements.

By addressing these key points and considering the insights gained from analyzing the provided interfaces, the group established drafts in Figma that incorporated important design considerations meeting the needs of the intended users.

4.3 Application and programming structure development

In this section, the group reviewed various coding techniques for creating a demo for structuring components in an application.

4.3.1 Application

One of the project's goals was to develop an application that was capable of handling live data from a PLC and displaying the values inside the application. This section describes the technologies that were used to make it possible. The project is open-sourced and can be used by everyone that wants to try OpenBridge.

4.3.1.1 Electron functions

None of the members within the group possessed any prior experience in JavaScript, HTML, or CSS coding. To bridge this knowledge gap, the initial phase of the project involved engaging in self-directed learning through the utilization of freely available crash courses discovered through online platforms such as Google and YouTube. The supervisors played an important role by offering recommendations and conducting research, leading the group to familiarize themselves with Electron.

During the selection process for the application's primary platform provider, Electron was chosen to be the framework. The group ultimately opted for Electron due to its straightforward architecture and framework. Leveraging JavaScript, HTML, and CSS, Electron facilitated a smoother learning curve for the group members in contrast to competing options. Notably, the application harnessed the Chromium engine for rendering, thereby simplifying the utilization of developer tools and facilitating access to storage.

The installation process for Electron involved utilizing npm within the Node.js environment. Upon successfully installing Node.js and navigating to the desired directory within the terminal, Electron could be installed by executing the command in snippet 1.

Listing 1: Installing Electron example

```
npm init
npm install --save-dev electron
```

After installing electron it's needed to go into the package.JSON file and edit by inserting a script that starts electron, as shown in snippet 2.

Listing 2: Standard script added snippet

```
  "scripts": {
    "start": "electron ."
  }
```

The script allows the application to be launched using the new "start" command, which replaces the previous "electron ." command. In order for Electron to be launched correctly, it requires a relative path to the "main.js" file from the directory where the "package.json" file is located. This path specifies what should be launched within the Electron program. An example of a standard "package.json" file is shown in snippet 3, where you can observe the "main" property on line 5.

Listing 3: Standard package JSON file

```
1 {
2   "name": "my-electron-app",
3   "version": "1.0.0",
4   "description": "Hello World!",
5   "main": "./main.js",
6    "author": "Jane Doe",
7    "license": "MIT"
8  }
```

By following the quick start guide provided on the Electron website [3], obtains the complete standard code that serves as a foundation for building a basic Electron application. The code snippet 4 demonstrates the standard structure.

Listing 4: standard Electron example

```
1  const { app, BrowserWindow } = require('electron')
2  const path = require('path')
3
4  function createWindow () {
5    const win = new BrowserWindow({
6      width: 800,
7      height: 600,
8      webPreferences: {
9        preload: path.join(__dirname, 'preload.js')
10     }
11   })
12
13   win.loadFile('index.html')
14 }
15
16 app.whenReady().then(() => {
17   createWindow()
18
19   app.on('activate', () => {
20     if (BrowserWindow.getAllWindows().length === 0) {
21       createWindow()
22     }
23   })
24 })
25
26 app.on('window-all-closed', () => {
27   if (process.platform !== 'darwin') {
28     app.quit()
29   }
30 })
```

The Electron application utilizes the app module[4] and BrowserWindow module[5] on line 1, which are imported using the require tag to leverage the functionality provided by these modules from Node.js. When executed correctly, running the command ”npm start” will open an empty application window.

As shown in the quick start guide, the whenReady function is used to ensure that the entire program is loaded before launching the browser window, reducing the likelihood of crashing during the booting process. This function ensures that certain operations are withheld until all variables are established, preventing attempts to apply new values to each variable prematurely.

The program offers various features. The ability to set custom frame sizes, enable full-screen mode, specify the monitor on which to launch the application, and determine the application to switch when the display becomes disconnected. In snippet 5, lines 3, 4, and 6 demonstrate the capability to detect extended displays and configure the appropriate display resolution.

Additionally, on lines 11 and 12 of the code snippet, the integration of Node.js modules is enabled within the Electron frame, allowing for global data sharing instead of being limited to a single

[3]electronjs.org
[4]information about the module: https://www.electronjs.org/docs/latest/api/app
[5]information about the module: https://www.electronjs.org/docs/latest/api/browser-window

window. It is important to note that the approach may pose security risks, but the risks are confined to the shared repositories of the Node.js modules[6].

Listing 5: Createwindow Function for full main.js in appendix I

```
function createWindow() {
  const { screen } = require('electron')
  const primaryDisplay = screen.getPrimaryDisplay()
  const { width, height } = primaryDisplay.workAreaSize
  mainWindow = new BrowserWindow({
    fullscreen: true,
    width,
    height,
    webPreferences: {
      nodeIntegration: true,
      contextIsolation: false,
    }
  });
```

The consideration of security should be evaluated on a project-by-project basis, as the requirements and needs can vary. In the case of the application demo, setting the context isolation to false can be a reasonable choice to avoid unnecessary challenges during the demonstration. It is important to note that in a production environment, security measures should be carefully assessed and implemented based on the specific requirements and potential risks associated with the application.

It is recommended to conduct a thorough security assessment and consider implementing appropriate security measures based on factors such as data sensitivity, potential vulnerabilities, user authentication, and authorization requirements. Adhering to best practices and following the guidelines provided by the Electron documentation can help ensure the security and integrity of the application.

4.3.1.2 File structures

In an Electron application, there are three main files involved in the boot-up process: main.js, renderer.js, and the HTML webpage. Each of these files has a specific role in the application's function.

The HTML document serves as the main layout for the program. It includes other files, such as CSS stylesheets and JavaScript files, to provide the necessary functionality for the application.

The main.js file contains the back-end functions and core operations that continuously occur in the background. It handles tasks such as creating the main Electron application window, managing application lifecycle events, and handling system-level operations.

The renderer.js file is responsible for communication channels between the main process and the renderer process. It establishes a channel to send information from the main process to the renderer process, allowing it to be utilized within the context of the HTML environment. The renderer process handles UI interactions and manipulates the HTML elements dynamically.

For communication between the main.js and renderer.js files, Electron provides the ipcMain and ipcRenderer modules. The ipcMain module is used in the main.js file to handle messages and events from the renderer process. It allows the main process to listen and respond to specific channels or events triggered by the renderer process. An example of using ipcMain is shown in snippet 6.

Listing 6: IpcMain example

```
const { ipcMain } = require('electron');
ipcMain.on("raised flag", (event) => {
  client.doSomething(function (err, data) {
    if (err) {
```

[6]information about the security: https://www.electronjs.org/docs/latest/tutorial/security

```
      console.log("error : " + err);
      return;
    }
    event.reply("named-channel", data.data[information]);
  });
});
```

The ipcRenderer module is used in the renderer.js file to send messages or trigger events from the renderer process to the main process. It allows the renderer process to communicate with the main process and exchange information. An example of using ipcRenderer is shown in snippet 7.

Listing 7: IpcRenderer example

```
const { ipcRenderer } = require("electron");
const namedElement = document.getElementById("raised-flag");
ipcRenderer.on("raised-flag", (event, data) => {
  console.log("named-channel" + data);
  namedElement.innerHTML = data;
});
```

4.3.1.3 The process

To achieve real-time synchronization of data across multiple control stations or clients in different locations, you would typically need a server-client architecture.

To achieve a system that can be accessed from multiple locations, the main, renderer, and HTML/CSS components can serve as individual clients that mimic the values of the server. They can also send information back to the server in the same order, establishing bidirectional communication. An overview of this ideal process is depicted in Figure 12, while the communication process is illustrated in figure 13.

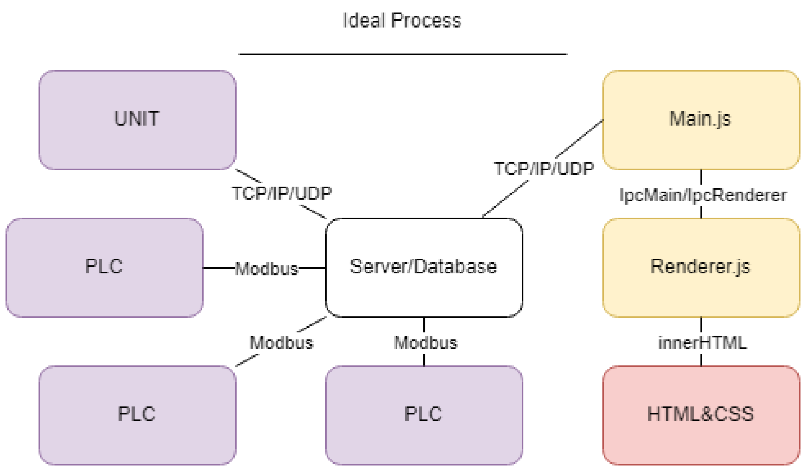


Figure 12: Overview of the ideal process

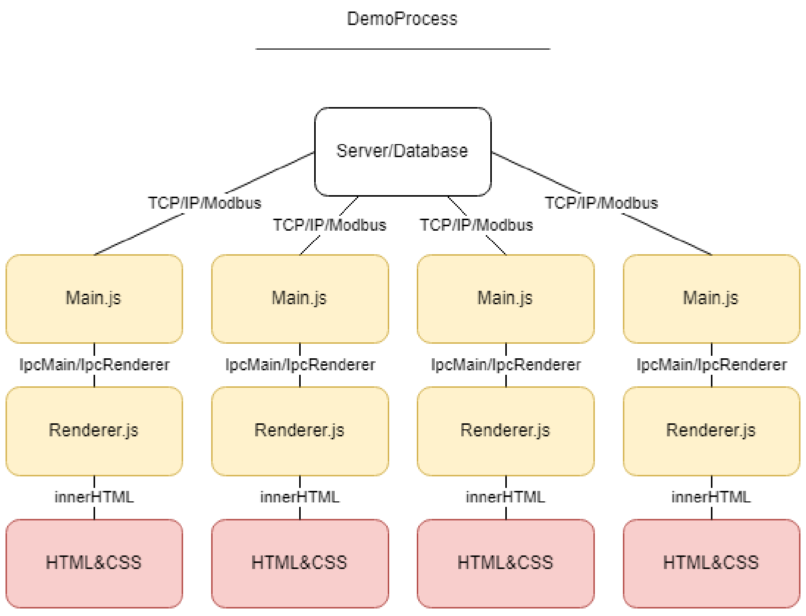


Figure 13: Overview of the communication between the server and clients

To demonstrate the feasibility and execution of this process in a larger project, the group made the decision to develop a smaller contextual demo. This demo involves retrieving values from a PLC and replicating those values in an Electron window. The communication between the PLC and the Electron window is depicted in figure 14.

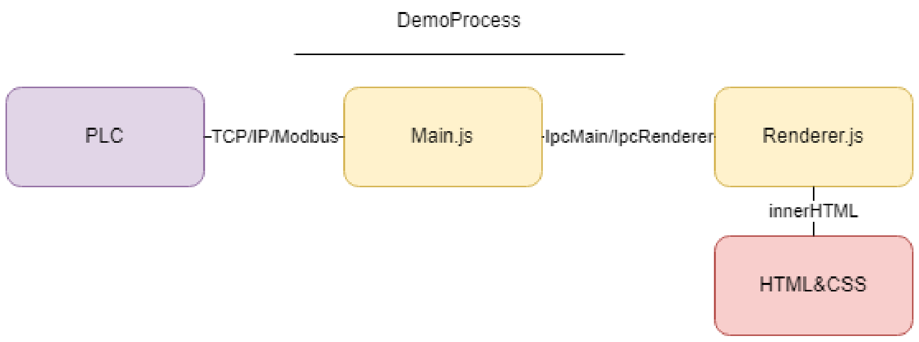


Figure 14: Overview of the demo process

4.3.1.4 The demo

The modular design of the demo enables its utilization as an illustrative example of creating components for a larger system that accurately reflects values on the screen. The process involves utilizing e!Cockpit, to configure a PLC to receive a signal on an input register. This signal is then stored in a new variable within the PLC, which subsequently returns the signal on a different register. Additionally, the demo incorporates boolean values as buttons, which are reflected on different registers.

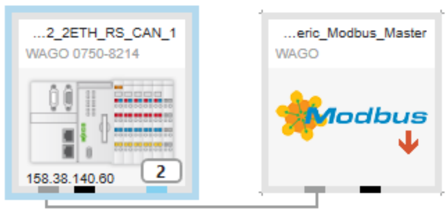
Snippet 8 showcases the e!cockpit program, which provides a visual representation of the configuration and setup of the PLC.

Listing 8: E!cockpit program for full program see appendix K

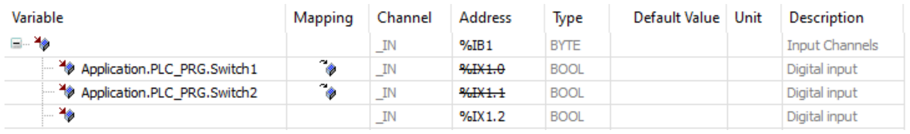
```
VarCount := VarCount + 1;
Sine:= IN_1;
OUT_1:= Sine;
OUT_2 := Switch1;
OUT_3 := Switch2;
```

The e!Cockpit program is installed on a Wago PLC model 750-8214, and it is connected to a virtual Modbus as depicted in Figure 15a. The PLC is connected to a local network with a static IP address.

In the program, a value is received on register 32000 and stored in the Sine variable. This value is then re-emitted on register 2, as shown in Figure 15c. Additionally, the program sends the status of its inputs to their respective registers, as illustrated in Figure 15b.

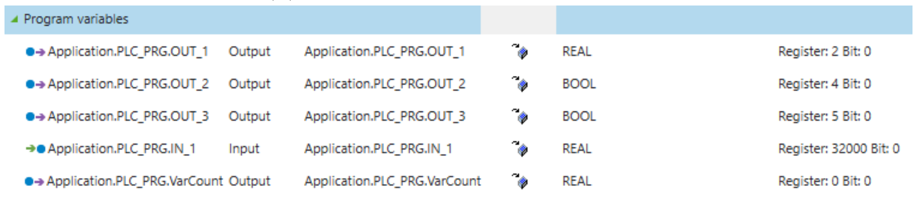


(a) Wago PLC and Modbus



Variable	Mapping	Channel	Address	Type	Default Value	Unit	Description
		_IN	%IB1	BYTE			Input Channels
Application.PLC_PRG.Switch1		_IN	~~%IX1.0~~	BOOL			Digital input
Application.PLC_PRG.Switch2		_IN	~~%IX1.1~~	BOOL			Digital input
		_IN	%IX1.2	BOOL			Digital input

(b) The assigned inputs to the PLC



Program variables

Application.PLC_PRG.OUT_1	Output	Application.PLC_PRG.OUT_1	REAL	Register: 2 Bit: 0
Application.PLC_PRG.OUT_2	Output	Application.PLC_PRG.OUT_2	BOOL	Register: 4 Bit: 0
Application.PLC_PRG.OUT_3	Output	Application.PLC_PRG.OUT_3	BOOL	Register: 5 Bit: 0
Application.PLC_PRG.IN_1	Input	Application.PLC_PRG.IN_1	REAL	Register: 32000 Bit: 0
Application.PLC_PRG.VarCount	Output	Application.PLC_PRG.VarCount	REAL	Register: 0 Bit: 0

(c) All the emitting Modbus variables

Figure 15: The complete overview of the Wago PLC program, see appendix K for full program

To accomplish the demo's functionality, the Modbus-serial library was utilized. The library, which can be obtained from npm, offers support for Modbus communication. The installation of the library was carried out by executing the command shown in Snippet 9 within the terminal.

Listing 9: Terminal call to install modbus-serial

```
    Npm install modbus-serial
```

To establish a connection with an existing Modbus using the Modbus-serial library, the environment can utilize the command depicted in Snippet 10. This command, provided by the library, allows the environment to initiate a connection with the desired Modbus device.

Listing 10: Modbus connect and read function

```
// create an empty modbus client
const ModbusRTU = require("modbus-serial");
const client = new ModbusRTU();

// open connection to a tcp line
client.connectTCP("127.0.0.1", { port: 8502 });
client.setID(1);

// read the values of 10 registers starting at address 0
// on device number 1. and log the values to the console.
setInterval(function() {
    client.readHoldingRegisters(0, 10, function(err, data) {
        console.log(data.data);
    });
}, 1000);
```

Using the Modbus-serial library, the group developed a function to facilitate the reading and writing of a sine signal to and from the input registers 0 through 7. This function was created to establish communication between the environment and the Modbus device, as demonstrated in Snippet 11. Initially, the code was designed to send a positive sine signal ranging from 0 to 2 over Modbus. However, to mitigate potential complications, the code was subsequently modified to send a signal ranging from -1 to 1 within the program by subtracting the necessary values.

Listing 11: Code for PLC values for the demo full code inside main.js in appendix I

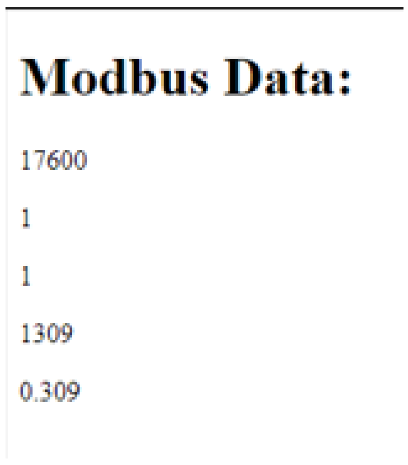
```
function connectModbus() {
  client.connectTCP("158.38.140.60", { port: 502 })
    .then(() => {
      console.log("Connected to Modbus PLC");
      startReadingModbus();
      sendData();
     })
     .catch(error => {
      console.error(error);
    });
  }

// Generate and write the sine wave values continuously
function sendData() {
      let i = 0
      setInterval(() => {
      if (i < NUMSAMPLES) {
        value = Math.sin(2 * Math.PI * FREQUENCY * i / NUMSAMPLES) * AMPLITUDE + OFFSET;
        i++;
        value = value.toFixed(3); //Restrict the decimals
        client.writeRegisters(REGISTERADDRESS, [value]) //send values
        .catch(error => {
          console.error(error);
        });
        i=0;
    }, DELAY);
  }

//Receive signals
function startReadingModbus() {
  setInterval(() => {
    client.readInputRegisters(0, 7, function (err, data) {
      if (err) {
        console.log("Modbus error: " + err);
        return;
      }
      SINVALUE = data.data[SINRAW];
      mainWindow.webContents.send('sine-raw-data', SINVALUE);
    });
  }, DELAY);
}
```

The use of mainWindow.webContents.send allows the HTML environment to receive and handle the values for projection within the HTML context.

By utilizing the constructed system, the HTML component receives the necessary information to project values. This can be observed within the index.html file, as presented in Appendix I. The outcome of this process is depicted in Figure 16, which showcases the projection of values from the connected PLC within the Electron application.



Modbus Data:

17600

1

1

1309

0.309

Figure 16: Live Data from PLC updated in Electron, see index.html in appendix I

4.3.1.5 Implementing the design into Electron

The group sought quick solutions to implement Figma designs into the application. This approach aimed to streamline and speed up the repetitive process associated with each production, particularly when aligning with various OpenBridge release versions. During their search, the group discovered community-developed plugins that facilitate the export of SVG files as comprehensive and interactive constructs.

the plugins were SVG to JSX with MUI, which exports components as React code components. This plugin allows for seamless integration of Figma designs into React-based projects[7]. Another plugin called Codelessly simplifies the conversion of Figma designs into Flutter code[8]. Additionally, there is a plugin called Figma to code that enables the conversion of designs into code in the language of choice[9].

It is worth noting that currently, there is no universally recognized and future-proof solution for this particular step in the process. The available plugins provide various options for exporting and converting Figma designs, but further research and exploration may be required to identify the most suitable and sustainable solution.

The group determined that the most advantageous approach for implementing a page was to export it as an SVG file for the demo. The group chose to import the conning page from the first UI version (V0.1), which can be found in appendix H. This page was selected because it contains reusable components that can be leveraged in the future of the implementations.

When exporting the page from Figma, several important settings need to be considered. These include checking the options for "ignore overlapping layers," "include bounding box," and "include 'id' attribute." Additionally, it is recommended to uncheck the options for "outline text" and "simplify stroke." Enabling the "include 'id' attribute" option ensures that the IDs of the document's elements are set accordingly. Choosing to uncheck "outline text" allows the text to remain as editable text within the SVG, which can be manipulated and stored as a variable. This is preferable to having the text appear as an image representation of the text example in the design, the different exports are shown in snippet 12.

Listing 12: Example Code for different exports

```
//correct "id" attribute and non-outlined text example
    <text id="Label" fill="black" fill-opacity="0.6" xml:space="preserve"
         style="white-space: pre" font-family="Open Sans" font-size="32"
         letter-spacing="0em"><tspan x="0" y="35.4141">Pressure</tspan></text>

```

[7]Source: https://www.figma.com/community/plugin/1115700158761255786/SVG-to-JSX-with-MUI
[8]Source: https://codelessly.com/#careers
[9]Source: https://www.figma.com/community/plugin/1083031796594968801/

```
//incorrect "id" attribute and outline text example
   <path d="M9.09375 12.1562C12.0417 12.1562 14.1927 12.7344 15.5469 13.8906C16.9115
       15.0365 17.5938 16.6823 17.5938 18.8281C17.5938 19.7969 17.4323 ...
```

If the exported page contains multiple units of the same components, there may be a risk of overlapping IDs. Figma addresses this issue by appending "$_n + 1$" to the ID, ensuring uniqueness. To avoid confusion, it is important to appropriately name each component before exporting.

Once the SVG code is placed within an HTML file, it becomes possible to manipulate specific elements by using the getElementById(" ") function[10] (later replaced with querySelector("#"))[11]. This approach allows for the creation of modular functions that can be called to rotate or set values for different variables. By connecting the "sine-raw-data" mentioned in Snippet 11 to the IDs attributes, the earlier method described in section 4.3.1 can be integrated. This method saves significant production time as it eliminates the need to recreate components from scratch for every Figma design.

To ensure that variable values are not changed before the variables themselves are created, the code is structured to execute static functions first. After a brief timeout, the remaining functions are loaded. Snippet 13 provides an example of the static section of the windAndWeather.html component in appendix I.

Listing 13: Code of the buttons inside the windAndWheater Component for full code got to windAndWheater inside component folded in appendix I

```
export function windStartUp() {
  var windButtons = document.querySelectorAll("#tab-item");
  var svgWind = document.querySelector("#svgWind");
  var svgWeather = document.querySelector("#svgWeather");
  var weatherButtons = document.querySelectorAll("#tab-item_2");

  windButtons.forEach((buttonElement) => {
    buttonElement.style.cursor = "pointer";
    buttonElement.addEventListener("click", function () {
      console.log("clicked wind.");
      svgWind.style.display = "inline";
      svgWeather.style.display = "none";
    });
  });
  weatherButtons.forEach((buttonElement) => {
    buttonElement.style.cursor = "pointer";
    buttonElement.addEventListener("click", function () {
      console.log("clicked weather.");
      svgWind.style.display = "none";
      svgWeather.style.display = "inline";
    });
  });
}
```

By utilizing the "windStartUp" function in the includer.js file, the system can systematically initialize itself and establish its necessary components before executing any changes. This function takes priority in ensuring the proper setup of the system. Following the start-up process, the switch-case statement[12] in Snippet 14 is executed to handle different cases and perform corresponding actions.

Listing 14: Code of the includer to bring component htmls into the main page for full code see I

```
import { windStartUp } from "./Components/windAndWheater.js";
import file from "./components.json" assert { type: "json" };
```

[10]Source: https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById
[11]Source: https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector
[12]Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch

```

file.components.forEach((component) => {
  switch (component.name) {
    case "windAndWeather":
      container.innerHTML += `<div data-windspeed="${component.speed}"
          data-winddir="${component.direction}"
          w3-include-html="./Components/windAndWeather.html"></div>`;
      setTimeout(() => {
        windStartUp();
      }, 1000);
      break;
    case "speedometer":
      container.innerHTML +=
        '<div w3-include-html="./Components/Speedometer.html"></div>';
      break;
```

After the booting order, the code will fetch the necessary functions by utilizing an import statement in the script section of the main "indexNew.html" page in appendix I. This import statement, as illustrated in Snippet 15, allows the code to access and use the required functions.

Listing 15: Code of the function import and call with values inside the main HTML page for full code see appendix I

```
    <script type="module" src="Includer.js"></script>
        <script type="module">
        import {
            windWindRotate,
        } from "./Components/windAndWheater.js";
        windWindRotate(0);
        </script>
    </script>
```

The function calls the value 0 into the function defined in the windAndWheater.js file that changes the value within the id of windAndwheater.html as shown in snippet 16. The windDir gets a minus value on line 3 before adding the new value in the parameter using the setAttribute[13]. It is to set the offset to zero in the correct zero position, making the absolute zero rotate in relevance to the forward position.

Listing 16: Code of the function windWindRotate, for full code see appendix I

```
1 export function windWindRotate(windDir) {
2   setTimeout(() => {
3     windDir -= 35;
4     var windDirEl = document.querySelector("#windAndWheaterFrame_331");
5     console.log(windDir);
6     windDirEl.setAttribute("transform", "rotate(" + windDir + ",181,116)");
7   }, 200);
8 }
```

The described method concludes the approach employed by the group to manipulate SVG files and project accurate information using components from Figma.

[13]Information about the function: https://developer.mozilla.org/en-US/docs/Web/API/Element/setAttribute

4.3.2 Outdated method of implementing OpenBridge design

During the development of the Electron app, alternative approaches to implementing OpenBridge elements were explored. The incorporation of elements into the application was facilitated through the utilization of the online library provided by OpenBridge. This method offers a convenient means of importing elements, albeit with a limitation of an outdated library version. Specifically, the library houses OpenBridge 2.0 elements, while the most recent and up-to-date elements employed by the group during the Figma design phase correspond to OpenBridge 4.0 elements.

Importing elements via the source tag represents an efficient approach, enabling production with reduced requirement for in-depth comprehension[14]. By locating the corresponding element ID within the library, the element can be imported as demonstrated in Snippet 17. This particular snippet exemplifies the importation of the ”ob-rudder-large” element.

Listing 17: Code snippet of importing element from online library

```
<head>
    <script> src="https://unpkg.com/openbridge-web-components@0.2.1"></script>
</head>

<div>
    <ob-rudder-large id = "largerudder" />
    <script src="../JS/ob-rudder.js"></script>
</div>
```

Snippet 18 demonstrates an approach for assigning values to the various properties of an element. This code is embedded within an HTML file and employs JavaScript within a script tag. By leveraging JavaScript, the values can be dynamically manipulated. The process of accessing and modifying the element is facilitated through the utilization of JavaScript.

Listing 18: Code snippet of JS for element movement for full version see largerudder.js in appendix J

```

const largerudder = document.getElementById('largerudder');

largerudder.clipAngle = 90;
largerudder.rudderAngle = Math.sin(t*(2*Math.PI)* .1)*20;
largerudder.rudderSetPointAngle = Math.sin(t*(2*Math.PI)* .1)*30;
largerudder.showSetPoint = 1;
largerudder.showPortStarboard = 1;
```

Figure 17 presents a flowchart delineating the step-by-step process of obtaining an element from the OpenBridge online library and ultimately transforming it into a functional component within an HTML document, which can subsequently be executed in the Electron environment.

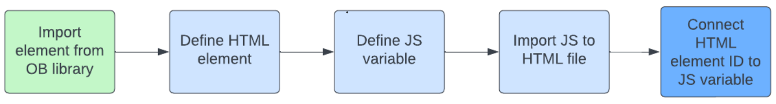

Import element from OB library
Define HTML element
Define JS variable
Import JS to HTML file
Connect HTML element ID to JS variable

Figure 17: Flowchart of the element from OpenBridge library to HTML

[14]OpenBridge online library: https://gitlab.com/openbridge/openbridge-web-components

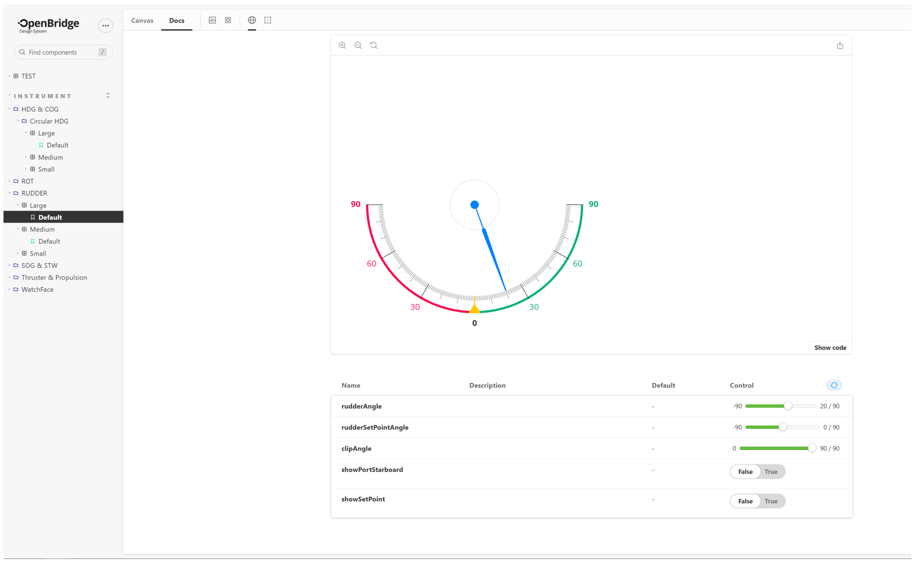
The decision to embed the JavaScript code within the div section of the HTML code, rather than separate it into a distinct script section, is a deliberate choice. In the approach depicted in Snippet 19, the values of different properties are directly incorporated into the HTML code. This strategy is particularly suitable for HTML pages with a smaller number of elements, as it helps mitigate potential clutter and complexity arising from excessive div sections.

Listing 19: Code snippet of HTML element movement for full version see rudder.html in appendix J

```

<div>
    <ob-rudder-large id = "largerudder"
    clipAngle = 90
    rudderAngle = 45
    rudderSetPointAngle = 45
    showSetPoint = 1
    showPortStarboard = 1
    />
</div>
```

The OpenBridge online Web Components demo serves as a valuable resource for identifying the distinct properties associated with various elements. A visual representation of the online demo is presented in Figure 18[15].



Name	Description	Default	Control
rudderAngle		-	-90 20 / 90
rudderSetPointAngle		-	-90 0 / 90
clipAngle		-	0 90 / 90
showPortStarboard		-	False True
showSetPoint		-	False True

Figure 18: OpenBridge online demo

This method proves to be straightforward, efficient, and user-friendly, its limitation is compromised by the presence of an outdated library, halting the seamless implementation of the newer, updated versions of various OpenBridge elements.

Snippet 20 exemplifies the integration of the top bar component into an HTML page. This process entails incorporating the top bar on each desired page. The "navigation.html" represents a simple HTML page that encompasses all the elements within the top bar, arranged in a row. This page is subsequently imported and positioned within the top bar section.

[15]OpenBridge online demo: https://openbridge.gitlab.io/openbridge-web-components/?path=/docs/instrument-rudder-large–default

Listing 20: Code snippet of topbar implementation for full version see html files in appendix J

```

    <div id="nav-placeholder"></div>
    <script>
        $.get("navigation.html", function(data){
            $("#nav-placeholder").replaceWith(data);
        });
    </script>
```

By implementing the script provided in Snippet 21, a functional mechanism is activated that accurately identifies the name of the current HTML window being accessed by the user. This function facilitates the addition of a class, which subsequently modifies the color of the top bar.

Listing 21: Code snippet of active-window function for full version see html files in appendix J

```

<script>

    document.addEventListener("DOMContentLoaded", function(){
    // get the current page URL
    var url = window.location.href;
    // get the ID of the link corresponding to the current page
    var currentPage = url.substring(url.lastIndexOf("/") + 1).replace(".html", "");
    var currentLink = document.getElementById(currentPage);
    // add the "active" class to the current link
    currentLink.classList.add("active");
    });

  </script>
```

5 Result

This chapter presents the results of the four parts of the project. First, the results from the Figma design process will be presented. Second, the results of the implementation of OpenBridge to an app in Electron. Third, an evaluation of the different functions and aspects of OpenBridge has to offer, and a comparison to systems that have and have not implemented it.

5.1 Figma design

In this section, we will present the results of the design process in Figma. In the design process, we focused on the usability, readability, and navigational flow aspects of the design.

5.1.1 Design Process

The design process consisted of the development of three initial drafts, which subsequently evolved into the final result. One of the milestones of the project was to assess the usability of OpenBridge, with a particular focus on the UI navigation components. The drafts vary in how users navigate within the UI. The changes were based on the feedback received from the client, supervisors, experienced outsiders in the maritime industry (appendix L), and our own iterative testing of the UI drafts using the prototype function, as elaborated in Section 4.2.1.

Upon achieving the final result, the UI design was further expanded to depict a more comprehensive system. This expansion aimed to capture the broader scope of functionalities and interactions within the OpenBridge environment. A visual representation of the design process, showcasing the progression from the initial draft to the ultimate final result, can be observed in Figure 19.

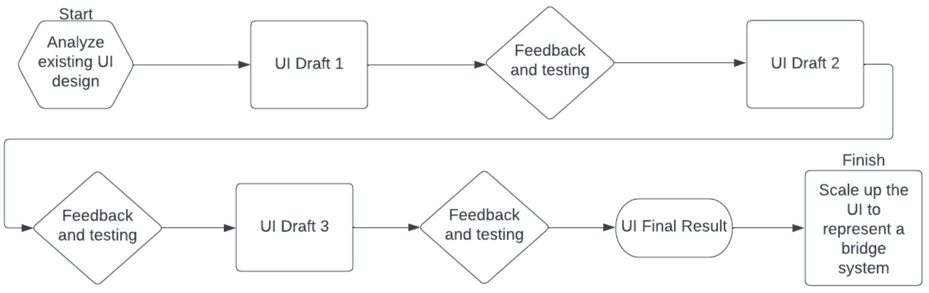


Figure 19: Design process, from the initial draft to the final result

5.1.2 Drafts

This section provides a presentation of the drafts made during the design process, highlighting the differences between them and the reasons behind the implemented changes.

Throughout the design process, three preliminary drafts were created prior to reaching the final UI design. The construction of both the UI drafts and the final outcome followed the method introduced in Section 4.2.1. The drafts vary in terms of user navigation within the UI. It is important to note that the technical components incorporated in the UI, such as azimuths, thrusters, engines, and other instruments, are purely illustrative and should not be considered functional elements.

The subsequent sections will go into the specific details of each draft and the final UI design, emphasizing the modifications.

5.1.2.1 Icons description

The icons incorporated in both the drafts and the final UI design are sourced from the OpenBridge library and adhere to its corresponding guidelines. The OpenBridge library consists of various components, including top bars and other elements, all of which feature icons. In most instances, these icons possess interactive functionalities when clicked upon, such as triggering a pop-up window or directing the user to a new page. Table 4 presents an overview of the icons utilized in the drafts as well as the final UI result.

SYMBOLS			
Nr	Symbol	ID	Description
1	⠿	Apps	Opens an App menu pop-up containing overlaying applications.
2	☀	Dimming	Opens brilliance control pop-up, which the user can control screen brightness and change between day, night, and dusk mode.
3	👤	User	Opens a user login pop-up where the user can log in with a personalized account.
4	💬	Notification	Opens a notification pop-up where the latest notifications are displayed.
5	🔔 2	Alert	Opens an active alerts pop-up that shows the latest alerts in the system
6	🔇	Mute	Mutes the notifications from alerts.
7	ACK	Acknowledge	Acknowledges the latest active alerts that are shown in the latest alerts tab.
8	≡	menu	Opens up an expanded sidebar that gives access to subpages within a sub-application.
9	⚠ 9	Alert List	Opens up a full window overview of the alert list.
9	▦	Screen Control	Opens up screen control window where the user can alter the screen layouts.
10	✢	Sub menu icon	Navigates the user to the sub-page of an application, with different icons for different sub-pages. Such as Propulsion, Power, Engines, Machinery, Cargo, System, Safety, and Thrusters.

Table 4: Overview of icons

5.1.2.2 Draft V0.1

The first initial draft (V0.1) was based on the given example from the client, shown in Figure 10. It was used as a starting point for what the example UI looked like using OpenBridge components.

The main navigation method in this draft involves utilizing the sub-application overview located on the IAS overview screen, as illustrated in Figure 20. By using the sub-application buttons, users were able to navigate to the desired sub-application within the broader IAS application.

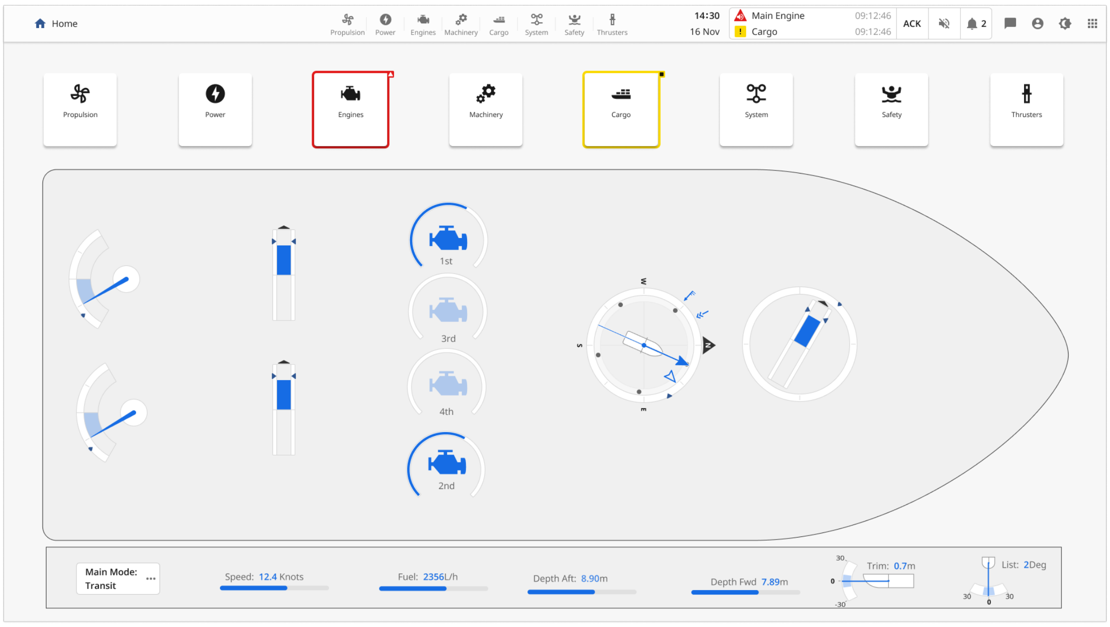


Figure 20: Draft V0.1 IAS overview page

Once inside a specific application, the principle of navigation relied upon the utilization of the top bar component, which displayed distinct application icons, as depicted in Figure 21. The addition of the top bar component aimed to provide users with access to other applications without requiring them to navigate through the overview page. To return to the overview page, users could simply click on either the home icon or the previous page icon.

It is important to note that the top bar component was an inclusion made by the group, utilizing the building blocks library introduced in Section 4.2.1. This custom top bar component was not a part of the standard top bar components available in the OpenBridge library.

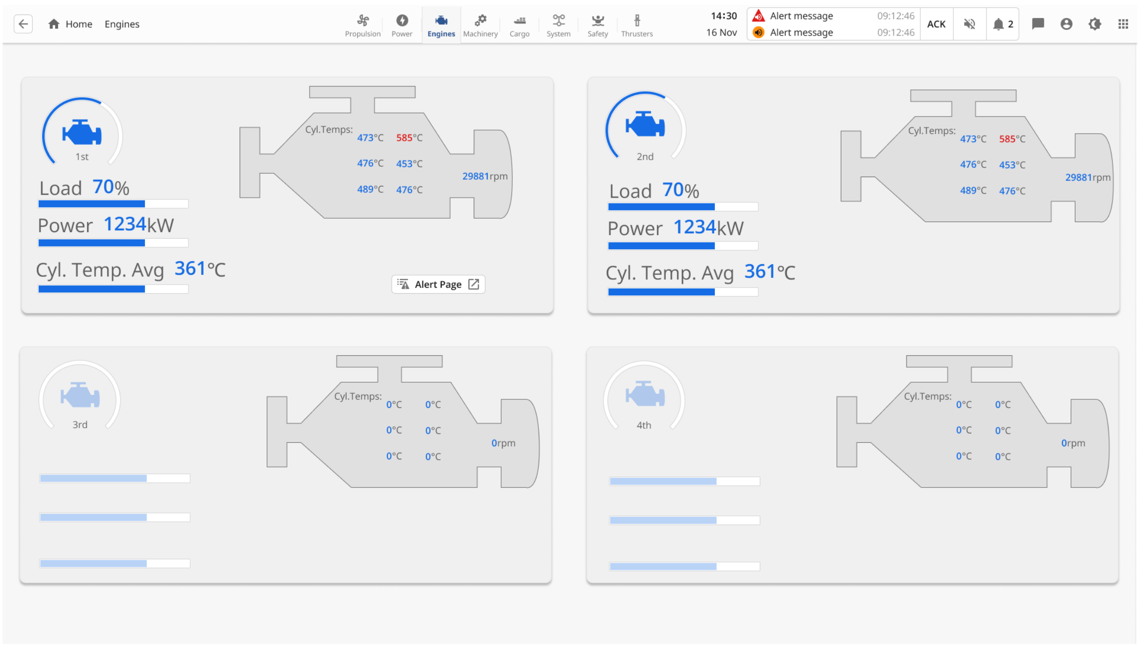


Figure 21: Example of a sub-application in draft V0.1

5.1.2.3 Draft V0.2

In the second draft (V0.2), see Figure 22, a modification was implemented by replacing the previously mentioned top bar component, as discussed in Section 5.1.2, with a sidebar. This change was motivated by the fact that the sidebar was readily available within the OpenBridge library. The sidebar fulfills the same purpose as the former top bar component, but with the added advantage of enabling users to navigate to various sub-pages, as illustrated in Figure 24 and Figure 23. Given that sub-applications often encompass multiple underlying sub-pages, the inclusion of the sidebar allows users to directly access these sub-pages. This functionality was deemed crucial within the UI, as it facilitates seamless navigation to sub-pages without having to traverse through the sub-application.

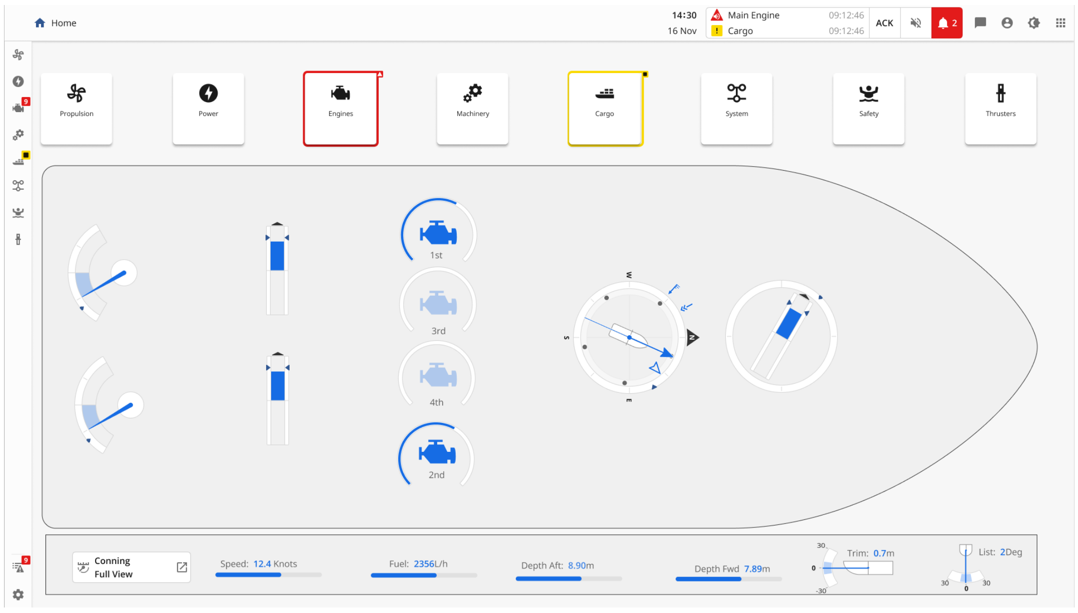


Figure 22: Draft V0.2 IAS overview page

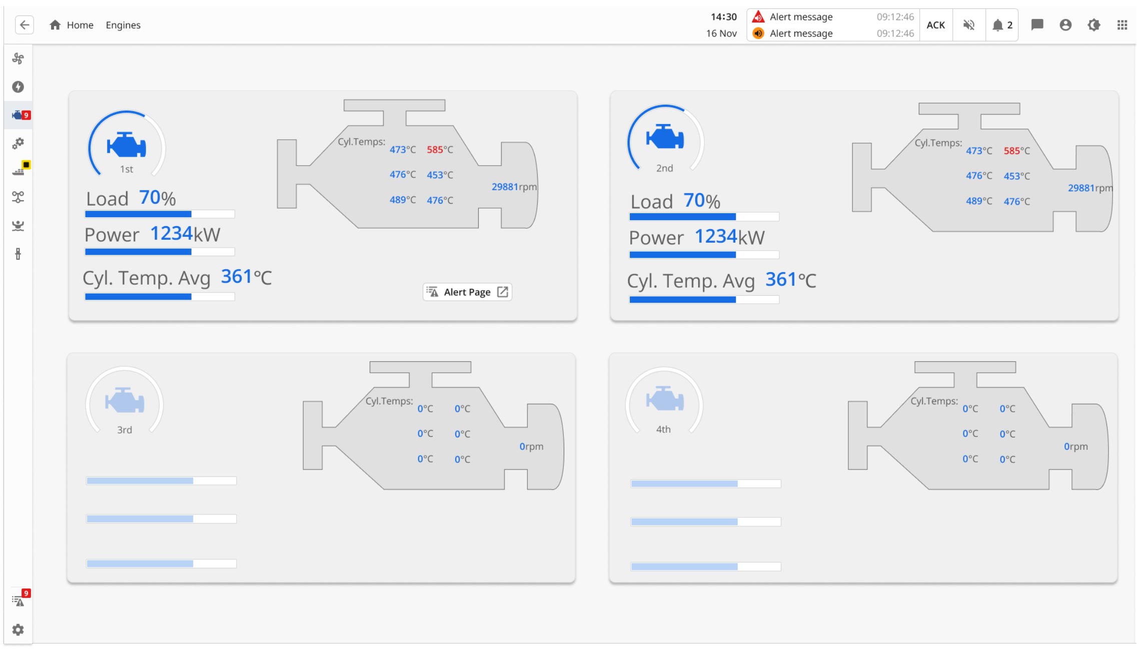


Figure 23: Example of a sub-application in draft V0.2

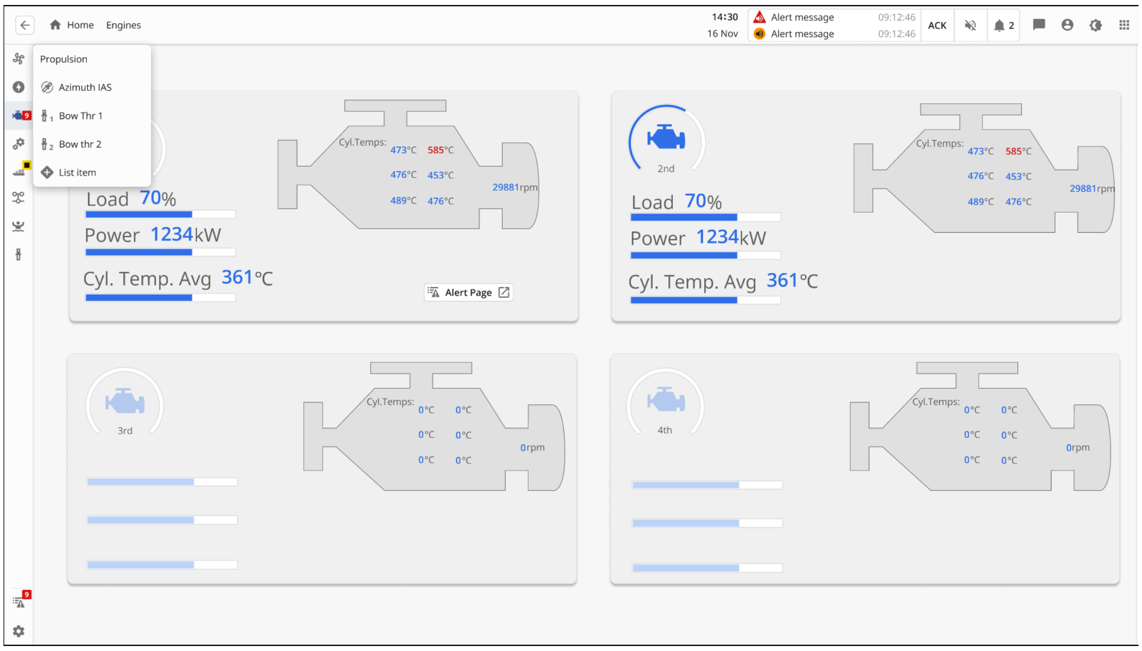


Figure 24: Example of navigating to sub-pages within a sub-application

5.1.2.4 Draft V0.3

In the third draft (V0.3), see Figure 25, the primary modification is the inclusion of a menu icon positioned in the top left corner of the top bar, see Table 4. This addition was based on private communication with Kjetil Nordby, as documented in the appendix L. It was emphasized that the menu icon should remain accessible regardless of the specific page the user is currently navigating, as seen In Figure 26. Upon clicking the menu icon located in the top left corner, an expanded sidebar menu is displayed. This sidebar menu grants users access to the diverse sub-pages associated with the sub-applications, as seen in Figure 27.

The incorporation of the menu icon holds significant importance within the design, as it ensures that the sub-application remains easily accessible to users at all times. Additionally, the sidebar menu also provides access to the alert list, as depicted in Table 4.

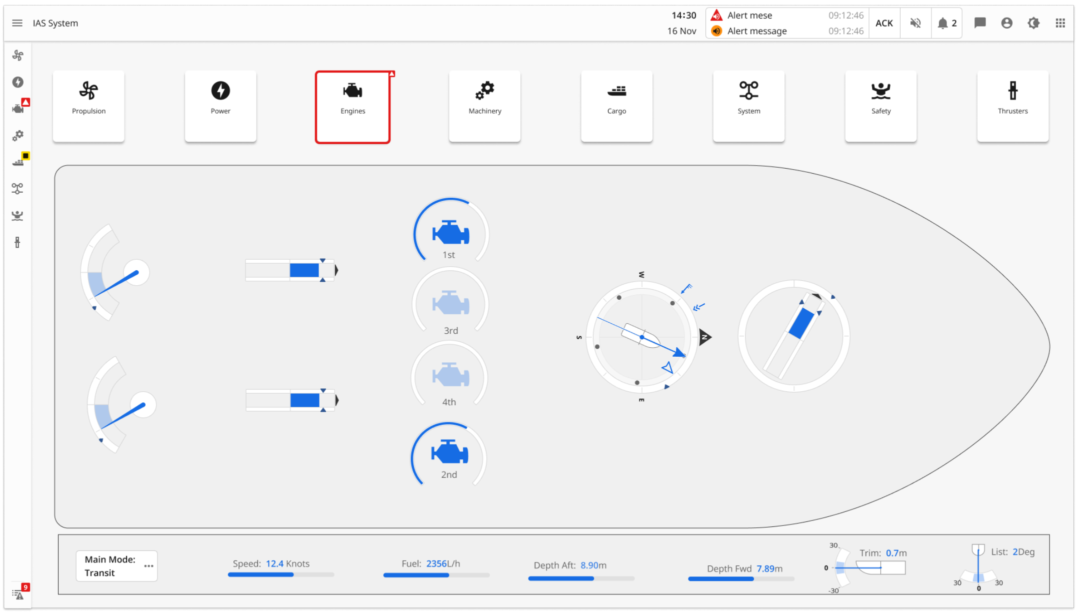


Figure 25: Draft V0.3 IAS overview page

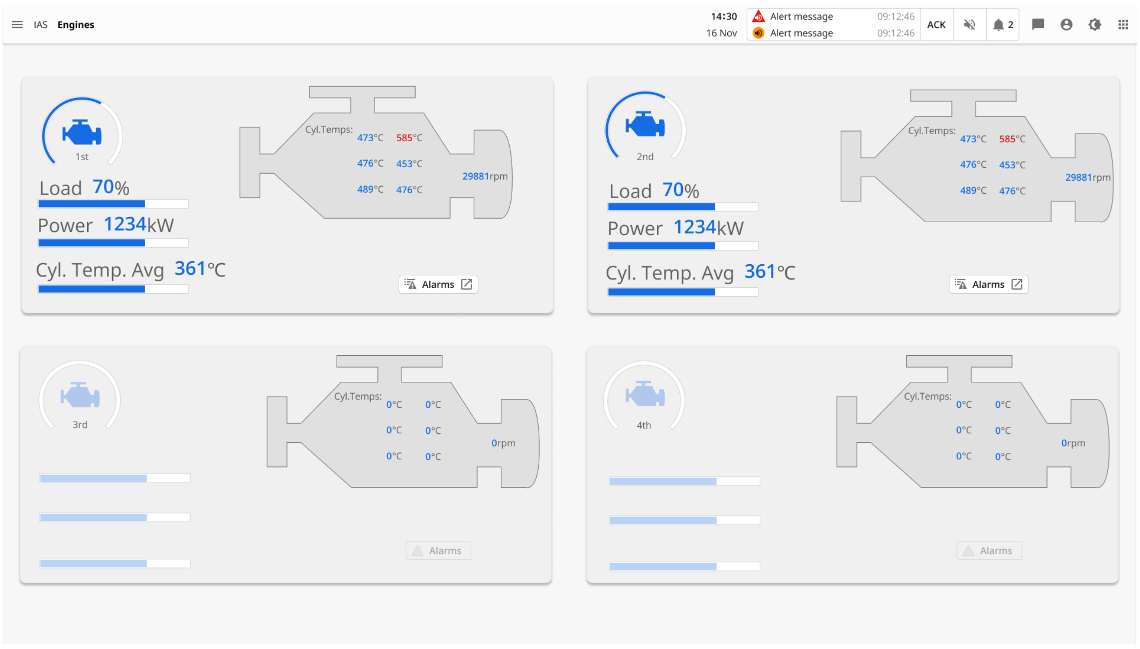


Figure 26: Example of a sub-application in draft V0.3

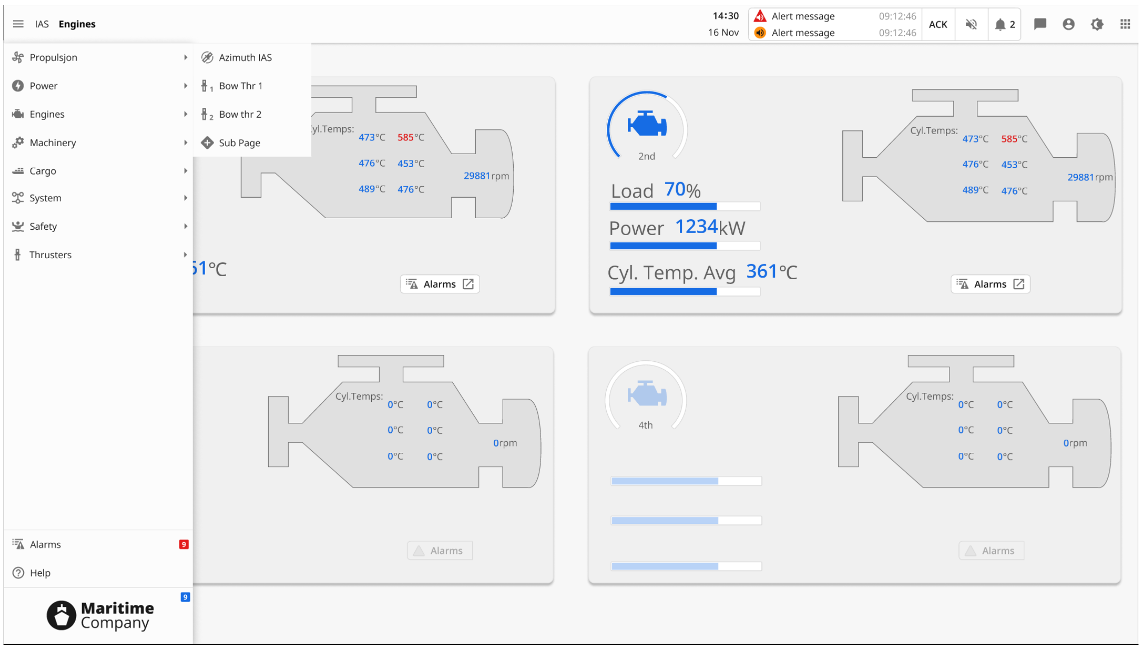


Figure 27: Example of navigating to sub-pages within a sub-application

5.1.3 Final UI design

The UI is organized in a structured and user-friendly manner, allowing users to navigate through different sections and applications. Whether the user is a machinist requiring detailed IAS system information or a navigator seeking Conning, ECDIS, or Radar applications. Throughout the design process, we have taken into account the valuable feedback received from users and domain experts, ensuring that the final UI design reflects their requirements and preferences.
In the following sections, it will be highlighted key features, components, and interactions.

5.1.3.1 Structure of the final design

The UI is organized in a hierarchical structure, with multiple layers of applications and sub-applications. At the top level, there are three primary applications, each with its own set of underlying sub-applications. These sub-applications, in turn, consist of additional sub-pages. The UI dynamically adapts based on the main application being accessed by the user. Notably, both the sidebar and top bar components vary depending on the active application.

The overarching applications within the UI encompass the IAS, which serves as the central system, alongside independent applications such as screen control, cameras, calendar, and others. Additionally, there are navigation applications. The structural arrangement of the UI, with its various applications and their relationships, is visually depicted in Figure 28.

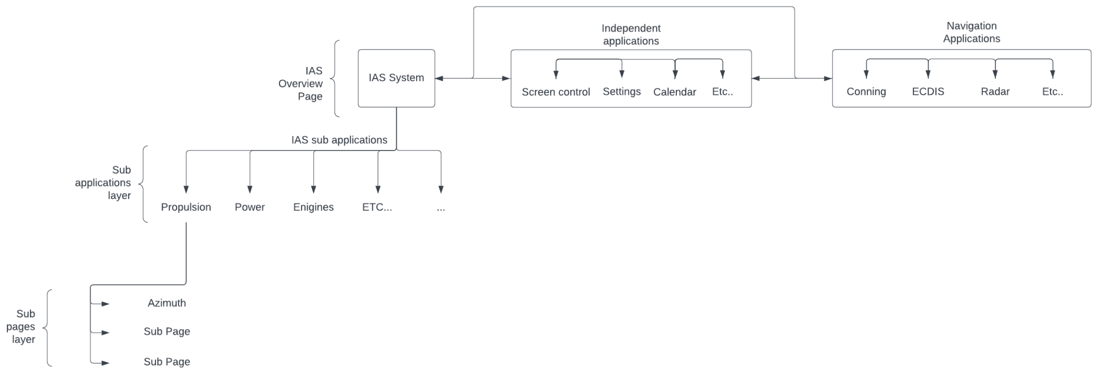
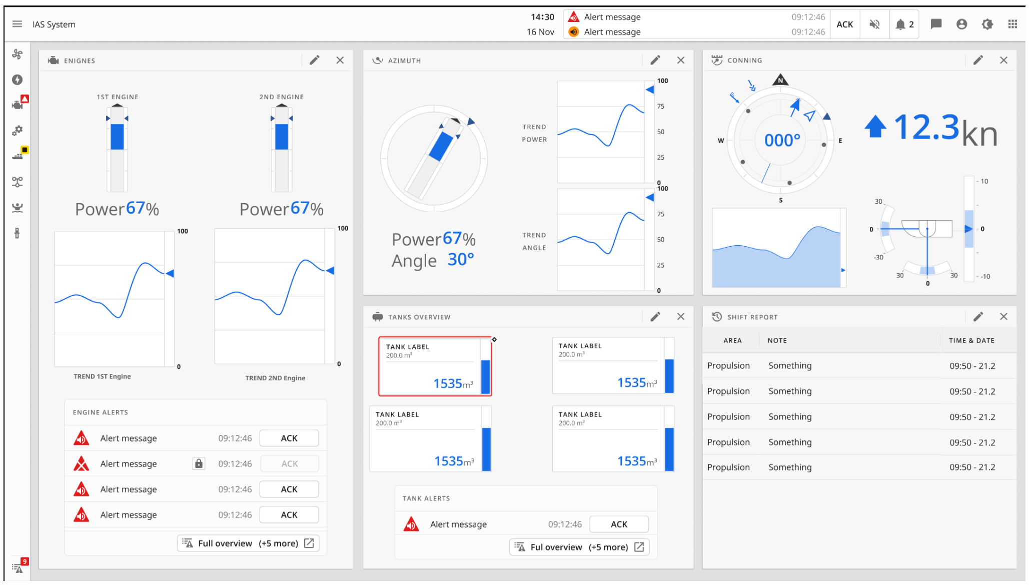


Figure 28: Illustration of the structure of the Final UI design

5.1.3.2 Final design IAS overview page

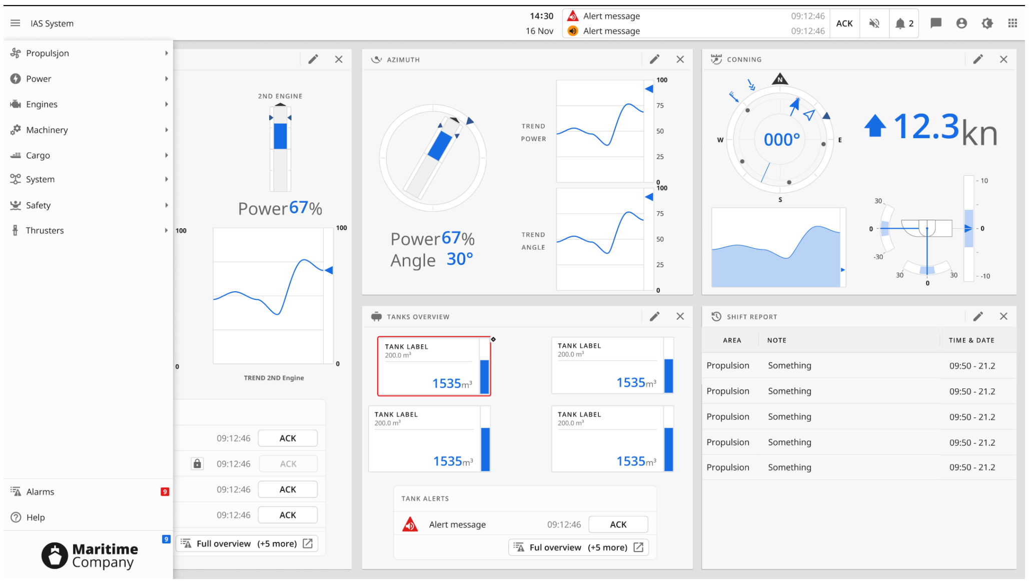
The UI encompasses a system that replicates a maritime application on a bridge, as introduced in Section 5.1.3. The final design, see Figure 29, adheres to the same navigational structure as the third draft, utilizing a sidebar for navigation to the desired sub-applications, as illustrated in Figures 30 and 31.

Significant updates were made to the content of the IAS overview page. The sub-application overview, which was initially introduced in Section 5.1.2, was removed based on the rationale that it was redundant since the sub-applications were already accessible through the sidebar. This decision resulted in more available space on the IAS overview page for user-related information. From private communication with Kjetil Nordby, appendix L, it was emphasized that the IAS overview page should contain crucial information for the user. Accordingly, the final design incorporated trends displaying historical values of the components, providing numeric values instead of just graphical representations or non-numeric indicators. Additionally, a shift report section was added to the overview page, as notes from previous users could offer valuable insights to the current user. Alarms are also more prominently displayed for the currently active sub-application on the IAS overview page.



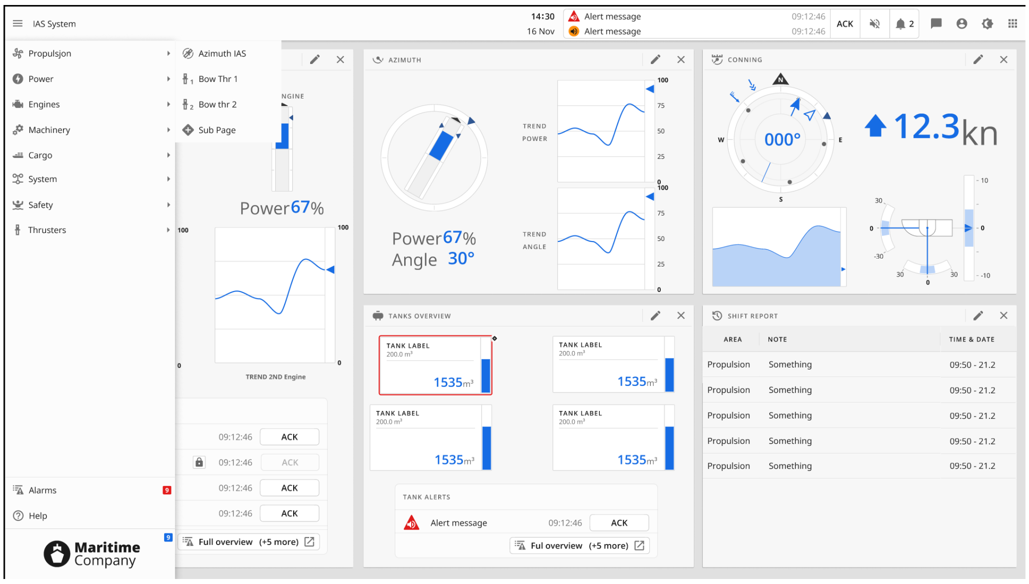
AREA	NOTE	TIME & DATE
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2

Figure 29: Final design IAS Overview



AREA	NOTE	TIME & DATE
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2

Figure 30: Final design Sidebar navigation



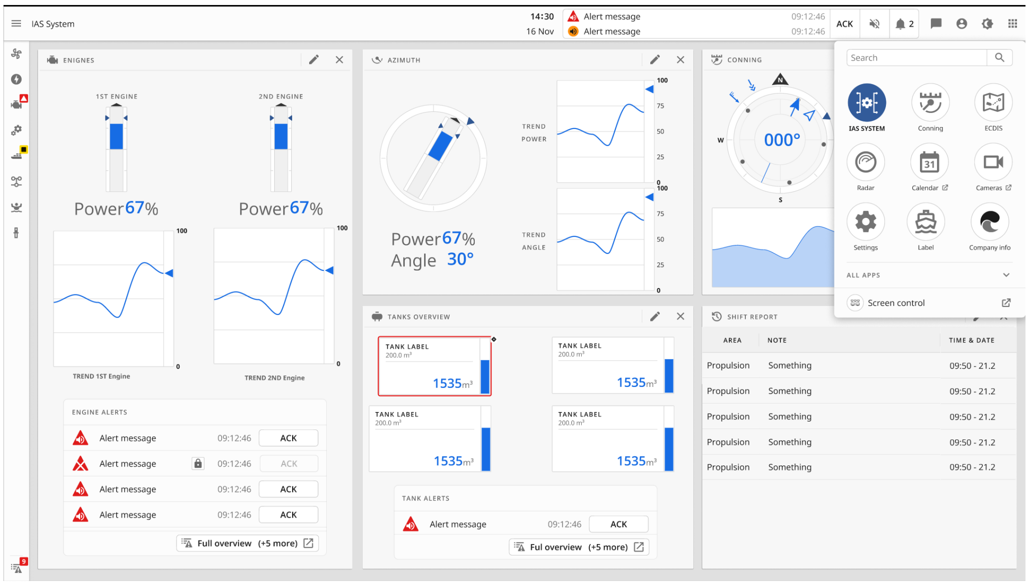
AREA	NOTE	TIME & DATE
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2

Figure 31: Final design Subpage navigation

Navigating between the Main applications

To navigate between the IAS system and the navigation applications, the user can utilize the apps menu icon, as introduced in Table 4. By clicking on this icon, an applications menu is revealed, as illustrated in Figure 32. Through this menu, the user gains the ability to navigate to various overlaying applications within the UI, including the IAS system, independent applications, and navigational applications.

To access the navigation applications specifically, the user can click on either Conning, shown in Figure 33, ECDIS, shown in Figure 34, or Radar, shown in Figure 35. It's important to note that the menu app icon remains readily available at the top bar, independent of the specific application page the user is currently controlling. This ensures consistent access to the applications menu throughout the UI.



ENGINE ALERTS

Alert message	09:12:46	ACK
Alert message	09:12:46	ACK
Alert message	09:12:46	ACK
Alert message	09:12:46	ACK

Full overview (+5 more)

AREA	NOTE	TIME & DATE
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2

Figure 32: The application menu popup

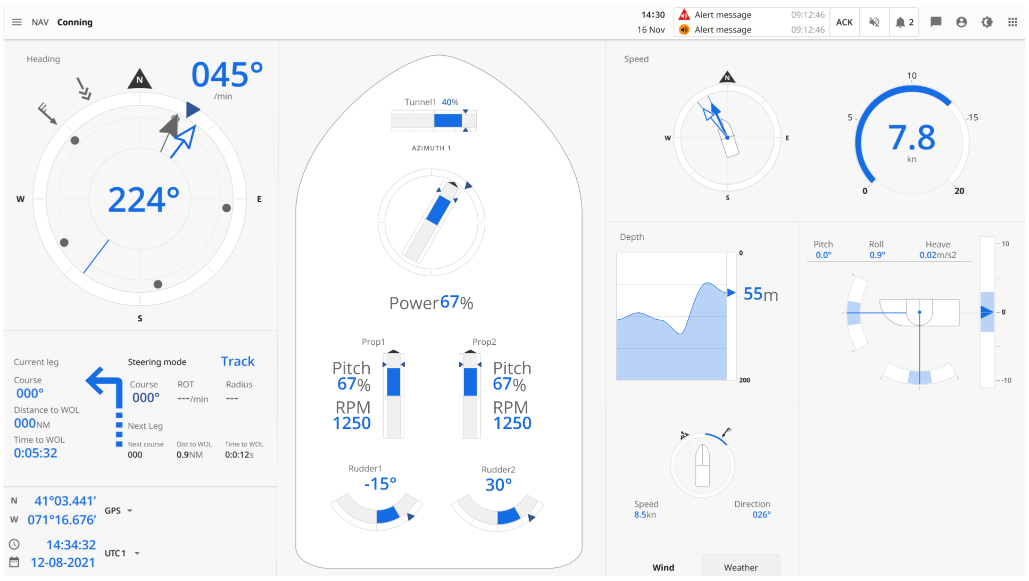


Figure 33: Final design Conning application

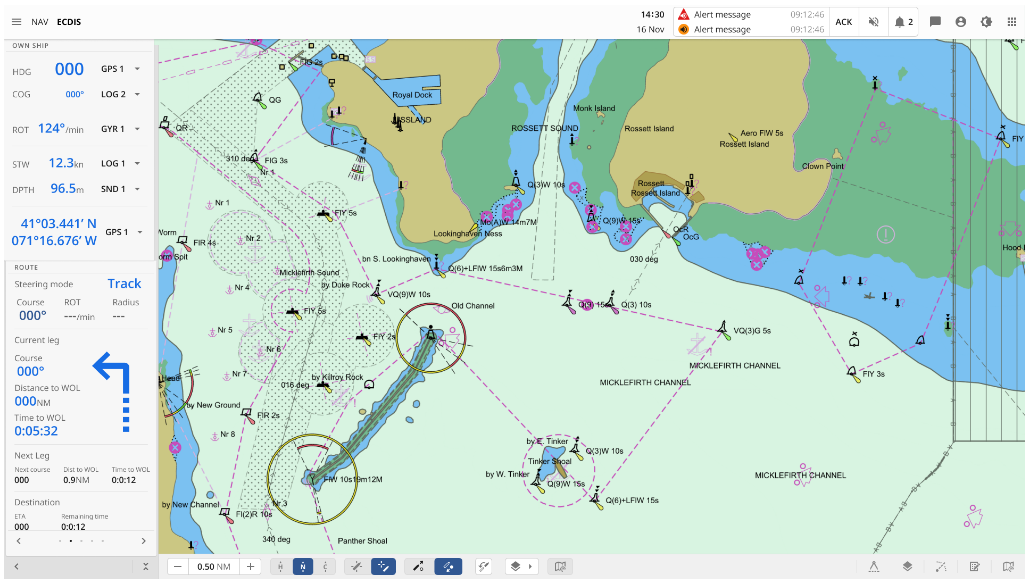


Figure 34: Final design ECDIS application

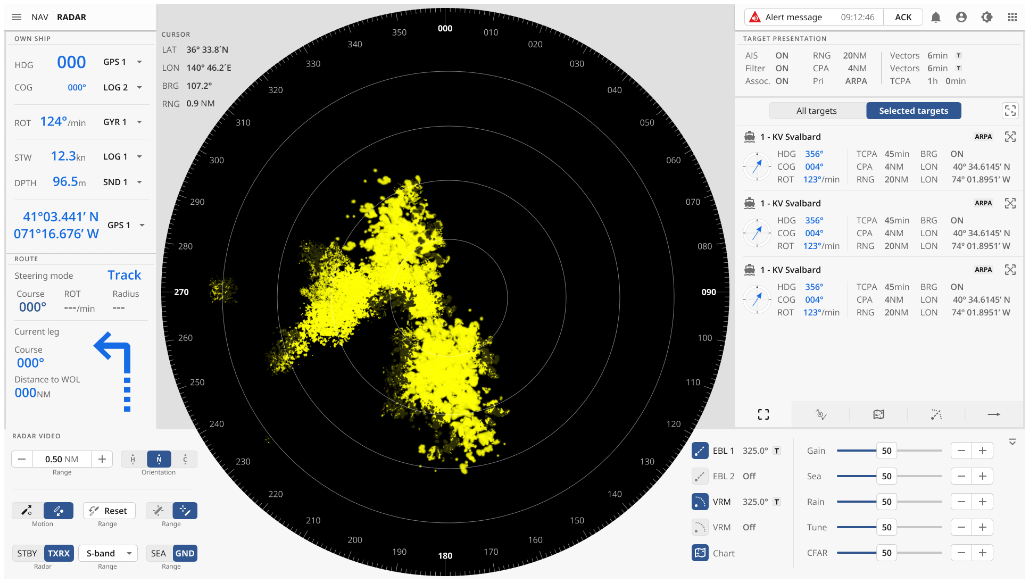


Figure 35: Final design RADAR application

The design incorporates a dynamic sidebar that adjusts depending on whether the user is in the IAS application or the navigation application. When the user is navigating within the IAS system, the sidebar displays the IAS sub-application, as illustrated in Figure 30. When the user is navigating the Conning, ECDIS, or Radar applications, the sidebar switches to display the respective navigation application, as demonstrated in Figure 36.

This dynamic sidebar ensures that the relevant sub-applications are easily accessible to the user, providing a streamlined navigation experience based on the context of their current activities within the UI.

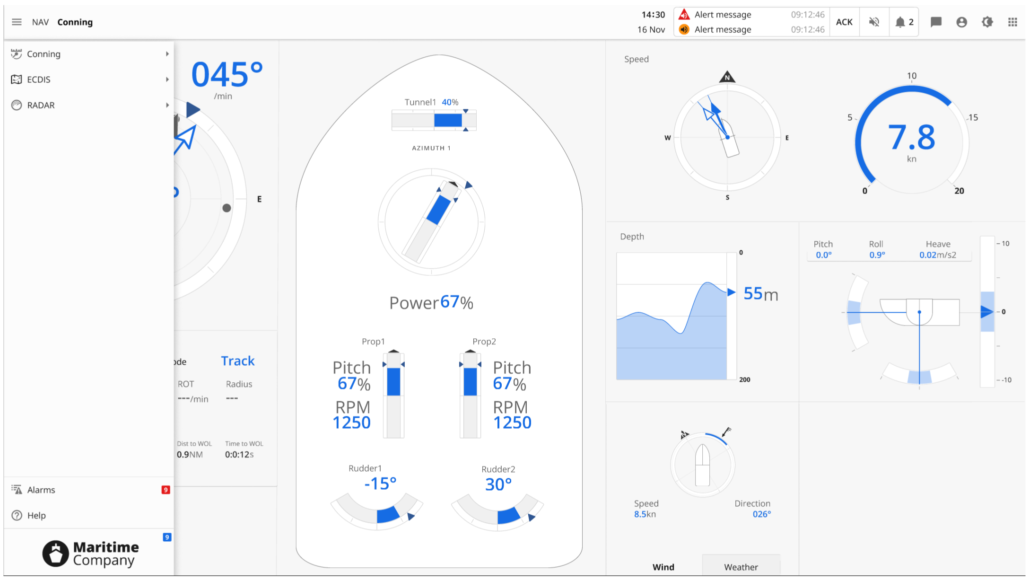


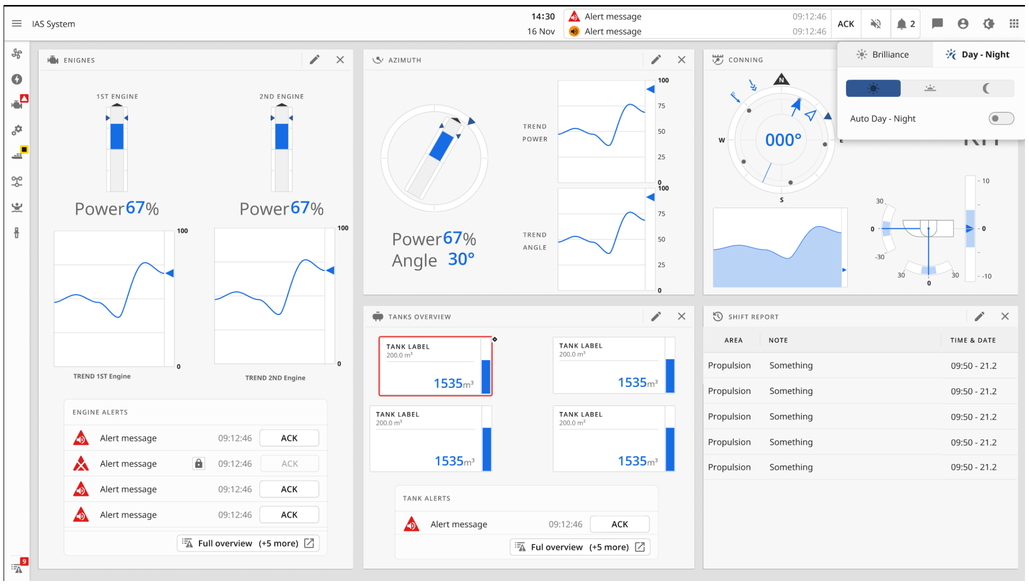
Figure 36: Exposed sidebar in navigation applications

Dimming panel

The inclusion of a dimming panel holds significant importance in the bridge UI of maritime bridge applications. Insights gathered from personal communication with Ralf Eirik Grønning, appendix L, emphasized the significance of the dimming function, as it emerged as one of the most frequently used features during ship operations.

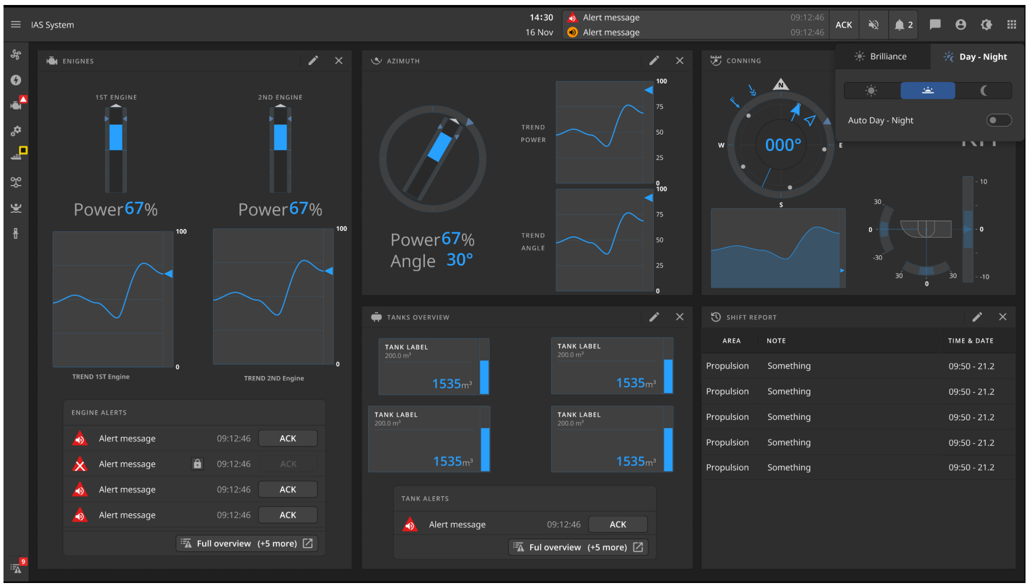
By clicking on the dimming icon, as presented in Table 4, a pop-up window is triggered, enabling the user to adjust the screen's brightness and toggle between different modes: dusk, night, and day in the OpenBridge environment. The corresponding visual representations of these modes can be seen in Figures 37, 38, and 39.

The provision of this dimming functionality allows users to adapt the screen's brilliance according to varying lighting conditions, ensuring optimal visibility and usability of the UI in different environmental settings.



AREA	NOTE	TIME & DATE
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2

Figure 37: IAS overview page in day mode



Alert message 09:12:46
Alert message 09:12:46

ENGINE ALERTS

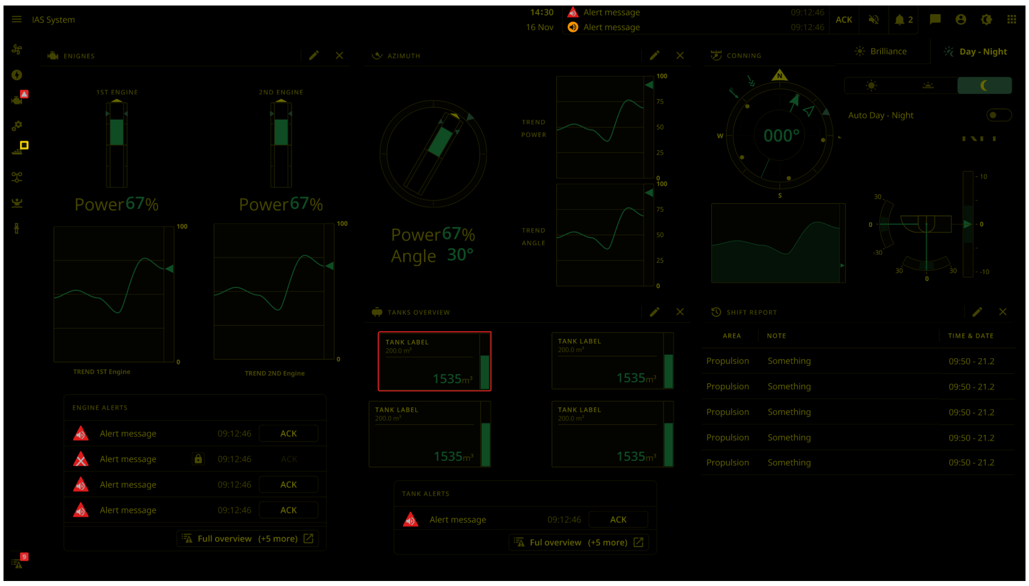
Alert message 09:12:46 ACK
Alert message 09:12:46 ACK
Alert message 09:12:46 ACK
Alert message 09:12:46 ACK
Full overview (+5 more)

TANK ALERTS

Alert message 09:12:46 ACK
Ful overview (+5 more)

AREA	NOTE	TIME & DATE
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2

Figure 38: IAS overview page in dusk mode



Alert message 09:12:46
Alert message 09:12:46

ENGINE ALERTS

Alert message 09:12:46 ACK
Alert message 09:12:46 ACK
Alert message 09:12:46 ACK
Alert message 09:12:46 ACK
Full overview (+5 more)

TANK ALERTS

Alert message 09:12:46 ACK
Ful overview (+5 more)

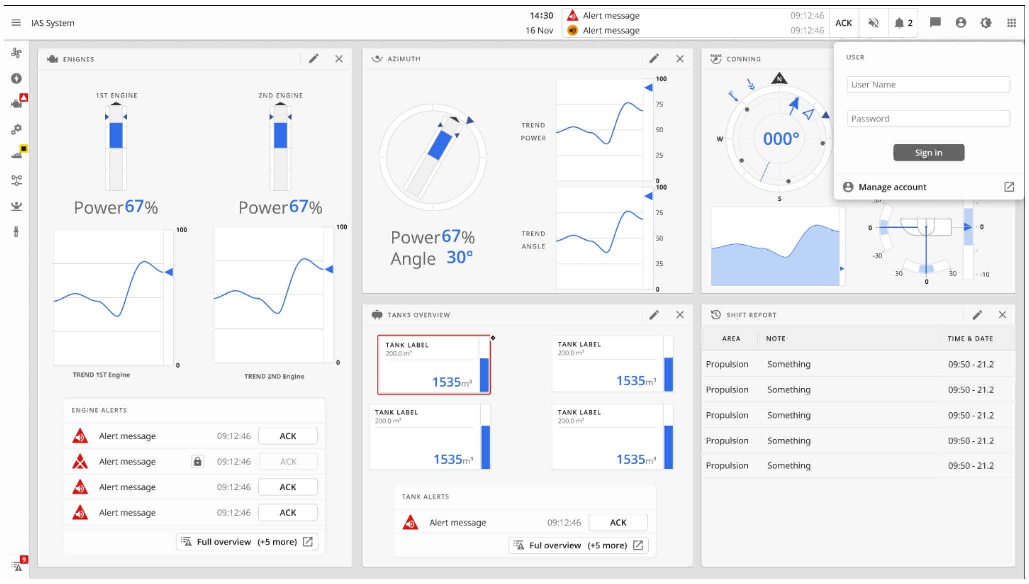
AREA	NOTE	TIME & DATE
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2

Figure 39: IAS overview page in night mode

User login

The user login panel serves the purpose of displaying the login status and facilitating multi-user logins within the maritime application. Insights obtained from personal communication with Ralf Erik Grønning, appendix L, highlighted the significance of personalized accounts in maritime applications. Individual users often have their own preferences for screen layouts and other personalized settings.

By clicking on the user icon, as indicated in Table 4, the user login window is triggered, enabling users to log in with their respective credentials. This login window is illustrated in Figure 40.

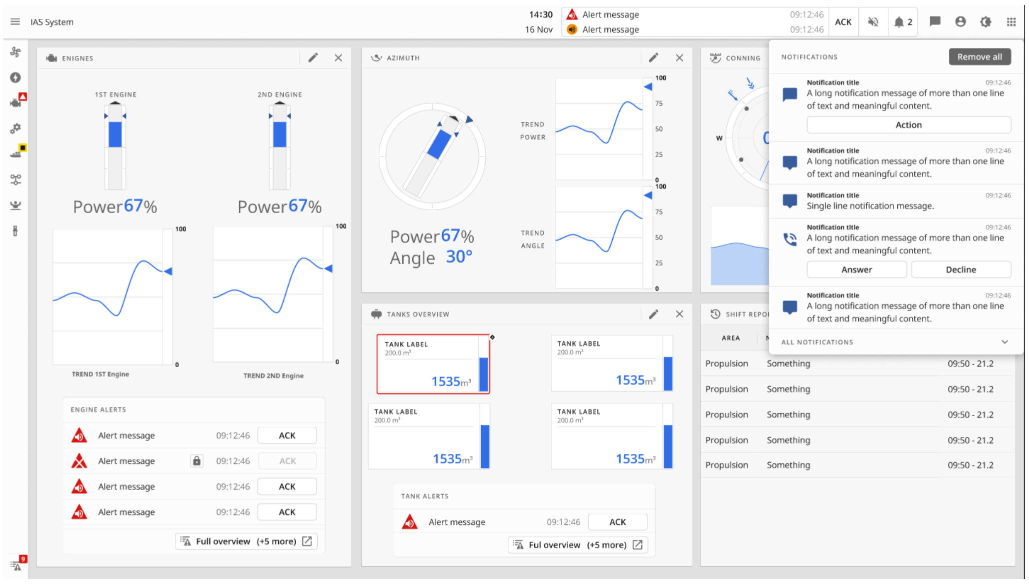


AREA	NOTE	TIME & DATE
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2

Figure 40: User Login popup in the final UI design

Notifications

The notifications panel serves as a centralized hub for accessing various non-alert notifications, such as messages or calls from other users, within the application. Clicking on the notification icon, as indicated in Table 4, triggers the notification window, which is depicted in Figure 41.



Alert message 09:12:46
Alert message 09:12:46

Notification title 09:12:46
A long notification message of more than one line of text and meaningful content.

Notification title 09:12:46
A long notification message of more than one line of text and meaningful content.

Notification title 09:12:46
Single line notification message.

Notification title 09:12:46
A long notification message of more than one line of text and meaningful content.

Notification title 09:12:46
A long notification message of more than one line of text and meaningful content.

ENGINE ALERTS

Alert message	09:12:46	ACK
Alert message	09:12:46	ACK
Alert message	09:12:46	ACK
Alert message	09:12:46	ACK

TANK ALERTS

Alert message	09:12:46	ACK

AREA		
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2

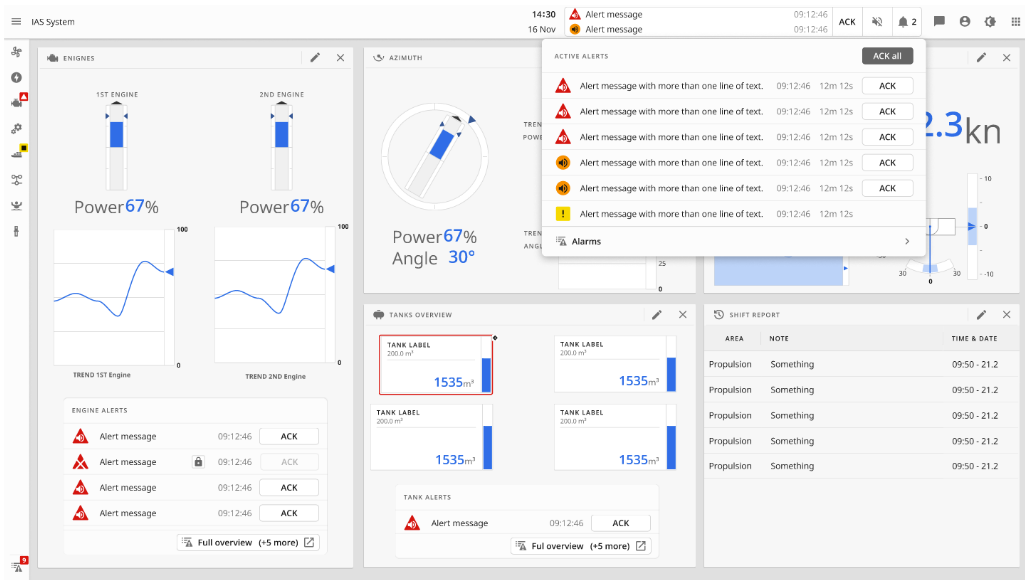
Figure 41: Notification popup in the final UI design

Alerts

Alerts and alarm handling are fundamental aspects of UI design in maritime applications. The top bar of the interface displays the two most recent alerts, allowing users to quickly access critical information. Clicking on the alert icon, as introduced in Table 4, triggers a pop-up window that provides more detailed information about recent alerts, as depicted in Figure 42.

For access to the complete alert list, users can click on the alert list icon, as shown in Figure 30, either within the sidebar or within the alerts pop-up. This grants users the ability to view and manage the full range of alerts and alarms in a dedicated interface.

By prominently featuring alerts in the top bar and providing easy access to the complete alert list, the UI design ensures that users can effectively monitor and respond to critical events and alarms, thus enhancing the safety and efficiency of maritime operations.



Alert message 09:12:46
Alert message 09:12:46

Alert message with more than one line of text.	09:12:46	12m 12s	ACK
Alert message with more than one line of text.	09:12:46	12m 12s	ACK
Alert message with more than one line of text.	09:12:46	12m 12s	ACK
Alert message with more than one line of text.	09:12:46	12m 12s	ACK
Alert message with more than one line of text.	09:12:46	12m 12s	ACK
Alert message with more than one line of text.	09:12:46	12m 12s	

Alert message	09:12:46	ACK
Alert message	09:12:46	ACK
Alert message	09:12:46	ACK
Alert message	09:12:46	ACK

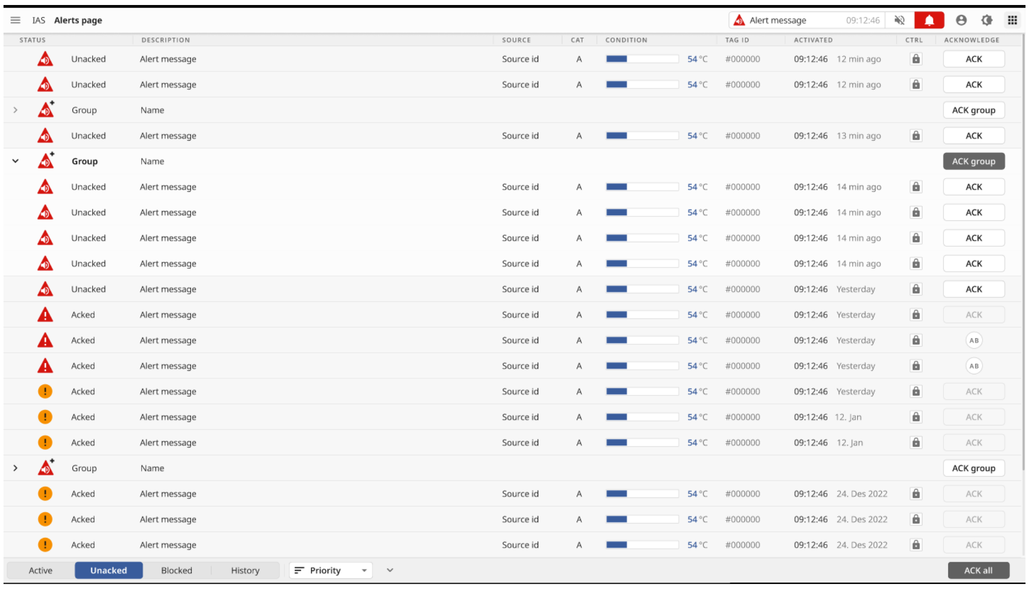
Alert message	09:12:46	ACK

AREA	NOTE	TIME & DATE
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2
Propulsion	Something	09:50 - 21.2

Figure 42: Notification popup in the final UI design

The alerts page within the application provides a view of all alerts, as depicted in Figure 43. This page serves as a centralized hub for monitoring and managing alerts. Users have the flexibility to filter the displayed alerts based on different criteria, including active alerts, unacknowledged alerts, blocked alerts, and the history of previous alerts.

Moreover, the alert list can be organized and sorted based on priority or groupings. For instance, sub-applications within the IAS system can be assigned to their respective groups in the alert list. This grouping allows for better organization and prioritization of alerts.


IAS **Alerts page**

Alert message 09:12:46

STATUS	DESCRIPTION	SOURCE	CAT	CONDITION	TAG ID	ACTIVATED	CTRL	ACKNOWLEDGE
Unacked	Alert message	Source id	A	54 °C	#000000	09:12:46 12 min ago	🔒	ACK
Unacked	Alert message	Source id	A	54 °C	#000000	09:12:46 12 min ago	🔒	ACK
Group	Name							ACK group
Unacked	Alert message	Source id	A	54 °C	#000000	09:12:46 13 min ago	🔒	ACK
Group	Name							ACK group
Unacked	Alert message	Source id	A	54 °C	#000000	09:12:46 14 min ago	🔒	ACK
Unacked	Alert message	Source id	A	54 °C	#000000	09:12:46 14 min ago	🔒	ACK
Unacked	Alert message	Source id	A	54 °C	#000000	09:12:46 14 min ago	🔒	ACK
Unacked	Alert message	Source id	A	54 °C	#000000	09:12:46 14 min ago	🔒	ACK
Unacked	Alert message	Source id	A	54 °C	#000000	09:12:46 Yesterday	🔒	ACK
Acked	Alert message	Source id	A	54 °C	#000000	09:12:46 Yesterday	🔒	ACK
Acked	Alert message	Source id	A	54 °C	#000000	09:12:46 Yesterday	🔒	AB
Acked	Alert message	Source id	A	54 °C	#000000	09:12:46 Yesterday	🔒	AB
Acked	Alert message	Source id	A	54 °C	#000000	09:12:46 Yesterday	🔒	ACK
Acked	Alert message	Source id	A	54 °C	#000000	09:12:46 12. Jan	🔒	ACK
Acked	Alert message	Source id	A	54 °C	#000000	09:12:46 12. Jan	🔒	ACK
Group	Name							ACK group
Acked	Alert message	Source id	A	54 °C	#000000	09:12:46 24. Des 2022	🔒	ACK
Acked	Alert message	Source id	A	54 °C	#000000	09:12:46 24. Des 2022	🔒	ACK
Acked	Alert message	Source id	A	54 °C	#000000	09:12:46 24. Des 2022	🔒	ACK

Figure 43: Notification popup in the final UI design

Screen control

The screen control window plays a crucial role in allowing users to customize the layout of multiple screens in maritime bridge solutions. Each user, whether a machinist or a navigator, may have their own specific preferences regarding the arrangement and content of the screens. Machinists may require screens displaying information from the IAS system, while navigators may prioritize Conning, Radar, and ECDIS pages.

To access the screen control window, users can navigate to the application menu as described in section 5.1.3. By clicking on the screen control icon, a dedicated window, depicted in Figure 44, will appear. Within this window, users can modify and adjust the layout of the screens according to their individual requirements and preferences.

This feature allows users to tailor the display and organization of information, ensuring that they have easy access to the most relevant data and applications based on their specific roles and tasks.

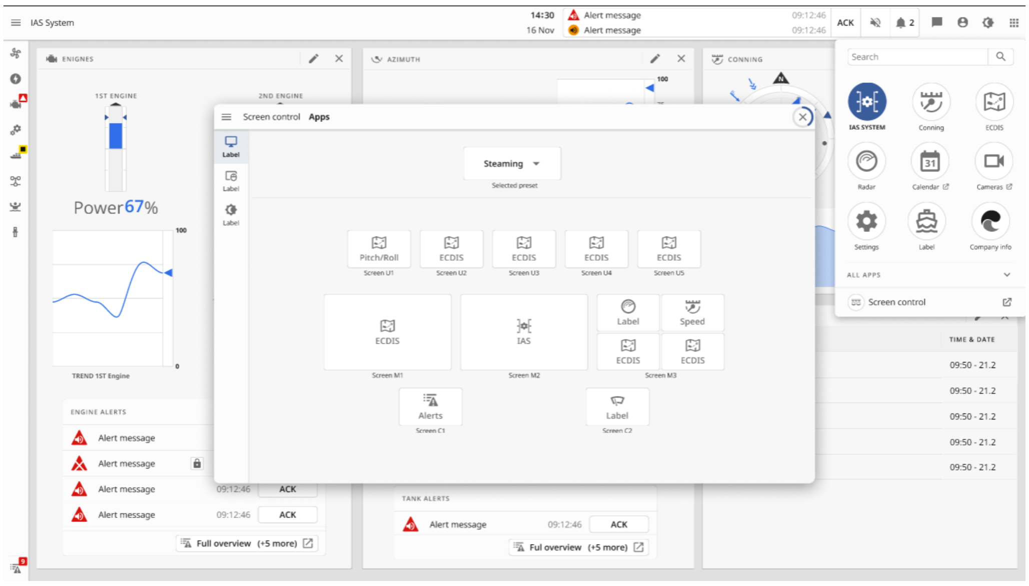


Figure 44: The screen control window

When configuring the screens, the support screens are typically designated to display a subset of information from a particular application. For instance, a support screen can host specific sections or components of the Conning application. On the other hand, the main screens can either accommodate entire applications or be divided into multiple support screens.

To assign an application to a support screen, users can click on the desired screen, triggering a pop-up window. Within this pop-up, a list of available applications that can be hosted on the screen is presented, as illustrated in Figure 45. Users can then select the appropriate application that they wish to assign to the support screen, as illustrated in Figure 46.

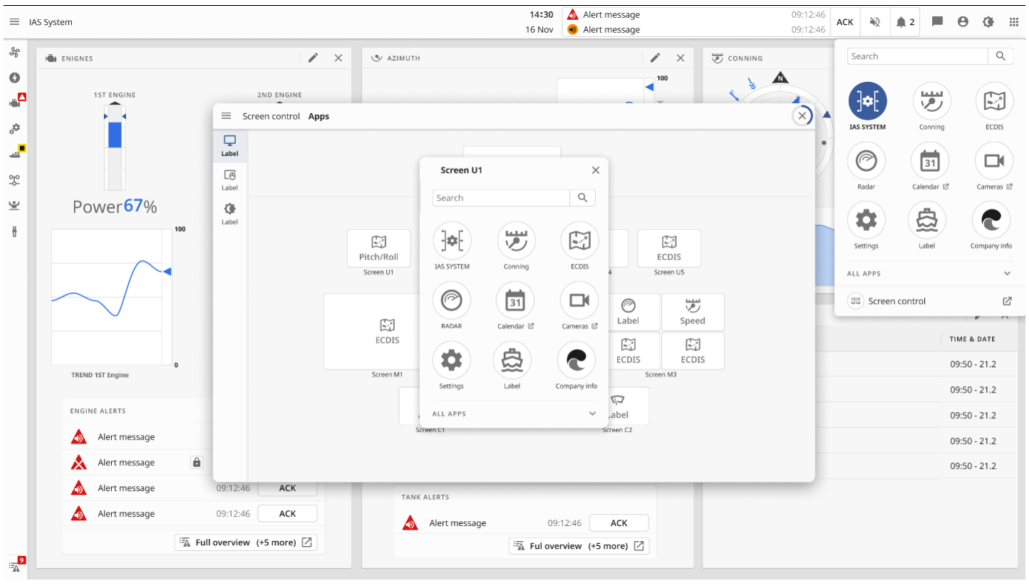


Figure 45: Example of choosing an application the screen will host

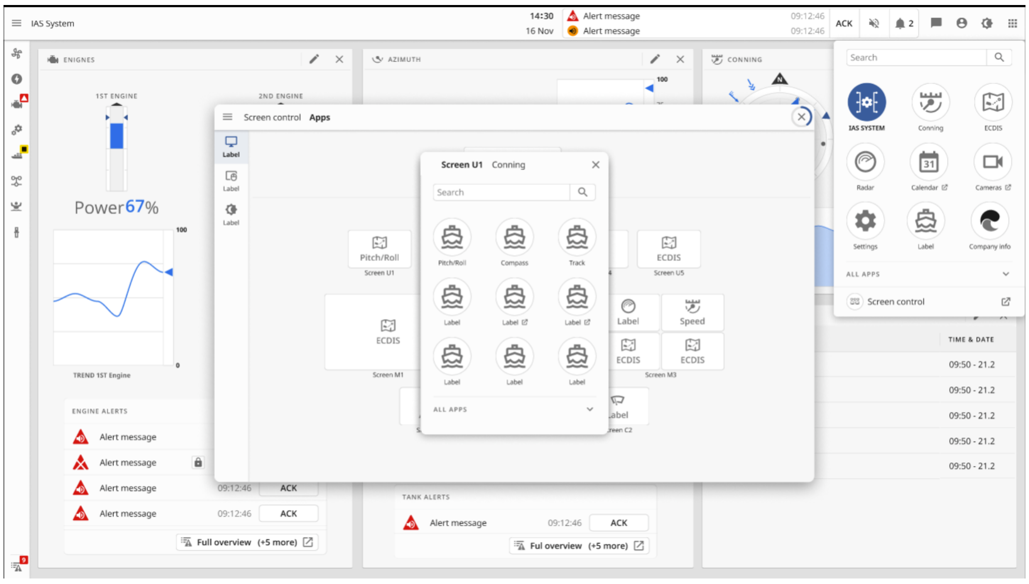


Figure 46: Example of choosing which part of an application the support screen should display, in this case, the conning application

Figure 47 demonstrates a screen layout that incorporates the final design. In this layout, support screens are utilized to display specific sections or components of the navigation system, allowing users to focus on relevant information for their tasks. On the other hand, main screens are dedicated to displaying main applications such as Conning, ECDIS, Radar, and potentially other relevant applications.

Additionally, integration footer screens are incorporated into the layout, which can be touch screens providing convenient access for users to control the screen layout and manage alerts.

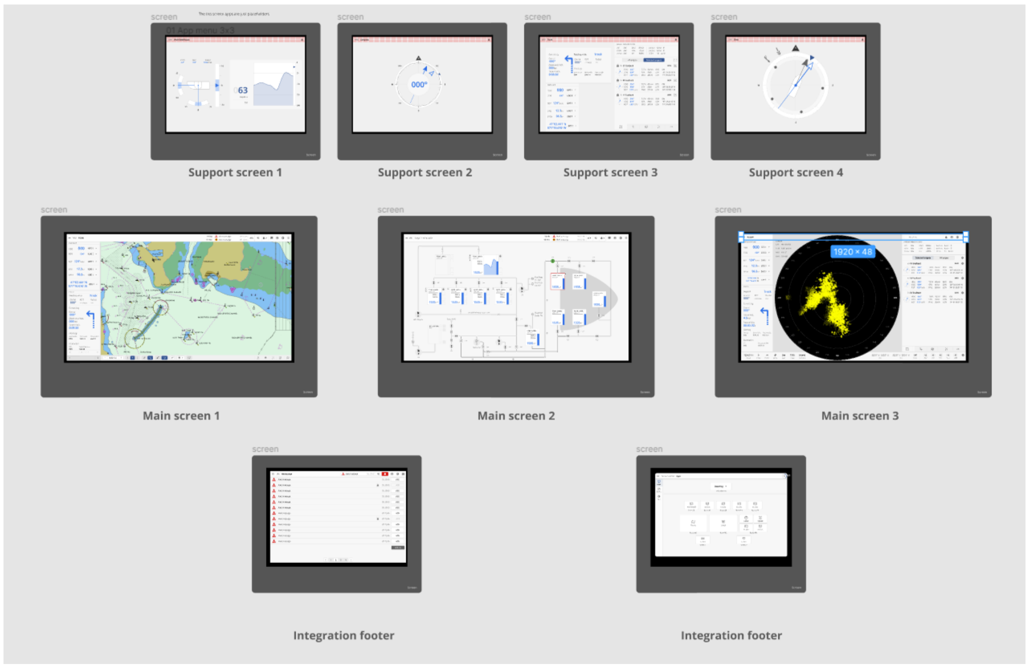


Figure 47: Example of how the screen layout could look like using the Final UI design

5.2 Electron application

In this section, the outcomes of the Electron app implementation are presented, demonstrating the functionality and performance of the application.

5.2.1 SVG layout implementation

The group made two separate SVG systems using the provided examples in section 4.3.1. The first is an SVG design file of the whole layout exported page, containing all the elements inside itself and only manipulates its values using local functions. It is used to illustrate the movement of components showing their design and readability with less focus on the code structure. To view the SVG it's necessary to change line 5 on the main.js file in the earlier example 4 to re-direct into the thrusters.html. Figure 48 illustrates what the page will look like if done correctly.

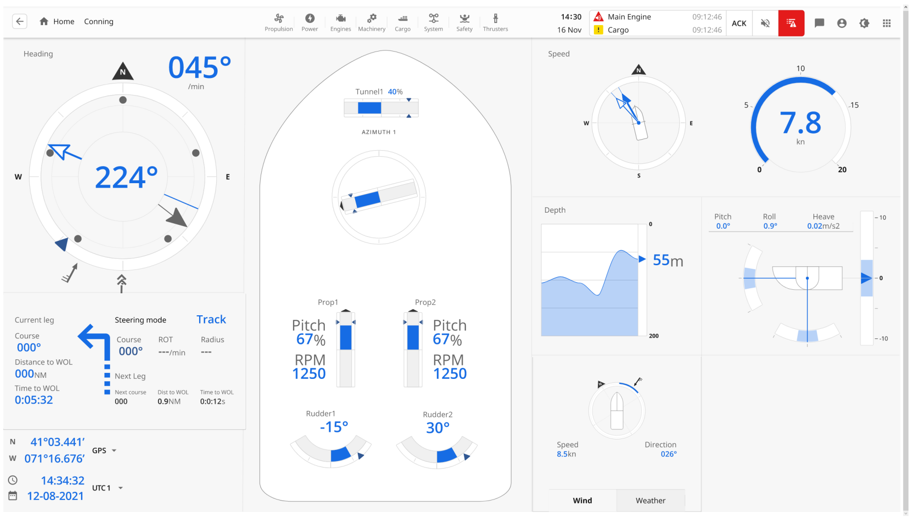


Figure 48: Conning layout SVG file example

The page consists of a rotating compass, a rotating speed component, an azimuth thruster, and a bow thruster. To make the rotational effects, it utilizes a simple sine function shown in snippet 22.

Listing 22: Code for the movement of components inside the SVG, for full code see thrusters.html in appendix I

```
let i = 0;
let j = 0;
let k = 40;
let l = 0;
let m = 0;

function Sinewave() {
    const t = new Date().getTime() / 1000;
    value = Math.sin(t * (2 * Math.PI) * 0.1) * 1;
    angle = (t * 10) % 360;
    i++;
    j--;
    k += 2;
    l -= 2;
    m += 3;
}

setInterval(() => {
            Sinewave();
            attributeValues(
             setPoint,
             "transform",
             "rotate(" + k + ", 256, 364)"
            );
            attributeValues(
             Tunnel1Bar,
             "transform",
             "matrix(" + value + " 0 0 1 809 205)"
            );
           }, 50);
```

The first method doesn't allow any re-usability or scaling as its' functions are local only to the specific SVG implemented inside. This method should never be used as it is never reusable or scalable.

5.2.2 SVG component implementation

The second method for building the designed page was to create each component as its own HTML file. Then be imported and implemented on multiple locations using the components.json file. The JSON file implements each element that is corresponding with the switch case shown in example 14. Utilizing the earlier way of coding in section 4.3.1, the group was able to construct four components onto a page with each of their functions to call on in the main HTML file for their layout. The group did not see the need to fully implement the compass as it had the same function and ways of handling the code as the other components. The functions would set the value for each component depending on the parameter assigned. The components are illustrated in Figure 49. To try the component system, please change line 5 in example 4 to navigate into"indexNew.html".

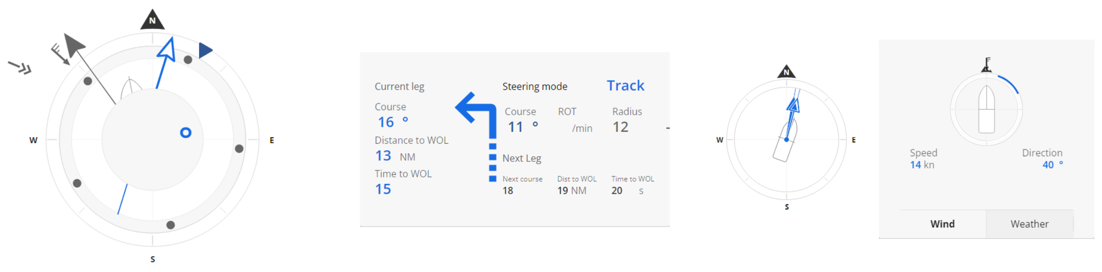


Figure 49: The components made for implementation into bigger systems

The second method allows components to be called on multiple locations rearranging their order and scale to fit a custom layout in future projects. It utilizes a grid to move component positions assigned from 1 through 8 inside its CSS elements. In snippet 23 the grid layout construction is shown[16].

Listing 23: CSS segment used to re-arrange positions, for full code see styles.css in appendix I

```
.container {
  width: 100%;
  display: grid;
    grid-template-columns: repeat(8, 1fr);
    grid-template-rows: repeat(3, 1fr);
    grid-column-gap: 5px;
    grid-row-gap: 5px;
}
```

[16]automatic generated CSS grid layout retrieved from: https://cssgrid-generator.netlify.app

To add more components to a page it's needed to navigate into the ”includer.json” file and add a new line with the name of the desired component from the switch case in mentioned example 14. In snippet 24 the components.json file is shown.

Listing 24: components.json construction example, for full code see components.json in appendix I

```json
{
    "components": [
        { "name": "windAndWeather" },
    ]
}
```

Using the second method of implementation with a more modular approach for applying code to retrieve element IDs inside SVGs and applying it to singular components, allows the code to be reused across new OpenBridge releases. Allowing it to be automated and avoiding the need to implement every function onto every OpenBridge design release manually.

5.2.3 Outdated method for app design

This section presents the outcomes of employing the method described in Section 4.3.2 for implementing elements and the top bar. The following results demonstrate the application of this approach.

Figure 50 showcases an Electron app developed in accordance with the first draft 5.1.2. It represents the initial outcome of the Electron app and illustrates the main page featuring navigation buttons. The top bar and middle buttons enable navigation between windows.

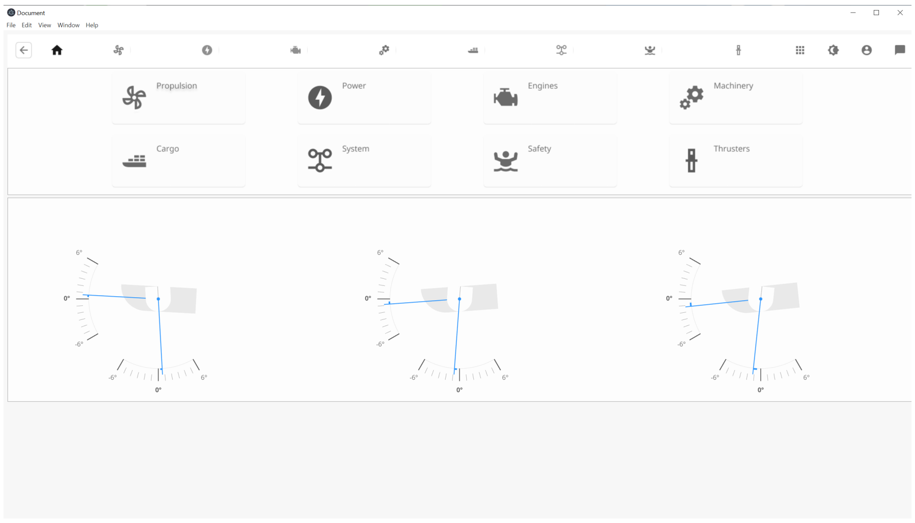


Figure 50: First Electron design in Electron

Figure 51 presents a window from the preliminary version of the Electron app, specifically the thruster window. It encompasses three distinct sizes of OpenBridge thruster elements: large, medium, and small. The app's layout is adjusted to a wide horizontal window to observe the responsiveness of the elements as the window size diminishes. Notably, the thruster elements in the top bar appear in a distinct color compared to the rest. This color differentiation indicates that the thruster window is active.

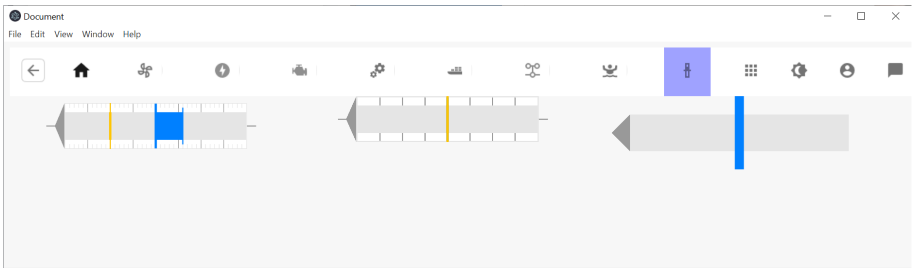


Figure 51: Scaled Thruster window

5.2.4 Responsive design

Responsive design offers a valuable means of distributing data and information based on the size of the bridge, the number of available screens, and the size of the operator's or machinist's workspace. By adapting to different contexts, responsive design ensures that the most crucial information is presented prominently to the relevant operator or machinist at any given time and location.

Figures 52, 53, and 54 exemplify a responsive design test window. This window was created to examine the visual appearance and behavior of various elements when the window size is modified.

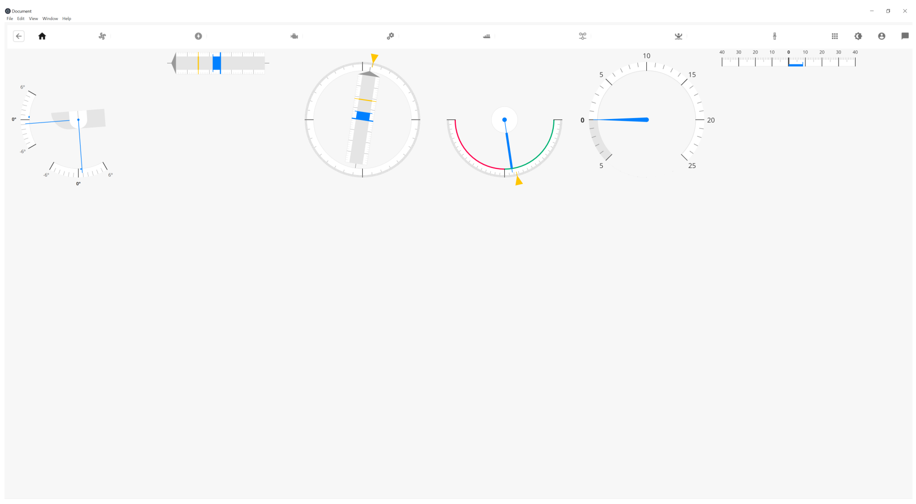


Figure 52: Full screen test window

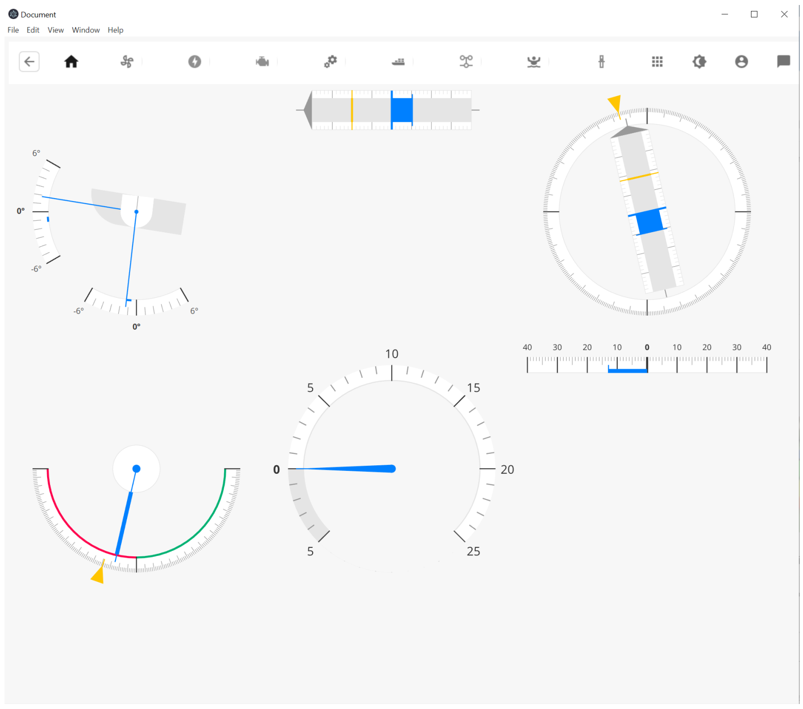


Figure 53: Smaller screen test window

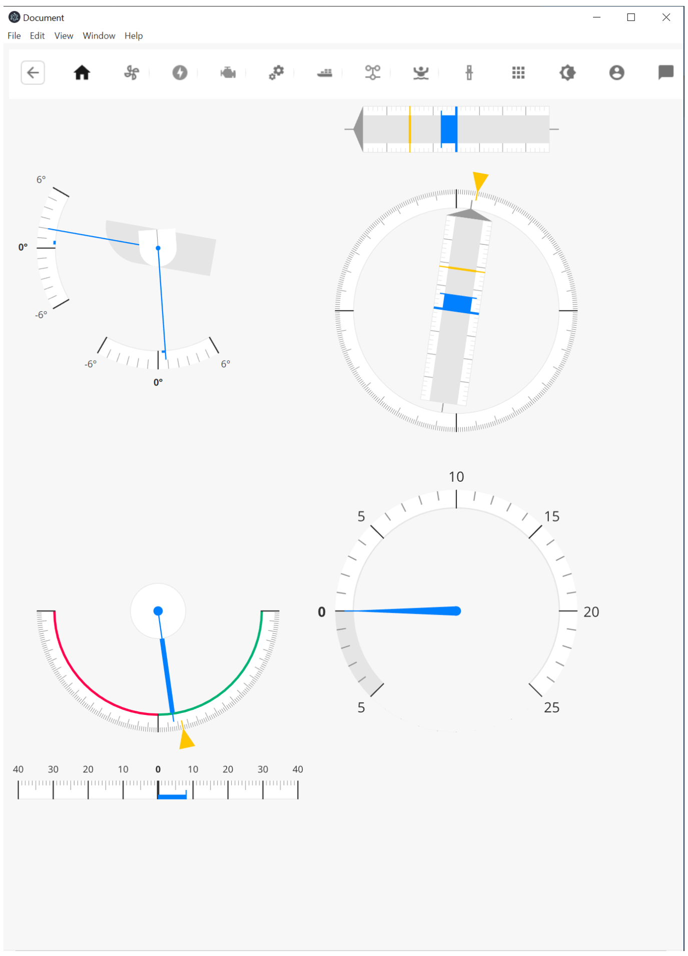


Figure 54: Long screen test window

An ideal responsive design solution involves the elements dynamically adjusting their size and layout based on the window dimensions. For instance, an element may transition from "ob-thruster-large" to "ob-thruster-small" as the window size decreases while maintaining the overall page layout.

6 Discussion

This chapter focuses on the discussion and evaluation of the results obtained in the project and is structured as followed:

- The scalability of what the group has accomplished and how it could have been done differently to accomplish a higher-scale program.
- The challenges the elements in the project have presented and their drawbacks.
- Other developers that have started using OpenBridge in their design.
- The usability of OpenBridge and its guideline.
- The groups' collaborative process during the project and the knowledge it rewarded.

6.1 Scalability

In this subsection, we discuss the potential scalability of Figma, Electron, and OpenBridge in the context of larger systems and future enhancements.

6.1.1 Figma

Utilizing Figma discovered that the platform offers extensive collaborative capabilities, enabling designers to collaborate seamlessly at any scale. With the ability for each designer to provide comments and specific feedback, Figma facilitates a comprehensive sharing of concepts. This assists in continuous monitoring of the design process, allowing team members to track progress and maintain a cohesive environment where everyone remains updated. Consequently, this eliminates the issue of multiple operators unintentionally duplicating tasks as a result of miscommunication.

As the size of Figma projects expands, the performance becomes increasingly demanding and sluggish. This is primarily due to the program's reliance on SVG files from the OpenBridge library, which demands substantial computational power. To prevent overwhelming the device on which it runs, the projects are structured with multiple pages for each design. By dividing the designs into separate pages, the group was able to optimize the prototype function, ensuring that testing the prototype designs didn't consume excessive time for each iteration.

6.1.2 Electron

The group's achievement in Electron is the development of an application that simulates the behaviour of OpenBridge elements on a smaller scale by importing values into the SVG format. The purpose of the app is to provide a practical understanding of how the elements operate, including the interaction with popup windows and navigation through top bars. There is potential to expand the app by incorporating the final design from Figma, integrating all its pages, and enhancing interactivity throughout the application.

6.1.3 Scaling the code

The group acknowledges the execution of the project's code would have been more efficient if we had possessed a deeper knowledge of JavaScript and Electron prior to the project. To compensate for this knowledge gap, the group dedicated significant effort to researching Javascript and Electron before determining our approach. This research process consumed approximately four weeks of valuable time. In hindsight, the group recognizes that incorporating React, TypeScript, or Vue alongside Electron for the layout and designs could have yielded better results. TypeScript, in particular, would have facilitated a more robust module system, enhanced classes, and interfaces, and provided a comprehensive type system for gradual typing [8]. The group believes the most beneficial method, would be a combination of the different methods for integrating as reviewed in section 4.3.1 and section 4.3.2.

React or Vue could have provided a more efficient solution for implementing a user interface with dynamic datasets over time, such as a graph system in the system UI [22]. The group opted to use Electron as it offered easier learning and debugging compared to React. The decision was influenced by the potential for future development, where React and Vue could be integrated to import and display information within the Electron framework. This approach allowed the group to prioritize initial implementation and later explore more advanced possibilities if time permitted.

Evaluating the scalability of OpenBridge as a system, it becomes evident that developing a comprehensive UI builder system could yield significant long-term benefits. Such a system would eliminate the need for hard-coding when implementing new updates or components, providing a more flexible and adaptable solution.

In an ideal scenario, a generating function could be created to extract information, including IPs, IDs, Registers, and other relevant details, from an Excel information file and export it into a CSV file. This information could then be assigned as a list of variables, allowing for easy access and manipulation.

To facilitate the user interface design process, a drag-and-drop GUI interface could be implemented, enabling users to select components from a library and place them on the screen. A grid system could be used to align and lock the components in place. Parameters of the components could be adjusted through a parameter interface, including the ability to assign variables from a drop-down menu.

A drag-and-drop feature similar to the one accomplished by Peter James on Stack Overflow[17] could serve as a reference for implementing this functionality. The capabilities of CDP Studio[18] could be explored, as they have achieved similar advancements in this field.

By developing a fully-fledged UI builder system with these features, OpenBridge could greatly enhance its scalability and usability, allowing for easier and more efficient development and customization of user interfaces over time.

6.1.4 OpenBridge

OpenBridge is a collaborative project that has benefited from the research contributions of multiple academic partners. It is important to note that OpenBridge is still a work in progress, continuously evolving and improving. Over 2000 companies and partners are actively participating in the OpenBridge guideline [28]. This strong support and participation indicate a promising future for the project, as more organizations recognize OpenBridge as a solution for enhancing UX in maritime bridge solutions.

OpenBridge 5.0 is in development, introducing new features and elements aimed at improving safety and UX. The group is confident that future versions, such as OpenBridge 6.0 and beyond, will continue to contribute to the safety and advancement of the maritime interface industry.

[17]Possible solution for drag-and-drop inside a JS environment: https://stackoverflow.com/questions/73251435/drag-and-drop-cells-on-css-grid-only-works-properly-when-moving-a-cell-to-the-ri

[18]CDP Studio company site offers more information here: https://cdpstudio.com

The collaborative nature of OpenBridge, combined with its expanding user base and commitment to ongoing development, suggests that it will continue to play a crucial role. Promoting a safer working environment and driving innovation in the maritime sector.

6.2 Challenges

This subsection discusses challenges within the different results the group has found working with this project.

6.2.1 Figma

Working with Figma introduced the group to the challenge of implementing the designed layouts into interactive code. While Figma offers significant improvements for designing interfaces, there was no straightforward method available during the project to convert the designs into functional code. The Figma community is actively exploring this challenge and working towards creating plugins that would simplify the conversion process. Despite the absence of a current solution, the group believes that given Figma's status as a leading design tool, it is likely that a plugin or a solution will be developed in the near future to address this issue.

6.2.2 Electron

The use of Electron for creating server-client communication software presented its own set of challenges for the group. Without further extensions of TypeScript and React or Vue applications, the operation of Electron seemed limited. Particularly, when multiple elements required local properties on the same page, the program became more complex to manage.

To address the needs of a larger system onboard a vessel, a more extensive hierarchy of classes would be necessary to ensure the construction of the software remains readable and understandable. By organizing the code into a well-defined class structure, the group would be able to maintain clarity and manage the complexities of a bigger system more effectively. This approach would contribute to the overall maintainability and scalability of the software solution.

6.2.3 OpenBridge

The group holds a positive perception of OpenBridge and recognizes its positive impact on the market. There are challenges within OpenBridge that have been discussed; One such challenge is the absence of an up-to-date online library for web components. This lack of a comprehensive library made the programming aspect of the project more complex during the implementation of elements.

The group is aware of the intricacies of maritime vessels, noting that each ship has its unique construction that may not match others. This individuality poses a challenge as not all equipment found on vessels would be available in the OpenBridge library, requiring designers to create them themselves. While OpenBridge provides rules and equipment for styles, typography, and colour coding, there is a void in terms of UI elements for certain types of equipment.

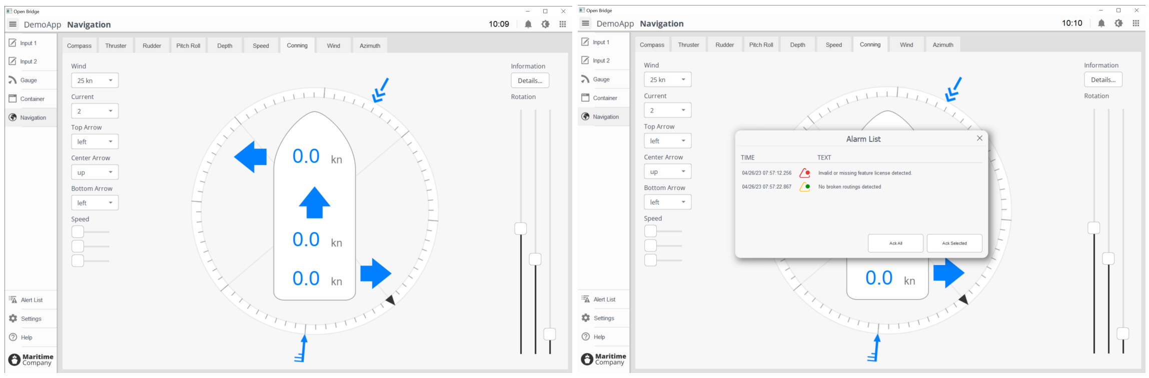
Addressing these challenges by expanding the online library for web components and enhancing the coverage of equipment within OpenBridge would contribute to a more comprehensive and user-friendly experience for designers working with maritime vessels.

6.3 Existing programs

The following section, discuss existing programs that already implemented OpenBridge, in designs and hardware systems.

6.3.1 CDP Studios

Figure 55 illustrates a design template in CDP Studio that showcases OpenBridge elements. This visual representation serves as a suggestion for the appearance of a user interface utilizing OpenBridge elements. It is important to note that this design is for illustrative purposes only and does not possess any functional capabilities. Nonetheless, it offers an example of how a user interface can be visually presented with the integration of OpenBridge elements.



TIME	TEXT
04/26/23 07:57:12.256	Invalid or missing feature license detected.
04/26/23 07:57:22.867	No broken routings detected

(a) CDP UI

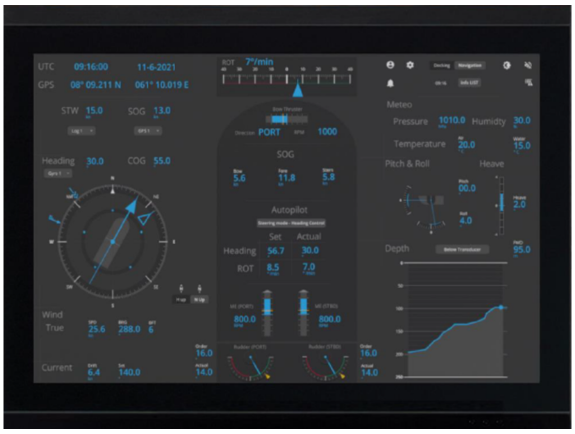
(b) CDP UI Alarm window

Figure 55: CDP Studios UI

The group made an attempt to recreate the conning layout presented in section 5.1.2 using CDP Studio. They found the program to be overwhelming and challenging to work with. Given the time constraints and limited resources available, the decision was made to abandon the project in CDP Studio, as it was not deemed a focal point that would provide substantial value to the overall goals of the project.

6.3.2 Alphatron Marine

Figure 56 displays a design created in AlphaMINDS utilizing OpenBridge. The design showcases two distinct layouts for different operations: one for conning and another for docking.


(a) AlphaMINDS Conning [5]

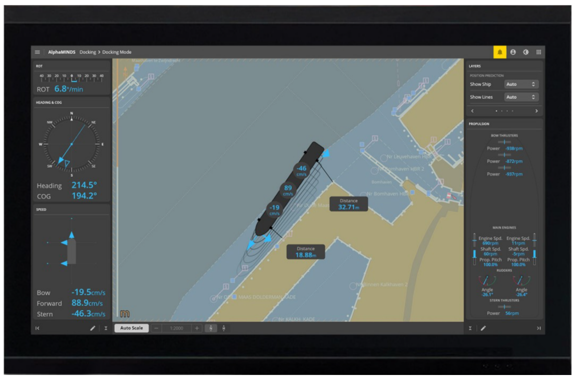
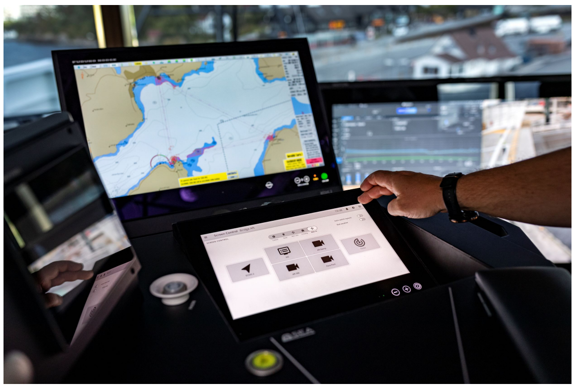

(b) AlphaMINDS Docking [6]

Figure 56: AlphaMINDS UI

6.3.3 SEAM AS

Figure 57 presents real-life images of a bridge where OpenBridge has been implemented.


(a) e-SEA Bridge with OpenBridge implemented [12]

(b) e-SEA Bridge [12]

Figure 57: e-SEA Bridge

6.4 Usability

In this subsection, we will discuss the usability and ease of use of OpenBridge, as well as the reasons behind its open-source nature.

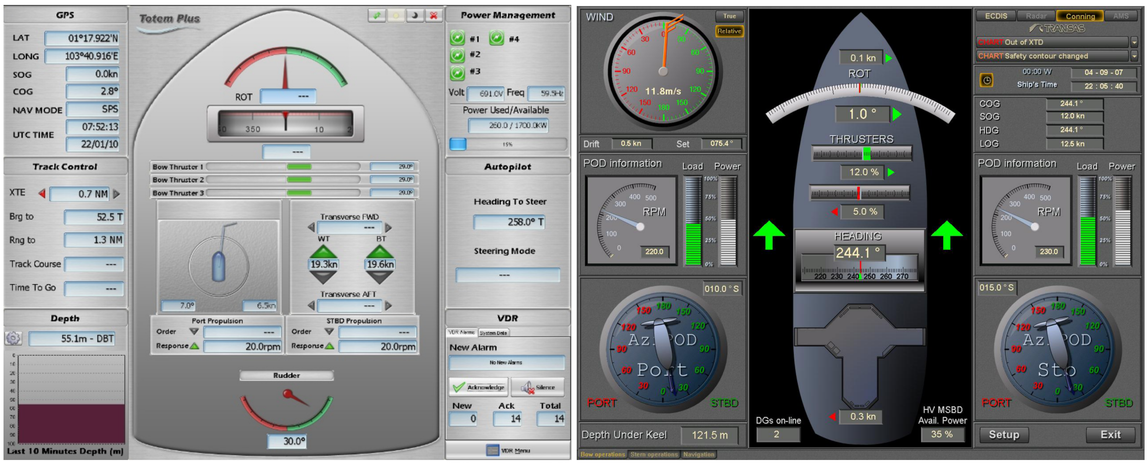
6.4.1 Ease of use

UI designs are carefully crafted to be intuitive and user-friendly, providing a seamless experience for users. The goal is to present information in a subtle and effortless manner, without overwhelming users with complex requirements or the need for extensive prior knowledge. Services such as sending messages, online shopping, or streaming music are expected to be efficiently performed with minimal training or user effort.

Over the past two decades, Apple and Google have established a standard in UI design, offering design guidelines that ensure users can easily comprehend and navigate systems. Adhering to these guidelines has become a reliable approach to guaranteeing user understanding and familiarity.

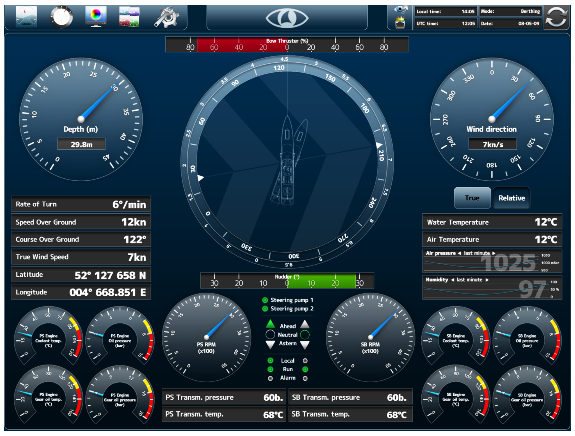
OpenBridge prioritizes self-explanatory designs that facilitate user interaction. In comparing the maritime industry to social media platforms like Twitter and Facebook, similar principles can be observed. Ocean Industries Concept Lab's studies [40] have indicated that maritime user interfaces intentionally differentiate themselves from other applications for marketing purposes and individual brand recognition.

Figure 58 and Figure 59 exemplify different conning pages in maritime systems, showcasing how designs within this industry can vary.



(a) Totem Plus Conning [11] (b) Navi-Conning 5000 [25]

Figure 58: Totem Plus- & Navi Conning


(a) Free Technincs Conning [10]

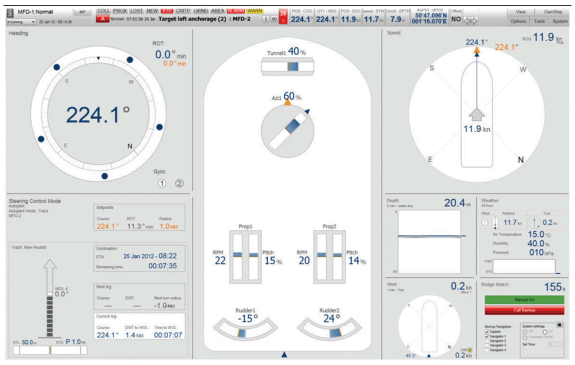

(b) K-Bridge Conning [23]

Figure 59: Free Technics - & K-Bridge Conning

6.4.2 Open source

According to studies [40], maritime systems are intentionally designed to distinguish themselves from other applications, serving marketing and branding purposes. OpenBridge takes a different approach as an open-source library, inviting contributions from anyone to expand its capabilities. This allows for a collaborative effort, where multiple systems and stakeholders work together to find effective solutions for enhancing UX in maritime bridge solutions.

6.4.3 Presets for operation

Presets for different operations play a crucial role in providing a comprehensive overview of the vessel's state during various activities. The value of data differs depending on the specific operation being carried out. For instance, when docking a vessel, certain information becomes more critical compared to when the vessel is engaged in a fishing operation out at sea.

While the group did not create any designs specifically addressing this aspect, Figure 56 showcases a solution developed by Alphatron Marine that addresses the need for distinct operations, namely conning and docking, within their user interface.

6.5 Process

This subsection will reflect and discuss the group's approach, process, and work ethics over the period of the project.

The team embarked on a unique project that diverged from the typical endeavors of electrical engineering students. It proved to be a valuable experience, providing us with valuable insights into design, programming, software, and teamwork, which will greatly benefit our future engineering careers. It deepened our understanding of equipment communication with human operators.

Extensive research was required for the project, as it was essential to familiarize ourselves with the software necessary for its development. The majority of the software employed in the project was unfamiliar to the team prior to undertaking this assignment.

The group maintained a steady work pace throughout the project duration after the parallel subject was completed. The most challenging aspect of the project has been the process of acquainting ourselves with maritime terminology, design procedures, and complex programming codes. These difficulties have proven to be beneficial, as they have expanded our understanding of various engineering production processes beyond automation alone.

7 Conclusion

The primary objective of the project was to implement the OpenBridge design system standard in both existing and prospective maritime applications. This implementation involved integrating the OpenBridge standard within the Acon to K-Chief project, thereby unifying the software and design principles of a tangible hardware system. Additionally, the project aimed to assess OpenBridge's suitability as a standard UI and HMI in terms of usability, safety, scalability, and security. It sought to explore the potential future applications of OpenBridge across various maritime domains.

OpenBridge has demonstrated its efficacy as a user-friendly interface, employing universally recognized elements and symbols. This feature facilitates comprehension for operators and unfamiliar individuals who may encounter the system. The widespread familiarity with OpenBridge contributes to enhanced safety within the maritime industry.

The design of the system was accomplished using Figma, allowing for a comprehensive evaluation of the system's flow and overall aesthetic before implementing the design into an Electron app. Figma provides an intuitive and accessible drag-and-drop methodology for design purposes. The successful conversion of the design into an Electron app, which integrates software and hardware, was exemplified by the group's demonstration of connecting a PLC to the Electron demo.

OpenBridge is a design guideline that has attracted the participation of numerous companies aiming to foster a safer maritime work environment. Global investors and academic researchers continue to refine this guideline, which is reinforced by its open-source library. Enabling users to freely provide feedback directly to the designers.

Throughout the project, the group acquired useful experience in design, programming, the maritime industry, and the intricacies of collaborative work involving multiple stakeholders.

7.1 Further work

In this section, we evaluate the tasks needed for future iterations and improvements, together with possible ways to expand our results and evaluations given in section 6.

- Expand the design in Figma and enhance its overall feel and flow, attention should be given to refining visual aesthetics, optimizing UX, and ensuring smooth navigation. This can be achieved through a thoughtful selection of color schemes, typography, and intuitive UI elements while maintaining consistency across different screens and interactions. Iterative design processes, user feedback, and usability testing can greatly contribute to improving the overall feel and flow of the application's design.
- Enhance the results. The application development process can be refined by incorporating TypeScript for robust class definitions, integrating Electron to enable GUI-driven functionality, and leveraging React to improve layout design and streamline application animations. This approach promotes efficient development practices and facilitates a seamless user experience.
- Establish a user database with administration rights for interface stations on a vessel. Define user roles and access privileges, design a structured database, implement secure user authentication mechanisms, develop user management functionality, and secure the UI with a custom layout connected to the user's profile.
- Ensure future-proofing of OpenBridge updates. An effective solution involves adopting a modular code structure that facilitates easy extraction of IDs, input registers, and IPs required for component implementation. This approach enables efficient integration of new features and updates, allowing for agile development and minimizing the impact on the existing codebase.

Bibliography

[1] *10 reasons why you should use NodeJs.* 2023. URL: https://www.projectpro.io/article/10-reasons-why-you-should-use-nodejs/129 (**urlseen** 15/05/2023).

[2] *About.* NaN. URL: https://www.openbridge.no/home/about (**urlseen** 28/04/2023).

[3] *About.* NaN. URL: https://www.oicl.no/home/about (**urlseen** 15/05/2023).

[4] *About — Node.js.* NaN. URL: https://nodejs.org/en/about (**urlseen** 28/04/2023).

[5] *AlphaMINDS Conning.* NaN. URL: https://www.alphatronmarine.com/en/product/alphaminds-conning-548/ (**urlseen** 15/05/2023).

[6] *AlphaMINDS Docking.* NaN. URL: https://www.alphatronmarine.com/en/product/alphaminds-docking-562/ (**urlseen** 15/05/2023).

[7] *Automation system, K-Chief - Kongsberg Maritime.* NaN. URL: https://www.kongsberg.com/maritime/products/engines-engine-room-and-automation-systems/automation-safety-and-control/vessel-automation-k-chief/ (**urlseen** 15/05/2023).

[8] Gavin Bierman, Martın Abadi **and** Mads Torgersen. ?Understanding TypeScript? **in***ECOOP 2014 – Object-Oriented Programming*: Springer Berlin Heidelberg, 2014, **pages** 257–281. DOI: 10.1007/978-3-662-44202-9_11. URL: https://doi.org/10.1007%2F978-3-662-44202-9_11.

[9] *CDP Studios - Who we are.* NaN. URL: https://cdpstudio.com/aboutus/ (**urlseen** 15/05/2023).

[10] *Conning.* NaN. URL: https://www.freetechnics.eu/products/conning (**urlseen** 15/05/2023).

[11] *Conning System - For any type of vessel.* NaN. URL: https://www.totem-plus.com/conning (**urlseen** 15/05/2023).

[12] *e-SEA® Bridge - Developing a bridge solution for the future.* NaN. URL: https://www.seam.no/insights/developing-the-bridge-solution-of-the-future-with-openbridge (**urlseen** 15/05/2023).

[13] *Electron Development: The Complete Guide to Getting Started.* 2020. URL: https://www.trio.dev/blog/electron-development-guide (**urlseen** 15/05/2023).

[14] Nærings- og fiskeridepartementet. *Maritim næring - regjeringen.no.* NaN. URL: https://www.regjeringen.no/no/tema/naringsliv/maritim-naring/ny-temaside/forste-kolonne/maritime-naringer/id2589227/ (**urlseen** 04/05/2023).

[15] *Human Machine Interface (HMI) vs Graphical User Interface (GUI).* NaN. URL: https://nelson-miller.com/human-machine-interface-hmi-vs-graphical-user-interface-gui/ (**urlseen** 15/05/2023).

[16] *Human Machine Interface (HMI) vs Human-Computer Interface (HCI).* 2020. URL: https://nelson-miller.com/human-machine-interface-hmi-vs-human-computer-interface-hci/ (**urlseen** 15/05/2023).

[17] *IAS - Norwegian Electric Systems.* NaN. URL: https://www.norwegianelectric.com/products/ias/ (**urlseen** 20/04/2023).

[18] *IEC 62288:2021.* 2021. URL: https://webstore.iec.ch/publication/64659 (**urlseen** 15/05/2023).

[19] *Integrated Automation System - Ulstein.* NaN. URL: https://ulstein.com/marine-automation/ulstein-ias (**urlseen** 15/05/2023).

[20] *International Convention for the Safety of Life at Sea (SOLAS Convention).* 2016. URL: https://www.jus.uio.no/english/services/library/treaties/08/8-03/safety-life.html (**urlseen** 15/05/2023).

[21] *Introducing JSON.* NaN. URL: https://www.json.org/json-en.html (**urlseen** 16/05/2023).

[22] *Introduction to React - Cory Gackenheimer - Google Bøker.* NaN. URL: https://books.google.no/books?hl=no&lr=&id=NZCKCgAAQBAJ&oi=fnd&pg=PR6&dq=React&ots=KBztRlCy8e&sig=odhKnOtqOGjUmNhGlkEF9iVh0dk&redir_esc=y#v=onepage&q=React&f=false (**urlseen** 15/05/2023).

[23] *K-Bridge Conning Display.* NaN. URL: https://www.kongsberg.com/maritime/products/bridge-systems-and-control-centres/navigation-systems/conning-display/ (**urlseen** 15/05/2023).

[24] *Kongsberg Maritime fast facts - Kongsberg Maritime*. NaN. URL: https://www.kongsberg.com/maritime/about-us/kongsberg-maritime-fast-facts/ (**urlseen** 04/05/2023).

[25] *NAVI-CONNING 5000*. NaN. URL: https://www.maritech-adriatic.com/en/Navi-Conning/ (**urlseen** 15/05/2023).

[26] *New OpenBridge conning library in Lynx by JRC/Alphatron Marine*. NaN. URL: https://www.alphatronmarine.com/en/article/new-openbridge-conning-library-in-lynx-by-jrcalphatron-marine/ (**urlseen** 15/05/2023).

[27] *OpenBridge Design System - ODES - Prosjektbanken*. NaN. URL: https://prosjektbanken.forskningsradet.no/project/FORISS/296151?Kilde=FORISS&distribution=Ar&chart=bar&calcType=funding&Sprak=no&sortBy=score&sortOrder=desc&resultCount=30&offset=0&Fritekst=296151 (**urlseen** 19/05/2023).

[28] *OpenBridge Design System - ODES - Prosjektbanken*. NaN. URL: https://prosjektbanken.forskningsradet.no/project/FORISS/296151?Kilde=FORISS&distribution=Ar&chart=bar&calcType=funding&Sprak=no&sortBy=score&sortOrder=desc&resultCount=30&offset=0&Fritekst=296151 (**urlseen** 19/05/2023).

[29] *OpenBridge library link to Figma*. NaN. URL: https://www.openbridge.no/figma/current-release (**urlseen** 01/05/2023).

[30] *OpenBridge maritime UI's*. NaN. URL: https://cdpstudio.com/blog/openbridge-maritime-uis/ (**urlseen** 15/05/2023).

[31] *Presenting OpenBridge Design System*. NaN. URL: https://medium.com/ocean-industries-concept-lab/presenting-openbridge-design-system-3aac447a0a02 (**urlseen** 19/05/2023).

[32] *SOLAS*. NaN. URL: https://www.imo.org/en/KnowledgeCentre/ConferencesMeetings/Pages/SOLAS.aspx (**urlseen** 15/05/2023).

[33] *SVG files*. NaN. URL: https://www.adobe.com/creativecloud/file-types/image/vector/svg-file.html (**urlseen** 15/05/2023).

[34] *SVG Standarer*. NaN. URL: https://www.digdir.no/standarder/svg-scalable-vector-graphics/1731 (**urlseen** 15/05/2023).

[35] *User Experience (UX) Design*. NaN. URL: https://www.interaction-design.org/literature/topics/ux-design (**urlseen** 15/05/2023).

[36] *Web Content Accessibility Guidelines (WCAG) 2.0*. 2008. URL: https://www.w3.org/TR/WCAG20/ (**urlseen** 15/05/2023).

[37] *What is Electron?* NaN. URL: https://www.electronjs.org/docs/latest (**urlseen** 15/05/2023).

[38] *What Is Figma and What Is It Used For?* NaN. URL: https://www.makeuseof.com/what-is-figma-used-for/ (**urlseen** 20/05/2023).

[39] *What is User Experience? — Definition and Overview*. NaN. URL: https://www.productplan.com/glossary/user-experience/ (**urlseen** 20/05/2023).

[40] *Why digital user interfaces matter for ship owners*. NaN. URL: https://www.oicl.no/content/why-digital-user-interfaces-matter-for-ship-owners (**urlseen** 15/05/2023).

Appendices

A Preliminary project report

See attached Zip/Preproject

B Gantt diagram

Gantt diagram of the project

						Milestone 1			Milestone 2				Milestone 3			Milestone 4		Milestone 5
	Jan 16 - 22	Jan 23 - 29	Jan 30 - Feb 5	Feb 6 - 12	Feb 13 - 19	Feb 20 - 26	Feb 27 - Mar 5	Mar 6 - 12	Mar 13 - 19	Mar 20 - 26	Mar 27 - Apr 2	Apr 3 - 9	Apr 10 - 16	Apr 17 - 23	Apr 24 - 30	May 1 - 7	May 8 - 14	May 15 - 22
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16	Week 17	Week 18
Documentation												Easter						
Preliminary project report	🟩	🟩										Easter						
Progress reports / Meetings		🟩		🟩		🟩		🟩		🟩		Easter		🟩		🟩		
Report												Easter			🟩	🟩	🟩	🟩
Research												Easter						
OpenBridge	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	Easter	🟦	🟦	🟦	🟦	🟦	🟦
Electron					🟦	🟦						Easter						
Figma			🟦	🟦	🟦							Easter						
JavaScript/HTML			🟦	🟦	🟦							Easter						
Maritime industry				🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	Easter	🟦	🟦				
Design & Programming												Easter						
Figma					🟧	🟧	🟧	🟧	🟧	🟧	🟧	Easter	🟧	🟧				
JavaScript/HTML						🟧	🟧	🟧	🟧	🟧	🟧	Easter	🟧	🟧	🟧			
Electron							🟧	🟧	🟧	🟧	🟧	Easter	🟧	🟧	🟧	🟧		
Communication												Easter						
Modbus										🟨	🟨	Easter	🟨					

C Hour list

See attached Zip/HourList

D Biweekly reports

See attached Zip/Biweekly

E Meeting reports

See attached Zip/Meetingreports

F Project pitch

See attached Zip/projectpitch

G Poster

Project poster

NTNU

Institutt for IKT og realfag

KONGSBERG

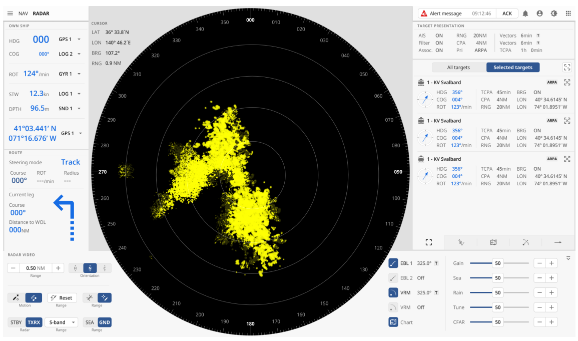
OpenBridge Design System

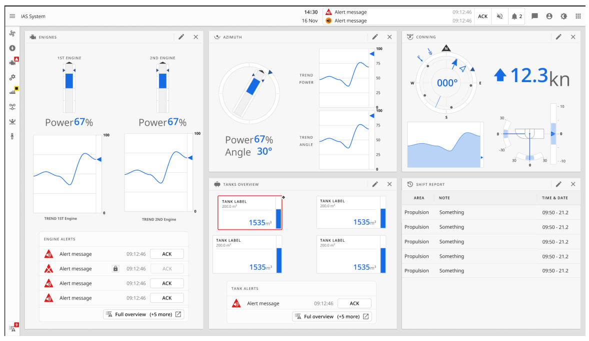
Implementation in existing, and future maritime applications

Introduction:

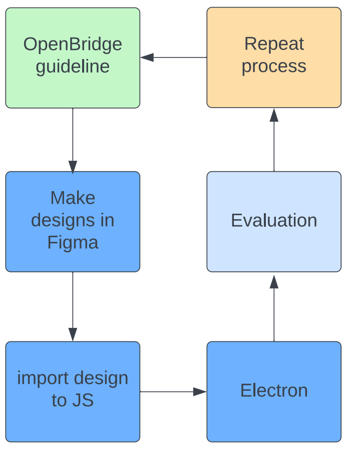
In the maritime industry, the bridge on boats and other maritime vessels contains a lot of systems with different interfaces and functions. As a result, an evaluation of the design system OpenBridge was done to explore its possibilities of tying hardware and software into one, together with its overall user experience.

Results:





Method:



Summary:

The goals of this project was to use OpenBridge guidelines to design a prototype system in Figma, implementing this to an Electron app, and lastly evaluating OpenBridge as a whole.

OpenBridge Figma ELECTRON JavaScript JS

Karl Johan Alvestad | Helene L. Rasmussen | Lasse Raaum | Henrik Rian

H Figma project

Link to Figma project containing the designs

https://www.figma.com/file/rTJOBciqGP3zqvJWIjenuC/OpenBridge-Bachelor-Kongsberg?type=design&node-id=39194%3A361519&t=D31E54EKyZyQFjhC-1

I Electron Code using SVG files

Code developed for Electron design

check GitHub or zip folder

https://github.com/kongapls/OpenBridge-Bachelor/releases/tag/Release

J Electron Code using online library

See attached Zip/Alternativeapp

K e!COCKPIT

The e!cockpit program used for the PLC configuration

can also be viewed from Zip/e!COCKPIT

https://github.com/kongapls/E-cockpit-Bachelor/releases/tag/Release

L Contact log

Contact log of the informants providing feedback throughout the project

See attached Zip/Contactlog

M Video

The bachelor video presentation

See attached Zip/Video