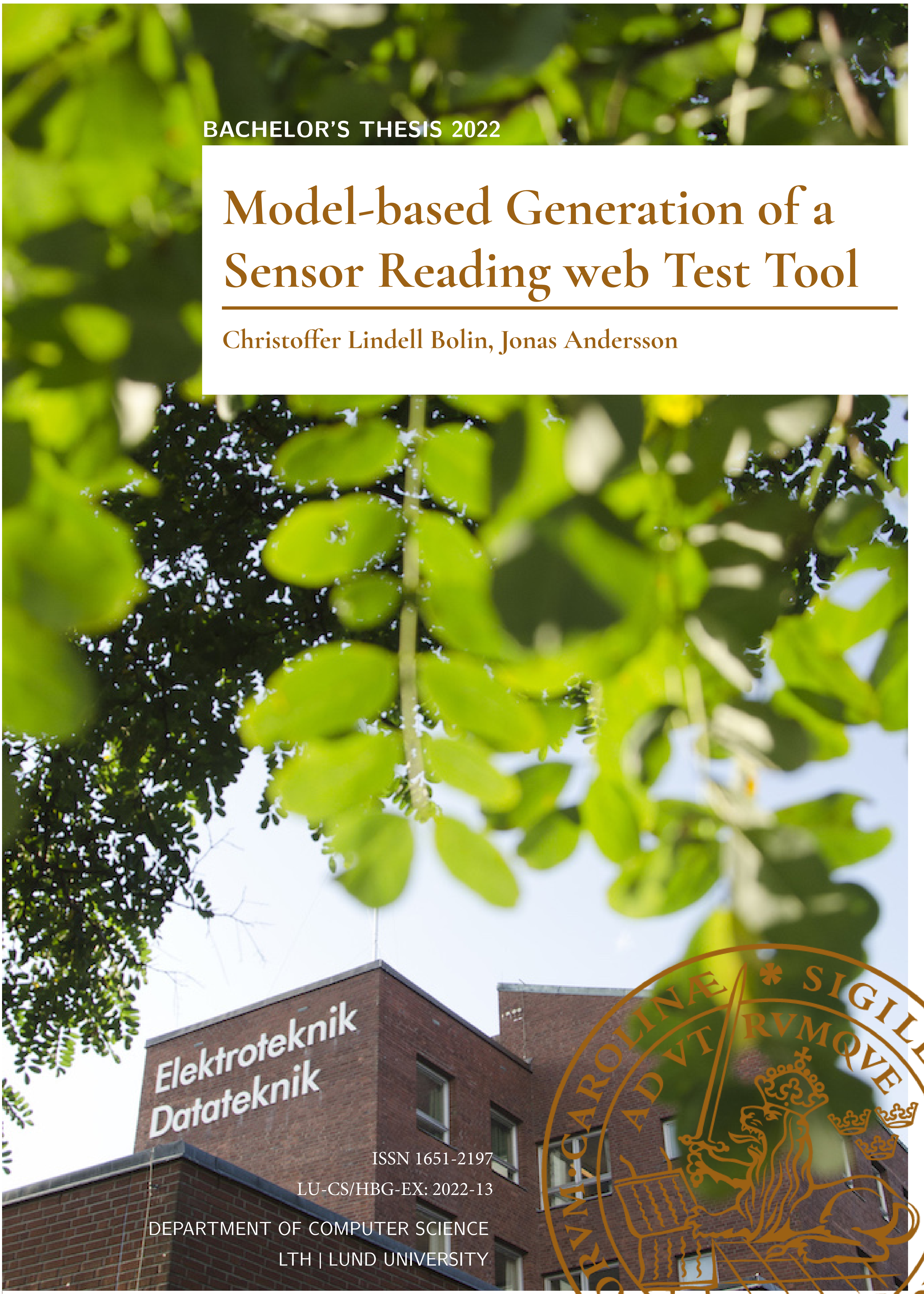
BACHELOR'S THESIS 2022

# Model-based Generation of a Sensor Reading web Test Tool

Christoffer Lindell Bolin, Jonas Andersson



ISSN 1651-2197
LU-CS/HBG-EX: 2022-13

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

KANDIDATARBETE
Datavetenskap

LU-CS/HBG-EX: 2022-13

# Model-based Generation of a Sensor Reading web Test Tool

Modellbaserad generation av sensorläsande webbaserat testverktyg

**Christoffer Lindell Bolin, Jonas Andersson**

# Model-based Generation of a Sensor Reading web Test Tool

Christoffer Lindell Bolin
chrisse.lindell@gmail.com

Jonas Andersson
jonas.bruse@telia.com

September 5, 2022

Bachelor's thesis work carried out at Saab Kockums AB.

Supervisors: Marcus Klang, marcus.klang@cs.lth.se
Daniel Thunberg, daniel.thunberg@combitech.se

Examiner: Jesper Öqvist, jesper.oqvist@cs.lth.se

## Abstract

This Bachelor's thesis is in collaboration with Saab Kockums AB. Saab Kockums develops and constructs marine technology, ships and submarines for civil and military use. In different development stages for systems to be implemented and built on ships, submarines or the like, different components such as sensors or information display systems need to be tested and functionality verified. For the system to work, the interfaces between components need to be able to send consistent and compatible data adhering to a standard. The communication requirements between interfaces are specified in an Interface Requirement Specification (IRS). To effectively test new hardware built by a third party requires a test tool to be built. Today, such a test tool is constructed from scratch for each project, a time-consuming process. A small change to the IRS can also add significant manual synchronization effort across different stakeholders and developers. This thesis investigates how to construct a generic test tool using a minimal machine-readable specification. We built a test tool that introduces dynamic specification of any field-based protocol with the use of an XML specification format. This test tool is designed to send, receive, and validate the user-specified protocols with the help of a React based web user interface and Saab's WISE framework for a modular network architecture. The implemented parts are the CJEX library, a React application, and the WISE components ParseSE, TLDriver and APIDriver, along with an XML standard for the solution. To validate the tool, we used a specification for an Inertial Navigation System (INS) that communicates using the NMEA 0183 protocol, and can send updates with a frequency up to 100 Hz. In addition, we also demonstrate the test tool's flexibility to support multiple protocols by implementing a JSON protocol.

**Keywords**: WISE, GUI, REST, Testing, Integration

# Acknowledgements

We want to thank Andreas Carlsson first and foremost for giving us the opportunity to work with this project. This project has been a great learning experience for both of us and we have made many new contacts within the field which we value highly. Andreas has been a great help throughout by giving accurate and valuable feedback to us.

We want to thank Daniel Thunberg for being our supervisor at Saab. He has had to endure plenty of questions from us. The outcome of this thesis would most likely not have been as favourable without you. Thank you for your patience and competence.

Marcus Klang, our LTH supervisor, is incredibly clear-sighted and experienced. We appreciate all the guidance we have received. He has widened our thought process to include things we probably would not have thought of ourselves.

We would like to acknowledge Michael Johansson for his great humour, support and valuable feedback throughout this thesis.

We are also grateful for Tobias Almén the creator of WISE for taking his time guiding us through WISE, being available for discussion and giving feedback.

We would like to thank Jesper Öqvist for deciding to partake in this thesis as examiner.

Finally we would like express a huge gratitude towards all Saab employees at Saab Kockums Helsingborg, they have all been really friendly, helpful and welcoming.

# Contents

# Foreword

This article is a bachelor's thesis and therefore the knowledge corresponding to a bachelor's degree in Computer Science or equivalent is assumed. As such the motivation to perform this work is partly to work towards a degree. However, in this process, we gain an opportunity to perform meaningful work and to acquire additional knowledge.

# Terminology

Some terms can be repeated during this dissertation, including the following terms.

**COTS** - Commercial off the shelf, public purchase-able product.

**CJEX** - Acronym from "Christoffer Jonas Examensarbete" (bachelor work), Swedish term for .NET class library

**EA** - Enterprise Architect

**ECDIS** - Electronic Chart Display and Information System

**GUI** - Graphical User Interface

**INS** - Inertial Navigation System

**IRS** - Interface Requirement Specification

**JS** - JavaScript

**MUI** - Material UI, a component library for react

**MVP** - Minimum Viable Product

**props** - A often used name for parameters passed into functional components in react development

**SE** - Synthetic Environment, a simulation environment

**QOL** - Quality of Life

**WECDIS** - Warship Electronic Chart Display and Information System

**XSD** - Extensive Markup Language Schema Definition

**XML** - Extensive Markup Language

# Chapter 1 Introduction

This thesis is done in collaboration with the Swedish aerospace and defence company Saab AB, more specifically with the business area called Kockums. Saab Kockums develops, constructs and maintains surface vessels, submarines and marine technology. The different constructed vessels contain a network of systems similar to cars with their CAN bus and different sensors and control systems.

Before a system is assembled for use in marine vessels, each separate system is tested to ensure that all of the individual components work as expected. Then the system is built together and more extensive testing is done to ensure no compatibility issues. Furthermore a reference system is constructed to test for new system modifications and down the line issues that may occur. This avoids having to conduct testing onboard which usually is less than ideal. The vessel may not always be available since it is in active service. And if it can be obtained, the ship will be out of service during the period. A ship out of service results in lost potential which can be expensive. Additionally, testing onboard is cumbersome due to tight spaces, which makes it time consuming.

For the mentioned steps, different components can be built by third-party, which brings the need to know how to set up the communication between components. This is where the Interface Requirement Specification (IRS) is needed, which is modelled as UML using a tool called Enterprise Architect. Interfaces can be defined as the rules for the communication between components such as what operations and types of messages that are exchanged, while protocols define how those messages are formatted and transformed. Each component can implement different interfaces, using different protocols and sentences, and the IRS describes those protocols and sentences to be used along with information about what ports to use and similar information. Additionally, tools to verify the interfaces and simulate communication to control systems, and read information from sensors, need to be implemented with the help of different IRSes, by using the IRS as a source for validation of the test tools specification, and then using the test tool to validate the components interfaces without a need for other parts of the complete system. Any change to an interface needs to be updated in the IRS and communicated between the parts. An example would be if a system is to be built, a certain need for a specific data format to one component exists. The sensor that feeds data to that component need to do so in a consistent manner. When ordering that component, the IRS is consulted to specify what format the communication has to have from the sensor. This is mainly for the one building the component to know what communication needs to be implemented.

## 1.1 The problem

The problem for Saab Kockums is related to the fact that test tools are ordered from third party and implemented from scratch for every IRS. Instead of having one test tool that can adapt to a given IRS, a test tool is ordered from third parties for every new or modification made to an IRS. Ordering a test tool involves sending e-mail, waiting for responses and validating continuously until delivery is completed. Any new change to the IRS need to be communicated, implemented and validated across parts. This process can take several weeks whereas reusable software can be changed in minutes.

Currently, IRSes are human-readable only and test tools are implemented by a developer reading the IRS. This could easily result in requirements being missed or misunderstood. Generating a test tool straight from a machine-readable IRS removes these risks and improves traceability between IRS and test tool.

Some questions asked at the initial phase of the project include:

- How many of all the WISE components can be generated dynamically by code?
- What are the interface requirements to consider when generating components?
- What language or techniques can be used to enter data into the generation?
- To what extent can existing functionality in Enterprise Architect be used?
- How does one design the use of component generation to be as efficient as possible while taking into account the limited time frame for the task?
- How is the balance between dynamic and hard coding made based on the time frame?
- What specification is required to generate a functional and dynamic GUI?

Additional subset of central questions which arose in the process:

- How does one show the user real time high frequency data in a comprehensive manner?
- What other protocols may be relevant to extend for?
- What data is interesting for the user?
- What software parts have what responsibility?
- What are the relevant capabilities of WISE?

## 1.2 Purpose

The goal of this solution is to show viability and concept for an in-house reusable tool for testing and simulating sensors. This tool is to be used to decrease time of integration and testing of communication systems. Any interesting functionality yet to be implemented is also documented in this report to lay the foundation for a better solution than currently exists. A secondary goal is to implement or create possibility to implement with minimal change, a coupling between IRS and test tool to increase traceability.

The initial proposal can be seen in figure 1.1, which was proposed by Saab at early briefings. What the figure shows is that Enterprise Architect generates some input to our solution and the output is a set of components. Those components should then be able to be reused at various moments.

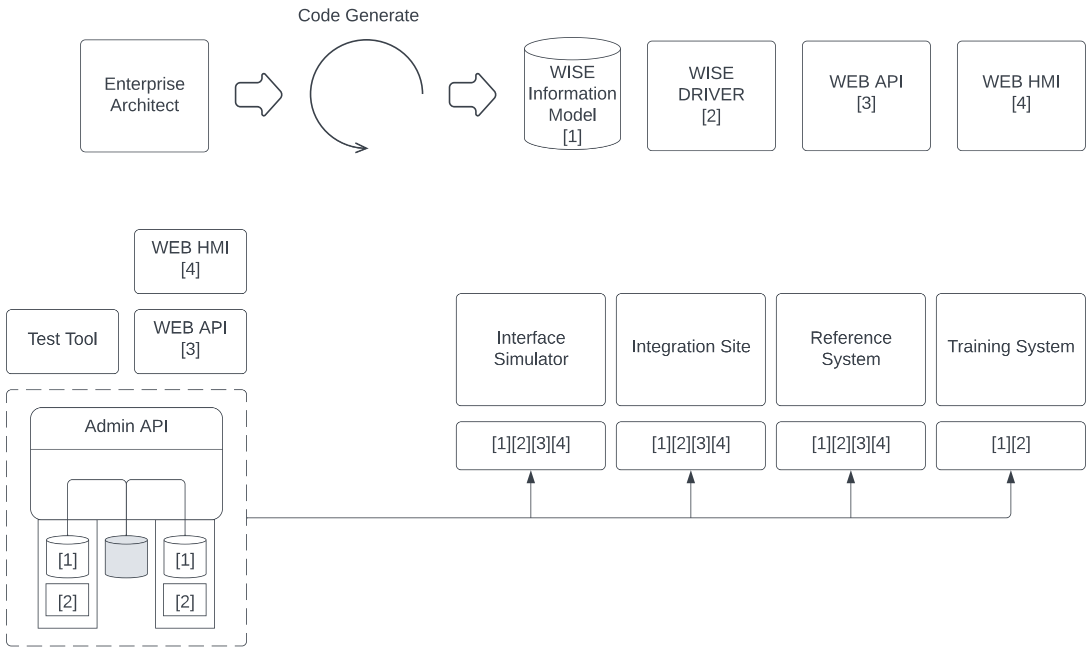


**Figure 1.1:** Reuse of components between different solutions, i.e. interface simulator, integration site, reference system and training system.

**The test tools purpose is to:**

- Send formatted information as a sensor according to a specified standard to a navigation system.
- Receive and look for faults in information formatted for a specified standard from a sensor.
- Show the data in a easy to manage format.

**The purpose of the WISE components is to:**

- Allow communication according to a communication-standard from sensor to web GUI
- Allow communication according to a communication-standard from web GUI to systems requiring sensor data.

## 1.3 Scope

- This thesis prioritizes support for a Log-sensor and its format.
- The supported platform is Windows 10 on PC.
- The web GUI is limited to support Google Chrome, Microsoft Edge and Mozilla Firefox. The supported versions of these browsers are the latest versions during the period of the work being done.
- The compatible transport layer protocols are limited to multicast UDP and Serial port.
- The web GUI shall handles messages being sent with a frequency up to 100 Hz.
- The presented solution in this thesis can support protocols that send field-based information.

# Chapter 2

# Related Work

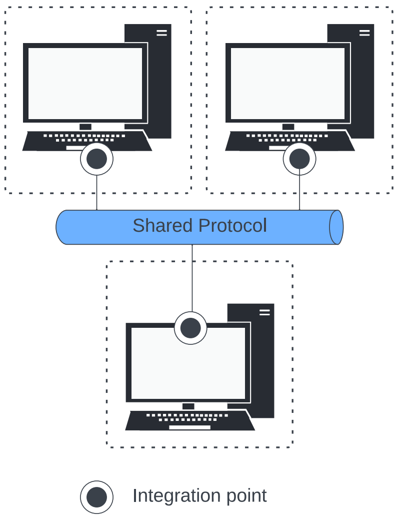
## 2.1 Material UI

Material UI (MUI) is an open-source framework featuring React components that is based on Google's Material Design. MUI was created in 2014 but is still supported today with a team of 12 engineers and 2 designers (MUI, 2022). The goal of the MUI team is to enable quick development of GUI's with components implementing Material Design. MUI was also deemed important from the stakeholders perspective in order to align with Google's Material Design, to increase the consistency among future Saab products.

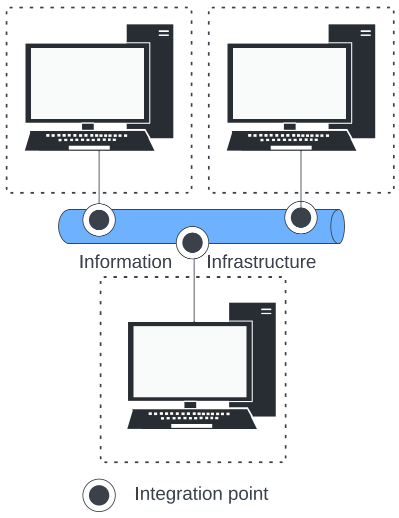
## 2.2 WISE

Building an integrated environment usually is a cumbersome process that involves modifying each of the participating systems so that they can communicate through a shared protocol. This is less than ideal for the system owners since they need to support multiple different protocols or communication standards, in the end leading to higher maintenance costs. This is also especially burdensome when the participating systems owned are controlled by third parties.

WISE, an in-house tool developed by Saab Kockums, is an integration solution allowing different systems to seamlessly exchange information without the need to modify the end system. WISE achieves this by moving the integration work to a central position, known as WISE Connectivity.



**(a)** Traditional approach.



**(b)** Integration with WISE.

**Figure 2.1:** Integration differences, WISE interacts with the participating systems on a network protocol level, preferably using the native protocol of the participating systems.

This means there is no longer a need to support multiple different versions of a protocol. However, by moving the integration work to a central position requires tools to create the compatibility layer. This is where the mapping between different information models comes in. So that each participating system can rely on a specification, but one can change the incoming data to fit an old system. In that way, there is no longer a need to modify the old system to support new ones, hence the majority of the maintenance costs are moved from each of the separate systems to a central system.

With the above said, there is a need to define how the information comes in and out of the new central system. This is where a WISE Drivers comes in. Thereafter is the need to store the data provided by the Driver so that it can be accessed from each of the participating systems. This is where the WISE Data Managers come in. Furthermore, there is a need to define the structure of the data, this is where a WISE Information model comes in. Finally is WISE Connectivity, which ties everything together in order to enable information to be converted and adapted for each of the participating system so that they receive the correct data.

### 2.2.1 WISE Driver

A WISE driver's purpose is to extract the information provided by the underlying protocol and store it in the local database that is handled by a WISE Data Manager. There exists a number of default drivers that comes with the WISE connectivity framework such as Secure Socket Layer (SSL) or null drivers that do nothing. This is handy when it comes to testing different configurations and speeds up the development time.

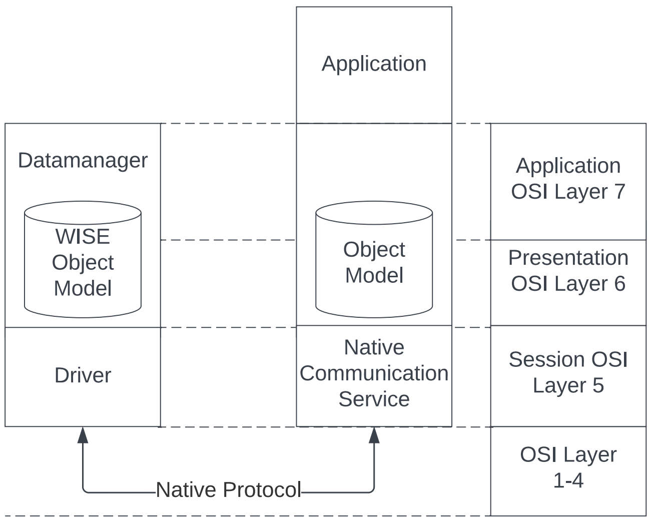


**Figure 2.2:** How a WISE Driver communicates with an application and the corresponding OSI layers.

### 2.2.2 WISE Data Manager

WISE Data Manager is a component that implements the local database that is formatted according to an information model. The WISE database can be described as following database principles although it is stored in the system memory. Which means that WISE is session-based and loses its data after a session has been terminated. The Data Manager uses a WISE Driver to distribute or synchronize the changes made in this local database.

A data manager may also contain a WISE service as can be seen represented in figure 2.3 as an S within the Data Manager for Driver A. A service subscribes to alternations made to its database and is used to extend functionality of an existing system. The service may either be triggered by a modification to its database, for example when a new object or event is created or independently such as internal timers.

### 2.2.3 WISE Information Model

The information model contains the objects and events that is supported by the associated Data Manager and is internally represented using XML. An example of how an Information Model may be defined can be seen in Appendix A. Similarities could be drawn between a Information Model and relational databases such as SQLite. Which uses tables to define the structure of the data.

Objects and events consist of attributes of basic data types such as long, string, vector and blob. An object can be defined as: it is created, lives a period and is then destroyed. Objects can be used to represent entities such as vehicles. Meanwhile events can be described as momentary, *fire and forget* i.e the event must not be handled by any component and may be dropped silently.

### 2.2.4 WISE Connectivity

WISE Connectivity consists of one or several Data Managers and a connectivity layer. WISE Connectivity is what determines the flow of information.

A typical WISE information structure contains several Data Managers. One Data Manager may be configured to contain the common information model, which acts as a source of truth and contains all values from all connected information models, and a backbone driver. The backbone driver can be used to output and input data from the common information model, for example when saving data across sessions. One could export the data from the common information model and import in a later session through the backbone driver. If a case exists where data conflicts between the common information model and another application specific information model, the common information model will be prioritized. This is what "Source of truth" refers to. The data managers that does not contain the common information model and backbone driver can be configured to contain an application specific communication driver and information model.

WISE connectivity also utilizes transactions to handle changes propagated through the network. This is done by grouping changes through the network to assure changes are visible to other parts simultaneously. As a result, any triggers associated with the grouped changes are not made until the transaction is committed. Leading to triggers and consequences being based off consistent data. If a data manager contains different variables that make up a position, for example longitude and latitude, transactions ensure that the two are never acted on when there is old data in one variable and new data in the other. Consequently, any object or event with incorrect format according to WISE will be rolled back across the entire network, hindering errors from reaching other parts.

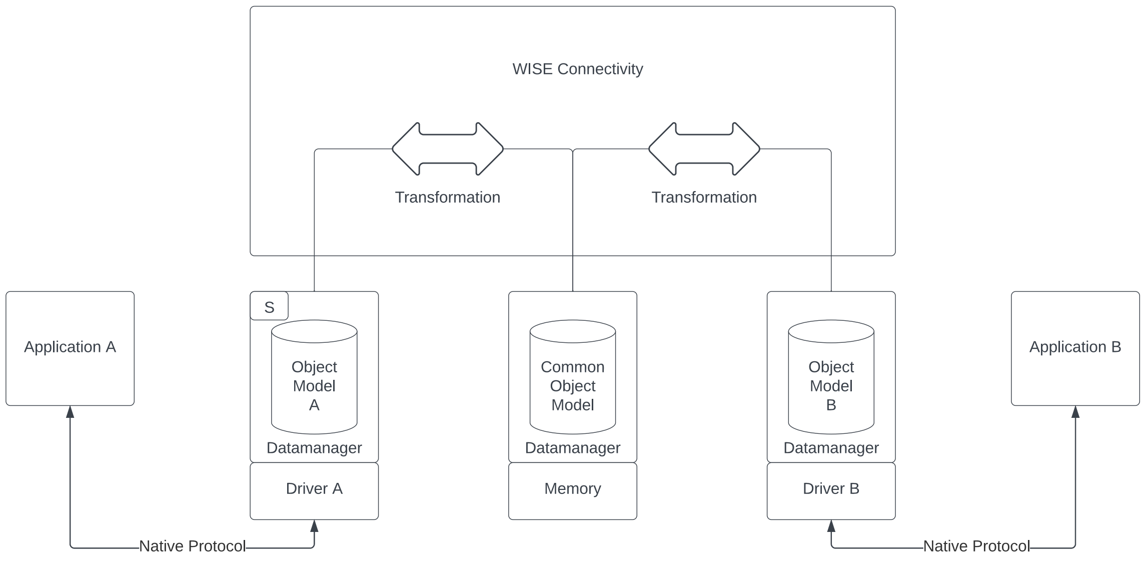


**Figure 2.3:** WISE Connectivity Layer.

As can be seen in figure 2.3 is the typical design of a WISE Connectivity layer. The layer is used in WISE to tie the different Data Manager instances together and allow them to exchange information, which is done in the designer edition of WISE Connectivity. The designer edition is a graphical tool that is used to create the flow of information by mapping between the information models, also known as connections, which can be seen in figure 2.4. In this case, whenever a Network_Settings object is created in the API database, another object is created in the TL database with same attribute values.

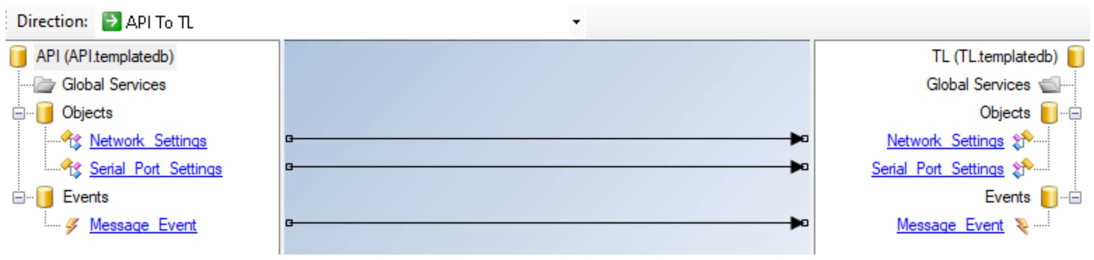


**Figure 2.4:** Example of mapping connections in WISE Designer Edition.

The designer can add triggers and transformations to connections between the attributes of the objects or events. For example, if one information model contains an attribute of type 3D vector, while another information model contains a string, the vector can be mapped to a string through a transformation which outputs the vector in string form into the data manager with the belonging information model with a string, as can be seen in figure 2.5.

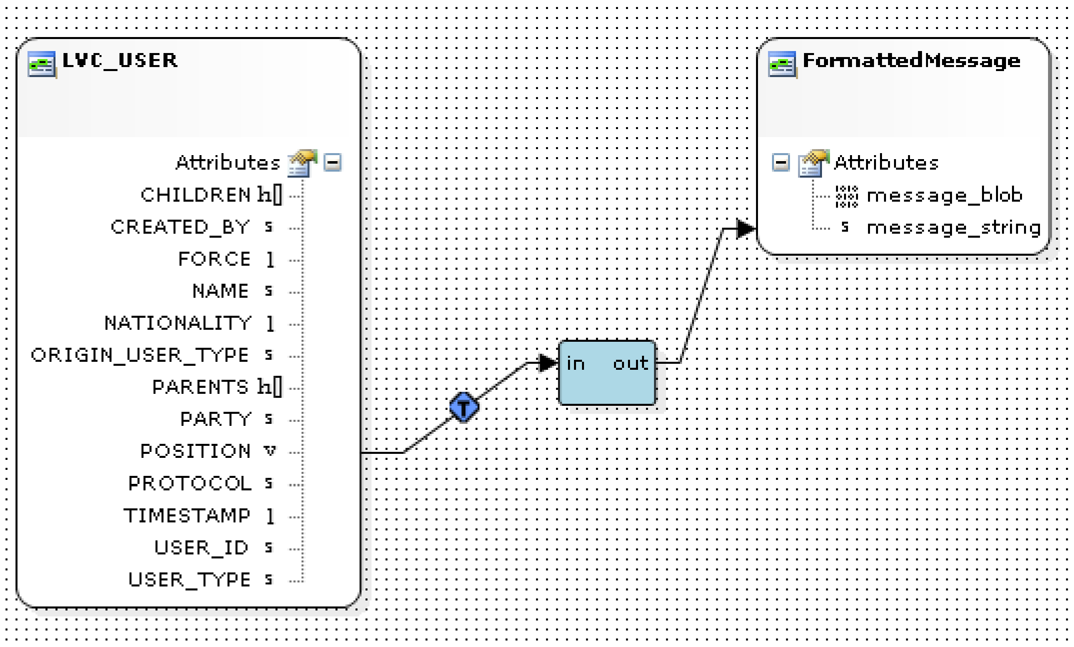


**Figure 2.5:** Example of transformation between the object LVC_USER attribute POSITION v (3D vector) to the event FormattedMessage attribute message_string in WISE Connectivity Designer Edition.

The integration can then be executed using the WISE Connectivity Runtime. WISE Connectivity Runtime is a software program that can be run on a local machine to execute configurations created in the WISE Connectivity Designer Edition.

There also exists tools to test and manipulate WISE components. Other than the designer edition, there exists a test tool to create and view events and objects in WISE Data Managers, which is useful in debugging.

## 2.3 Previous Test tool

There already exists a test tool from a third party, which is partly classified but some general description can be given. The test tool is re-built every time a new sensor needs to be tested and is used in implementation of sensor interfaces, but also testing of those interfaces. A new test tool is built manually according to an IRS which contain what messages the test tool needs to send. The flow of building one of these test tools is roughly:

- Order test tool
- Send IRS for test tool to third party
- Third-party developer(s) reads the IRS, interprets and implements test tool
- Test tool is acceptance-tested
- Test tool is used

Additionally, there is administrative work such as billing. Furthermore, any changes made to the IRS results in the test tool needing further development, testing and billing. Every test tool should be adhering to an IRS, but without any coupling, this needs to be manually checked. Communication is done by e-mail mostly, which means lead-time when waiting for answers.

The test tool can receive, transmit and log specific implemented IP protocols and messages. There exist a limited amount of surrounding functionality as to how messages are viewed and transmitted. However, some implemented functionality encountered during this thesis is unreliable, giving rise to usability issues. Furthermore, the test tool is implemented without WISE, which means integration with an SE is not possible without further steps.

The previous test tool may have set a bar pertaining to this work, but no source code was available as the test tool is a closed source.

## 2.4 Enterprise Architect

Enterprise Architect (EA) is an enterprise COTS application with UML-modeling capabilities (Systems, 2022). The application has several features such as being able to simulate state machines, generate some code based on UML-documentation and more. The generated code is a skeleton for a class representing the modelled object containing a set of variable name and types with empty methods for objects, EA seemingly has no knowledge of programming languages other than a few keywords to generate variables and classes with empty methods containing the names from the UML-model. The mentioned code generation has flaws, such as allowing the user to define whatever variable even though the specified programming language does not support it. For example, a user can define a variable as being of the type "text", without the type being available in C# . EA has the ability to export XML which defines the objects with relations, for example inheritance.

Furthermore, EA has enterprise modelling features outside of this work's scope. Such as modelling business flow, administrative tasks and more.

An example of how requirements could be modelled with EA can be seen in (Meeks, 2015).

## 2.5 Sailsoft's NMEA/AIS Simulator

SailSoft's NMEA/AIS Simulator (Sailsoft, 2014), is a tool for simulating NMEA and AIS components. The tool sends out valid NMEA sentences either via serial port or Ethernet UDP. This reduces the need for field testing, which this thesis also aims to achieve.

Sailsoft's tool has a limited dynamic GUI with the option of using different formatters for different NMEA sentences. The formatters are predetermined and not able to be specified by the user. Sailsoft's tool has some similarity to the one presented in this thesis. Hence, it is deemed to be interesting to make a comparison with, which is done towards the end of this thesis.

# Chapter 3
# Technical Background

With the rise of cloud based solutions, web interfaces are becoming increasingly popular. Furthermore, web based techniques are useful in a lot of ways. For example, the advantage of web is that the server can be run on a machine with a specific OS and the connecting client does not need more than a web-browser. This circumvents the need to download a program and prevents compatibility issues by putting the setup on the server side. Web based development also offer an increasing amount of frameworks and languages (Vuksanovic and Sudarevic, 2011) for efficiently developing professional tools.

## 3.1 React

React is an open-source front-end JavaScript library for building web user interfaces developed and maintained by Meta (Meta Open Source, 2022). React's focus is component-based and declarative. React became popular because of it's fast performance compared to Angular, and is today in some capacity used by many developers, including developers at Apple, Netflix, Microsoft, Airbnb, Twitter, Discord, Coinbase and Zoom.

React contains unique solutions to several concepts to optimize the performance, and React DevTools exist to check performance in the browser. React was used in combination with JSX (Meta Platforms, 2022) which is a syntax extension to JS which allows an interchanging of HTML objects and JS. JSX by default prevents certain code injection vulnerabilities.

### 3.1.1 Components

Components in React take principles from object oriented programming, which are reusable chunks of code that return HTML. React components are similar in structure and logic to classes in a high-level object oriented programming language. This allows a faster learning curve, especially for developers with back-end experience.

A big component of React is hooks. Hooks allow the programmer to use state changes and re-renders to trigger functions, which in turn bypasses the need for a class. Classes in React are only syntactic sugar since JS is functional at its core.

### 3.1.2 State

A state is an object containing data for a specific component. Multiple properties can be contained in state. A component's state is mutable and any changes will trigger a re-render of the component. Changes to state can be made by user input or system-generated events. This is achieved by using "setState()", which tells React to re-render the component.

### 3.1.3 Declarative

Another big benefit with React is that it is declarative. Declarative with regards to rendering means you can tell React what components look like, however the rendering of those components is controlled by React. This results in the developer not having to implement rendering or state behavior, but instead just use state and React will know the rest. This in turn leads to less surface area for bugs to appear, the state implementation is instead controlled by and rigorously tested by the React team during a longer duration. Separating concern also lead to easier understanding of React.

### 3.1.4 React Rendering

In regards to the rendering logic, React uses a virtual Document Object Model (DOM) and a real DOM. The real DOM is a tree representation of objects in the application and is used to render and control components in the application. The virtual DOM is a light-weight abstraction of the real DOM, which is used to efficiently check changes and check for conditions to trigger re-renders. React utilizes the virtual DOM to check for state changes in components and thereafter decide to re-render components with state changes, which in turn lets React re-render affected components only and not re-render the entire website.

A render life-cycle is first creating the virtual and real DOM. Secondly, changes are made to component parameters with any kind of event, which updates the virtual DOM. Thirdly, the virtual and real DOM is compared to check for changes, this is called reconciliation. Finally, the updated real DOM is used to paint components onto the users browser.

In the reconciliation step, where React compares virtual and real DOM, a heuristic algorithm is used with O($n$) time complexity instead of generic algorithms with O($n^3$). The algorithm allows the developer to define what components are static and do not need to be checked. When comparing DOMs in React and the compared objects are different types, a full rebuild of the DOM is made, but when objects are of the same type, attributes are compared and only that node in the DOM tree is updated. This means changing components, for example inserting a new image in the view when clicking a button, triggers a larger re-render; however, only changing attributes, for example clicking a button which changes the color of another object, will result in an update of affected components only.

## 3.2 REST API

Representational state transfer application programming interface (REST API) is an architecture style that allows applications or devices to connect and communicate with each other via for example HTTP requests (IBM Cloud Education, 2021). These requests can perform standard functions such as creating, reading, updating and deleting records (CRUD) in a database or similar. As an example, a GET request could be used to get hold of a record, while a POST request could be used to add a new record to a database. The information provided by REST can be delivered in nearly any type of format including JSON, HTML and plain text. Whereas JSON is the most popular choice of them all mainly due to it being readable by both humans and machines. Due to REST's relative high flexibility and freedom it has become widely adopted.

## 3.3 XML

Extensible Markup Language (XML) is a standardized text format that is easy to extend, structure and validate (Khare and Rifkin, 1997). The purpose of the XML is to be used in serialization, meaning converting an XML model to protocol structure. Another benefit with XML is the ability to validate the structure and content of an XML file with a XML Schema Definition (XSD) file (W3 Schools, 2022). XSD is primarily used to define the elements, attributes and data types the document can contain. The information in the XSD is used to verify if each element, attribute or data type in the document matches its description.

## 3.4 NMEA 0183

NMEA is a standard for electrical and data specification between marine electronics, for example sensors and several other instruments. It is controlled by the National Marine Electronics Association (NMEA) (Raymond, 2018). It is slowly being replaced by NMEA 2000 but remains the norm in commercial shipping.

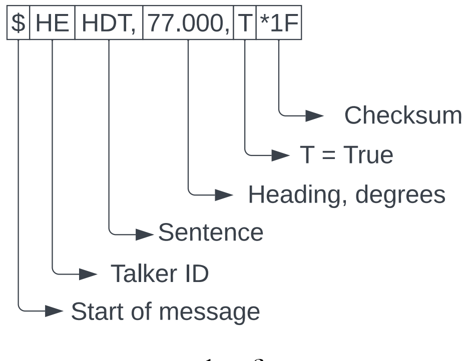


**Figure 3.1:** Example of an NMEA message.

NMEA is text based and a typical string can look like in figure 3.1 above. The first part of the string "HEHDT" contains information of the talker ID "HE", which is a string representation of the type of unit the sender is. In this case "HE" is a north-seeking gyro. The beginning part also contains "HDT" which is the type of message sent. HDT is according to the NMEA standard, a message with a heading value in degrees, a T for true and a checksum in hexadecimal after the asterisk. The checksum is a typical XOR of the message fields (Rietman, 2008).

### 3.4.1 Sensors

There are two types of sensors that are especially relevant for this project. First and foremost is the LOG-sensor, which is a sensor that can send out NMEA and other sentences with a given frequency. The LOG-Sensor consists of a speed log with two subsystems, the passive log and the active log. The passive log measures speed through water. The active speed log is a correlation log that measures transversal and longitudinal speed over ground, as well as longitudinal speed through water and distance travelled. Additionally, the LOG-sensor can measure relative depth below keel using an echo sounder. To emulate a LOG-sensor, an available Inertial Navigation System (INS) is used in this project. The INS used is a North-seeking Gyroscope which gives the orientation in degrees. The INS uses the NMEA protocol the same as a LOG-sensor would.

### 3.4.2 Applications

Warship electronic chart display and information (WECDIS), which is similar to the civil version electronic chart display and information system (ECDIS). The WECDIS is used to gather and display data pertaining to different sensors and the environment, which is mainly used to navigate as it can replace the need for a paper map. The WECDIS can receive NMEA sentences.

## 3.5 Azure DevOps

Azure DevOps is a platform developed by Microsoft similar to GitHub, with a focus on enterprise software development (Microsoft, 2022). The features include version control using git, automated builds, testing and project management.

The tool for implementing continuous integration in Azure DevOps is called Pipelines. Setting up a pipeline is deciding which actions to take with certain triggers, such as when a git push is done. The actions include building the project to check for compile errors and running tests on the application. There are also standard actions such as zipping files and creating build artifacts and many more. Azure was used to organize this project and access was granted through Saab Kockums.

# Chapter 4
# Approach

The approach was mainly iterative and agile. The beginning consisted of mostly elicitation and learning, the middle consisted mainly of development and design, while the ending contained more testing, evaluation, and writing. Work was done in an office environment with close proximity to involved parts. Collaboration was done through discussions, interviews and workshops.

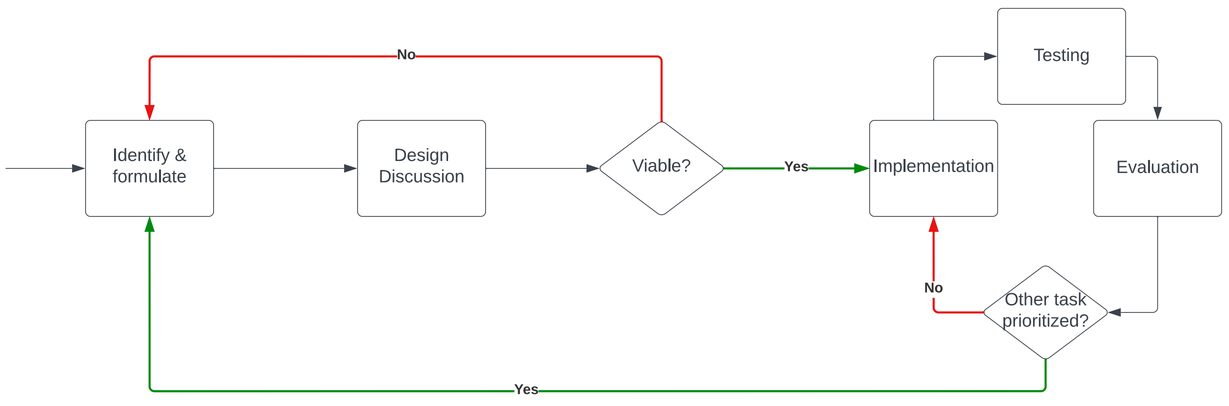


**Figure 4.1:** Process for each functionality.

## 4.1 Process

Our process for developing the solution stemmed from agile practices and thus from the agile manifesto (Beck et al., 2001). Inspiration was also taken from Kanban. This is especially apparent when the report was developed iteratively, parallel with the work and throughout the project. The problems and designs were solved and evaluated when needed, to allow the postponing of solutions to a point where knowledge was maximized. There was a strive to be flexible and to be able to adjust the scope or design if it was deemed advantageous. Therefore the principle of doing work only when necessary was adopted, to not have to change functionality or risk doing the wrong functionality with lacking information to begin with. The principle meant doing work when enough information existed. The determination of having "enough" information, was done through intuition and eliciting a set of statements from the stakeholder that supported that functionality.

### 4.1.1 Daily Stand-up

A relevant step in the process was the daily stand-up, which was not strictly always daily but nonetheless used frequently, the daily stand-up was used when there was uncertainty about what to do or to when there was a need to synchronize the work done between us. More specifically, if it already was known what had to be done and it had already been discussed with no information change, the daily stand-up was skipped or postponed. The involved parts in the daily stand-up were the developers of the solution. An example of what it could look like was that the last person to arrive to the office would come up to the other developer and ask what was being done or had been done and if there were any comments or obstacles in the current tasks. Thereafter the same questions would be asked back. The daily stand-up allows for close collaborative work and the commitment to a daily or weekly goal contributes with motivation to get that work done. The Kanban board was not implemented digitally because of the small scale of the developer team, it was in this case deemed sufficient by us to write needed functionality on a list and to cross it off as it was done. New task lists were made when the work on the previous list was completed. In some cases there could be left over tasks which were carried over to the next list or dropped if deemed no longer a priority. This allowed for a more lightweight approach which aligns with the agile principle "Individuals and interactions over processes and tools" and "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."

### 4.1.2 Office Environment

By working in an office there was also possibility for collaboration with the product owner and other competent employees. This was most apparent with the interviews where needs were elicited which were done whenever uncertainty appeared. Sessions were held towards the later stages after the MVP was made where the GUI was shown in a structured manner, by going over all the functionality, especially the latest functionality, and then asking for their opinion on the functionality. After the specific feedback was received general design opinions were gathered and possible future functionality was discussed to implement next, or if too large for the scope, documented in this report. Finally, an evaluation interview was held when the development slowed down where the included parts reflected on efficiency and fit of the solution to the problem.

## 4.2 Method

To start the project a start-date and a presentation date had to be planned. Then the available hours were approximated. For a bachelor's thesis with 22.5hp, one person is expected to work about 600 hours. Thus, with two people, there were 1200 hours to work with. Estimating the hours for various tasks and putting everything in a Google Sheets resulted in a time plan. The time plan was also used during the work to report hours worked. Both effective hours, the active hours spent towards a specific goal, and total hours, the hours spent in the Saab Kockums office. This metric allows us to evaluate our productivity as well as compare if the time plan is followed. Though, it is necessary to take into consideration the informal nature of the document, therefore the accuracy is questionable due to possibility to forget to report time.

With a security-oriented company, the initial phase of the project was slightly slower due to the need to go through various processes and on-boarding.

Initially information had to be gathered about the task and determine what information was available already. Interviews and meetings were arranged with employees and consultants who were involved or had experience with WISE and Enterprise Architect. The relevant diagrams received were retained and discussions about scope resulted in summaries in a document with an initial description of the project. Relevant references were gathered in the git repository and evaluated early in the process. Google Scholar was utilized to find techniques and academic articles for model-based generation of various components. The model-based research articles were useful for comparing designs and general concepts to be utilized. But the ones found were general tangent solutions and not directly applicable to our problem.

### 4.2.1 Prototype

An evolutionary prototype was developed early while more of WISE and React was to be investigated and learned about. The prototype was iteratively refactored and developed to become the solution. The evolution of the prototype into solution can be seen in Appendix B. Some alternative prototypes were made as git branches, mostly to briefly experiment with design such as the last image seen in Appendix B or when experimenting with MUI.

To create a user interface that is appealing, design principles were taken into consideration. They include principles found on Google by searching for "design principles web". This search inspired reflecting on purpose of every view opposed to overwhelming the user with everything at once and additionally reflecting on the notion that every view needed a logical structure that was consistent throughout the website. The search also highlighted concepts such as white-space, balance, visual hierarchy, use of colour, typography, contrast and imagery to create a coherent design. Doing work for a larger company meant design documents already existed, however, they were somewhat flexible. To improve the user interface even more, feedback was gathered from the product owner and various involved parts through semi-structured interviews.

### 4.2.2 Brainstorming

For each new feature a brainstorming session was completed to decide design choices, here the pros and cons of various designs were weighed against each other. Thereafter, some form of diagram or drawing was produced to ensure the mental image of the design was synchronized. Weighing pros and cons can include comparing experience with tools, scalability, maintainability and time required to implement compared to the potential impact of the feature. It also had to be taken into consideration how well the feature brought the product closer to solving the current problem. This results in some decisions being made that for example are not optimal for the long term, but are viable to implement in the given time span. When the functionality was backed up by a need and deemed necessary, it was prototyped and thereafter iterated on to increase its functionality. The diagrams or drawings were put on a whiteboard or in a notebook. In the case of the whiteboard, the diagrams and information was kept up to date and relevant for the design that was decided on at the given time.

### 4.2.3 Logbook

For each week an entry into an informal logbook was made containing information about what was done that week and any goals or obstacles that appeared. The main idea behind the usage of a logbook was to gain a better understanding of the choices that were made throughout and was also used as a reference for writing this thesis.

### 4.2.4 Pair Programming

When programming more complex functionality or design-heavy functionality and often at the beginning of new features, pair programming was done improve the quality of code (Hannay et al., 2009). Pair programming not only worked to increase the quality of the code but also as a catalyst for more design discussions. When doing back-end API work, this enables information sharing about what API points to fetch or post to.

### 4.2.5 Object Oriented Design

When programming, an effort was made to follow the SOLID principles to facilitate future development in the back-end. Such as classes having a single coherent responsibility and being open for extension but closed for modification. Liskov substitution principle was also followed with back-end formatting interfaces and conversion from XML for different protocols. Interface segregation principle is encouraged when extending for more protocols and conversions by not having too large interfaces, and bundling methods that are to be used together. The dependency inversion principle has also been taken into account by designing the system to encourage implementing interfaces, and not encouraging implementation of low-level classes.

### 4.2.6 Class vs Functional components in React

In the front-end, React with JS was used, which contain classes and object oriented syntactic sugar but also functional components. The functional components were chosen over the class components because of assumed increased support for functional components in the future from React compared to classes. The React team have previously encouraged functional components (Alpert et al., 2018). As this software is meant to be further developed if viable, it is advantageous if it supports a long shelf-life by utilizing future optimizations to increase its future performance. Functional components also decrease the amount of total code.

## 4.3 Division of work

Much of the work was done collaboratively due to close proximity when working. However, Christoffer did the overwhelming majority of the back-end programming related to WISE, setting up the REST API and implementing the CJEX library. Jonas was more focused on the front-end, gathering information and creating documentation, however he also helped Christoffer with deliberating different solutions and gave his insight on the back-end.

## 4.4 Experimental Setup

First and foremost, to check if the communication was functional, several communication pathways had to be investigated. These include:

- GUI to GUI. (Ethernet)
- GUI to the previous test tool. (Ethernet)
- Previous test tool to GUI. (Ethernet)
- INS to GUI. (Ethernet)
- Saab Kockums simulator (SE) to GUI.

The tests done with the communication was done to assure the messages arrived with the complete data and a correct validation in the GUI based on structure and value-ranges specified in the XML. In the test cases the specified NMEA sentences are HDT, DBT and VBW and a custom JSON message.

### 4.4.1 GUI to GUI

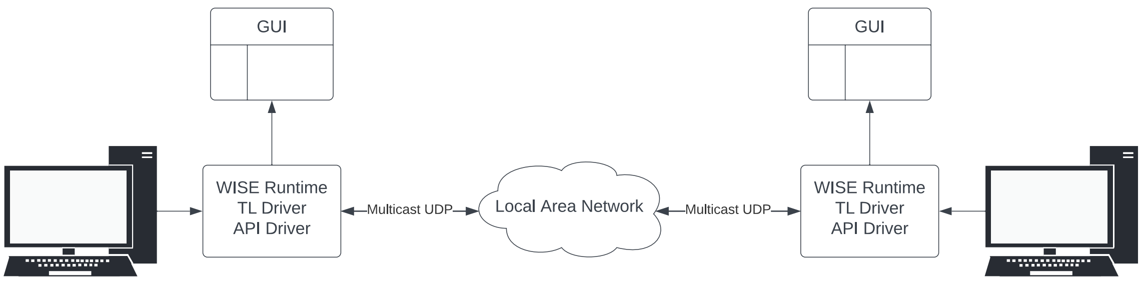


**Figure 4.2:** Configuration using the built TL and API drivers running on two different machines.

Testing the communication from GUI to GUI was the first of the tests done. This was realized by having two different PC's running respective WISE components and the GUI. One part would send to the other and vice versa. The messages were controlled to have the same information retrieved as was sent. In later stages this test setup was repeatedly used to test the validation by sending intentionally wrong messages and checking the error codes for any inaccuracy. If a field was removed, an error code for missing field was expected. If a faulty checksum was sent, the checksum error code was expected. The connection was made using ethernet and with specified ports to multicast UDP from. The PC's were on the same network.

### 4.4.2 GUI to Previous Test Tool

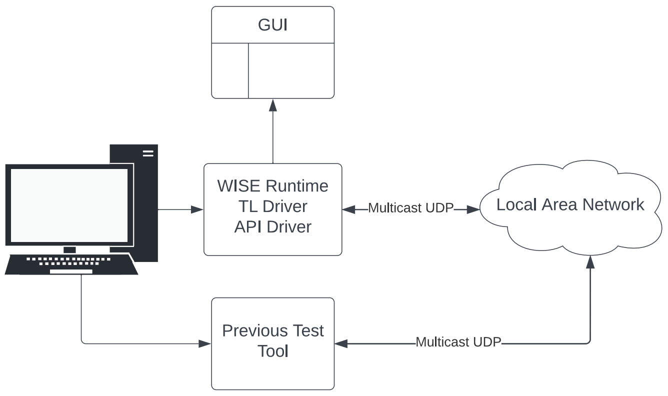


**Figure 4.3:** Configuration using the built TL and API driver and previous test tool running on the same machine.

The communication between GUI to previous test tool was performed both to the same PC and to another PC. This test was done to ensure the format of the messages was consistent with the format of the previous test tool in the case of NMEA messages.

The previous test tool contained automated sending according to a frequency, which was useful to stress test the GUI to test the frequency at which problems appeared. This could also be utilized to see how many messages the GUI could store and show before showing any slow down.

### 4.4.3 INS to GUI

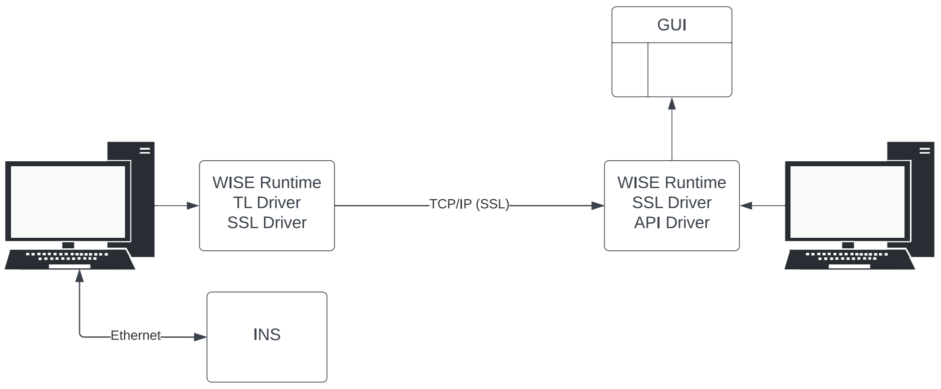


**Figure 4.4:** Configuration using the built TL and API drivers and existing WISE SSL Driver with a INS connected to a machine.

The WISE components were setup on a PC connected to the INS by ethernet. The INS can send various messages, but only HEHDT was tested to assert a functioning communication between the PC and the sensor. The INS in its built state is assumed to send NMEA messages correctly formatted, and therefore any messages specified in XML according to the NMEA standard received is assumed to be shown as valid when testing. The INS can send according to other protocols than NMEA but these are expected to be shown as invalid when receiving in the GUI because they are not specified in XML in the test cases.

### 4.4.4 SE to GUI

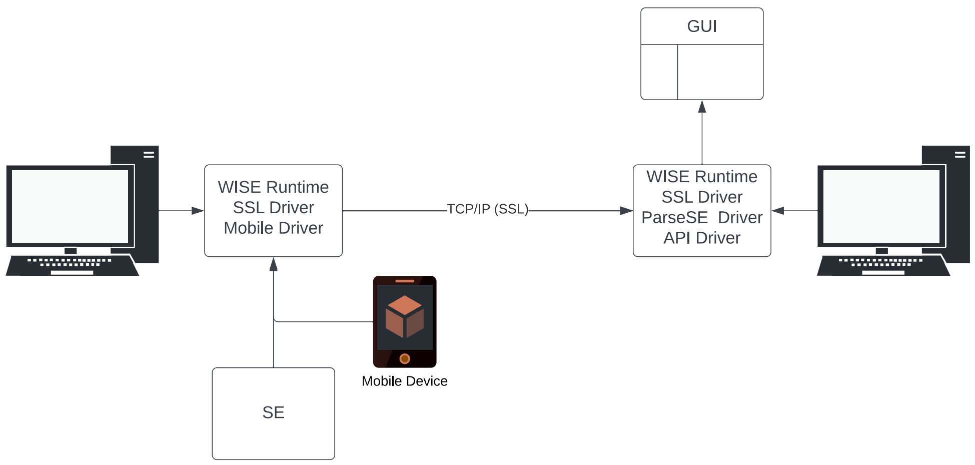


**Figure 4.5:** Configuration using the built ParseSE and API Driver with existing mobile and SSL drivers running a SE on a machine with connected mobile device.

The ParseSE driver was tested to check for correct parsing of data from a Synthetic environment (SE). This was setup with WISE connectivity runtime running on two separate computers. One PC was running the SE server and WISE connectivity runtime configured with a SSL driver with the SE information model. The SSL driver acted as the server and forwarded the information via TCP to the client configured on the other PC running a WISE SSL driver, ParseSE driver and API driver. A mobile device with a gyroscope was used to send a position in degrees with a frequency of up to 100 Hz. The position in degrees was sent using the HDT sentence.

### 4.4.5 MUI Component Tests

Some components have been tested as MUI components that are relevant to discuss. The tests performed included changing the tags from <div> and <p> to MUI Accordion and MUI Typography and observing the difference in input-latency from perceived click to perceived completed expansion of the div. The code differences are shown in Appendix C. This test was done without the use of further optimization using React hooks in both cases. React offers hooks such as UseMemo(), which can be utilized to optimize tasks by saving values previously rendered and only re-rendering when a shallow comparison of the components input parameters results in detected change.

### 4.4.6 Test Tool Comparison

The previous test tool is partly confidential, to compare the two solutions requires some form of anonymity in the comparison. This is done by ranking functionality from 1-3, with 1 being necessary functionality for MVP, 2 being functionality that provides value but is not required for MVP, 3 being unimportant QOL functionality. MVP in this case is a test tool which can view and send messages with some logging containing previous messages.

The list of functionalities is composed before any ranking and the ranking will be done by someone not directly involved in the project to minimize conflict of interest. When the ranking of importance for each functionality is done, the functionality is checked for in both solutions and deemed implemented or not implemented. When this is completed, a result is composed by counting the number of functionality in each importance category which is implemented in the respective solutions. If uncertainty exists for a functionality or it is partly implemented, it is counted separately to showcase a strict result and an optimistic result. This can then be compared to yield an estimated count of how much more of each functionality for each importance category is implemented in the solutions.

#### Comparison Example

Test Tools could have the ability to send and receive messages, while one has the ability to display a PDF and the other could send automatic messages with a given frequency. The ranking would look something along the lines of:

- Send Message
- Receive Message
- Display PDF
- Automatic send

Someone would then rank the functionality 1-3:

- Send Message: 1
- Receive Message: 1
- Display PDF: 3
- Automatic send: 2

Finally one would count the functionality found in each tool. Test tool 1 might have send, automatic send and a substandard implemented receive message, for example meaning it might sometimes break. Test tool 2 has display PDF and send message. The result would be:

| Category | Test Tool 1 | Test Tool 2 |
|---|---|---|
| 1 | 1(2) | 1 |
| 2 | 1 | 0 |
| 3 | 0 | 1 |

**Table 4.1:** Example-result of test tool comparison where the parenthesis contain a count of even partially or substandard functionality.

# Chapter 5
# Implementation

The WISE Drivers ParseSE, TLDriver and APIDriver were implemented with Visual studio 2017 in a mix of XML, C# , Newtonsoft JSON and WISE Connectivity SDK with .NET framework 4.6.1. The CJEX library was written using C# in Visual Studio 2022 using .NET framework 4.6.1 and Newtonsoft JSON. The solution's implemented functionality can be seen in Appendix D.

A balance was made between flexibility and rigidity for the implementation, leaning more towards flexibility. Both have their own trade-offs, whereas a more rigid system is generally less error prone although may not cover as many use cases and is harder to edit compared with a more flexible system. By allowing for example the user to edit the GUI in real-time made the system lean heavily towards a more flexible state. Although, this could lead to a number of issues. To minimize this, a number of error handling procedures where implemented such as validating the XML file so that it follows the reference as seen in Appendix E.

## 5.1 Azure DevOps Pipelines

Build pipelines were implemented in Azure DevOps to experiment with pipelines and automated testing. In total 10 automated unit tests were implemented for the CJEX library through a pipeline triggered by push to any branch. Exceptions were made when the push to the repository contained changes to the front-end only, in this case Azure DevOps did not build. The pipeline was used to assure the project could build and the unit tests were passed. A secondary pipeline was made with a manual trigger which was used to output files used for running the solution. The pipelines uses NuGet restore on a solution and then builds that solution, and then does that for all the solutions specified. The last stage is deleting files used in the builds to clean up. The manually triggered pipeline does the same but with the added step of outputting a .zip file, named after build ID, containing .dll files.

## 5.2 XML

The XML file used for generating the structure of various input-fields contain information pertaining to what protocol is to be used, what sentence type in that protocol and various Fields organized in groups to represent logical coherence. The fields themselves are defined with a datatype and acceptable range parameters. Every field can have several options. For the complete structure of the XML file, see Appendix E.

Given a field is of type string and contains options, a "drop-down" menu is instantiated to represent that value in the GUI. Similar logic is used to determine if ints or floats will be represented using "drop-down" as strict options or just a box with the functionality to write any value.

The XML format was not made to support EA out of the box. EA was deemed too time consuming to work with and thus the XML uses another format made from the ground up in this project. However, EA can export XML with similar structure. The parsing of the XML is therefore only in need of minimal changes to work with EA.

## 5.3 GUI

The GUI is a web based graphical user interface that was written in Visual Studio Code using HTML, CSS, JavaScript and ReactJS. The GUI also used Material UI and Node.js. To fetch information sent from the WISE components, the GUI uses the REST-API by fetching JSON-formatted packets. The information sent is used to generate input fields based on the datatype specified, the input field, with the help of a switch-statement, generates a component compatible with that datatype. InputField.js utilizes some ternary operators to check if required props are defined, otherwise default values are returned. The different types implemented is shown in Appendix E.

Some complex components such as the multi-select originates from the MUI component library. The multi-select has the ability to auto-complete input. If the option for "invalid" exists and the user types "inv" then presses Enter, "invalid" is selected. Using Materials UI is convenient if it is necessary to save time, such as with components implementing more complex filtering behaviour. However, styling MUI components requires learning API's that frequently change and sometimes are less intuitive. Thus MUI components was used only when needed to reduce dependency on third-party.

## 5.4 CJEX

Christoffer Jonas Examensarbete (CJEX) is a standalone library made to handle the parsing and validation of protocols with a given standard. The main purpose of the library is to extract the fields from the defined XML-file and create the protocols with the associated fields. CJEX also has the ability to generate a WISE information model based on the XML-file. The generated information model could later be used in WISE designer edition in combination with ParseSE Driver.

A couple of conscious choices were made in regards to following certain object oriented design patterns. Including strategy, factory and adapter pattern. Whereas the strategy pattern is the most compelling. The main reason behind implementing this pattern was to make it straightforward to replace the current implemented XML converter without the need to make major overhauls. All that has to be done is implement the interface for the converter and then change the strategy used to the new converter as can be seen below.

```
public class GenerateStructure
{
  private IXMLConverter converter;
  public GenerateStructure(IXMLConverter converter)
  {
    this.converter = converter;
  }

  public void SetConverter(IXMLConverter converter)
  {
    this.converter = converter;
  }

  public void GenerateProtocols()
  {
    converter.GenerateProtocols();
  }

  public List<IProtocol> GetProtocols()
  {
    return converter.GetProtocols();
  }
}
```

CJEX is able to support all types of field-based protocols and out of the box can support JSON and NMEA. This meant a generic approach had to be taken in order to support future extensions. The Protocols implement the interface IProtocol with a few methods as can be seen below. Whereas the most important ones are FormatMessage, Verify and GetFields.

```
public interface IProtocol
{
  /// <summary>
  /// Formats fields according to protocol,
  /// expects the values to come in order according to the
     protocol definition
  /// </summary>
  /// <param name="values"> value of the fields </param>
  /// <returns> </returns>
  byte[] FormatMessage(List<string> values);

  /// <summary>
  /// Verifies that data is according to the protocol
  /// </summary>
  /// <param name="data"> The bytes to verify </param>
  /// <param name="sb"> Contains if any the error messages </
     param>
```

```
    /// <param name="dict">Contains the id and value of a field
       </param>
    /// <returns> True if message is following the protocol </
       returns>
    bool Verify(byte[] data, StringBuilder sb, Dictionary<string,
        string> dict);

    Fields GetFields();

    string GetID();

    string ToString();
  }
```

The FormatMessage method returns a byte array of the formatted fields according to how the protocol is defined in the XML and takes in a list of string with values. The assumption was made that the list is sorted in the order that the protocol was defined. The user normally does not need to take this into account when using the GUI or ParseSE, since it is generated and follows this structure. Although when defining the XML-file this has to be taken into account.

The Verify method should return true if the data is formatted correctly according to the protocol. If for any reason the data is not formatted correctly, for example an NMEA message with incorrect checksum. This method shall return false and a log of what when wrong (StringBuilder) which the GUI will use to display any error messages to the user. Lastly is the dictionary with a field id as key and value of the field extracted from the data. This is used by the GUI to display the separate field values of the protocol.

The GetFields method should return the fields associated with the protocol. This is mainly used by the web GUI when generating the layout and inputs. As can be seen below is a number of attributes that a field can contain.

```
public class Field
{
  public string name { get; set; }
  public object value { get; set; }
  public string type { get; set; }
  public string id { get; set; }
  public int groupID { get; set; }
  public object min { get; set; }
  public object max { get; set; }
  public int? minLength { get; set; }
  public int? maxLength { get; set; }
  public bool required { get; set; }
  public List<KeyValuePair<string, string>> options { get; set;
      }
}
```

## 5.5 REST-API

The communication used a REST API with a proxy built using Microsoft.AspNetCore.Mvc in Visual Studio 2017. The Microsoft.AspNetCore.Mvc allows the use of routes to define what back-end API-method to use. For example if a user puts /apimethod in the end of the URL, the API checks for that route and if it is defined.

The API is implemented using a proxy following the design described in figure 5.1. Using a proxy is essentially decoupling the API from the front- or back-end. This makes extending the function of the API to include monitoring easier while being relatively lightweight.

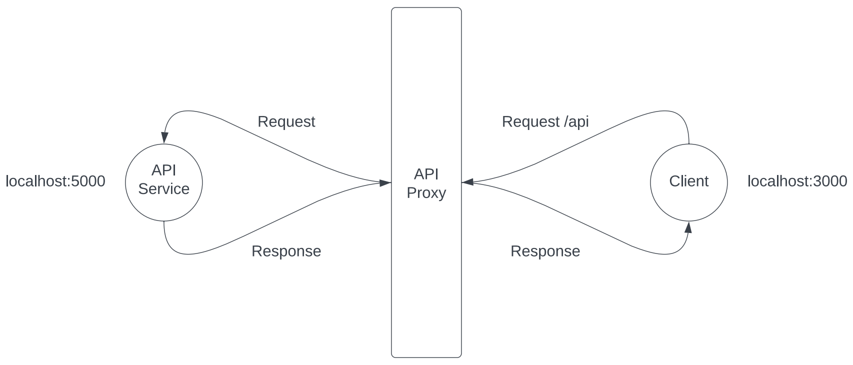


**Figure 5.1:** Proxy setup.

## 5.6 WISE

A total of three wise drivers were built in this thesis, Transmit-listen (TL) Driver, API Driver and the Parse synthetic environment (ParseSE) Driver.

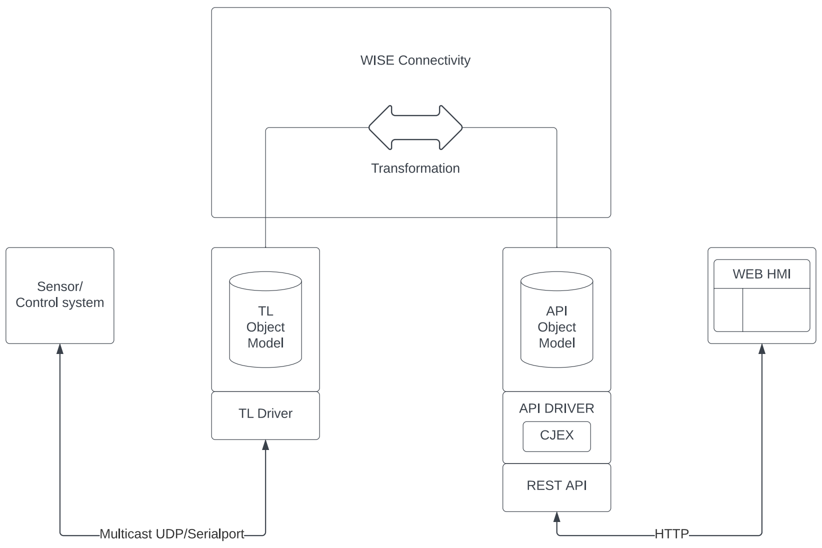


**Figure 5.2:** Connectivity layer for the web GUI.

The connectivity layer was configured for the TL and API driver according to the figure 5.2 above.

### 5.6.1 TL Driver

The responsibility of the TL Driver is to listen and to send data over the multi-cast UDP or serial-port depending on how the user configures it in the GUI. The TL driver waits for a change in its object model and acts upon it depending on the type.

The following can occur:

- A Settings object is created or updated
- A Message event is created
- It receives data from another system, for example a LOG-sensor

Initially on startup, the API Driver creates a new settings object for a UDP connection with a standard port, multi-cast address and the local IP of the current computer running the API Driver. This in turn via a WISE connectivity transformation creates an identical object in the TL object model with the same attribute values. When that occurs, the TL Driver opens the connection and starts listening. This was done in order to avoid having the user configure the connection for each session.

When a new settings object is created or updated, the TL Driver checks if a connection is currently open and if so stops it. Then it establishes a new connection according to the configuration.

When a message event is created, the driver checks if a current connection is open, and if so takes out the attributes of the event and sends the data on the underlying protocol.

When the TL Driver receives data from another system it creates a new event with the extracted data, determines who sent the message and adds it to its local database.

### 5.6.2 API Driver

The responsibility of the API Driver is to receive information from WISE connectivity and forward that information over the REST API to the GUI. It also is responsible for creating the layout for the web GUI with the help from CJEX as shown in figure 5.2. On initialization, it starts up the API service using a kestrel web server and configures it with the MVC-pattern.

The following can occur with the API Driver:

- It receives an event
- An API Request is made

When the API driver receives an event, it means new data has been sent to it. The Driver then uses CJEX to try and validate the data according to the defined protocols in the XML-file and then adds it to its data log.

When an API request is made, the API controller forwards it to its associated service. It may look like below.

```
    private IService service;
    public TemplateController(IService service)
    {
        this.service = service;
    }

    [HttpPost]
    [Route("clear")]
    public void ClearMessages()
    {
        service.ClearMessages();
    }
```

In this case whenever an API POST request is made to the web server for the specified URL "localhost:5000/api/clear" it tells the service that it should clear the log of messages.

### 5.6.3 ParseSE Driver

As part of this solution, ParseSE was built. ParseSE is a standalone generic WISE Driver that is able to convert attributes from, as an example, SE and outputs a formatted message event accordingly to the protocol that was mapped in WISE. This driver could be combined with other WISE Drivers such as with TL Driver to send out the formatted message via multicast UDP, serial port or combined with the API Driver to send the data directly over to the web GUI.

As can be seen in the figures below is how the ParseSE connectivity file may be configured and the chain of events after a transformation have occurred.

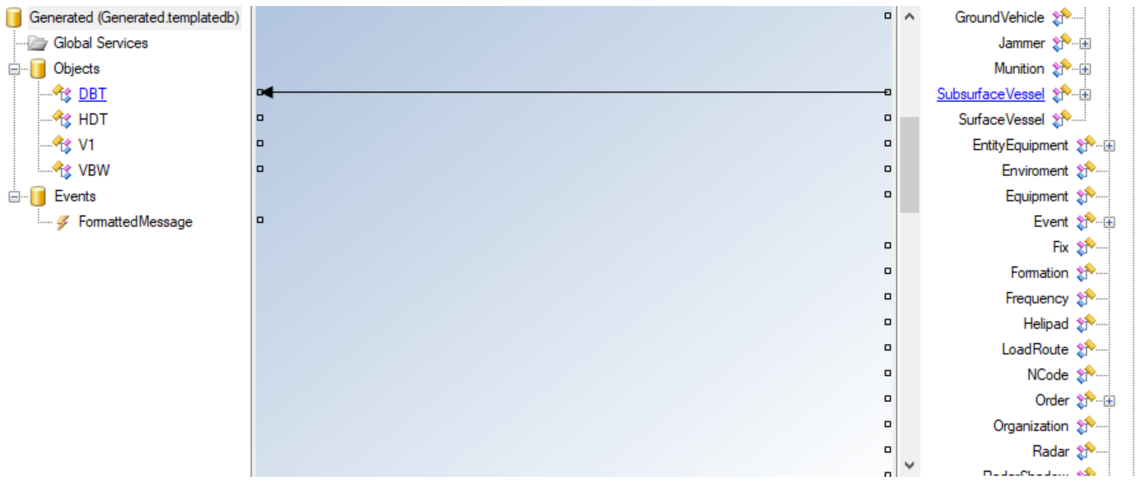


**Figure 5.3:** How a WISE connection may look like to the generated information model from a SE information model.

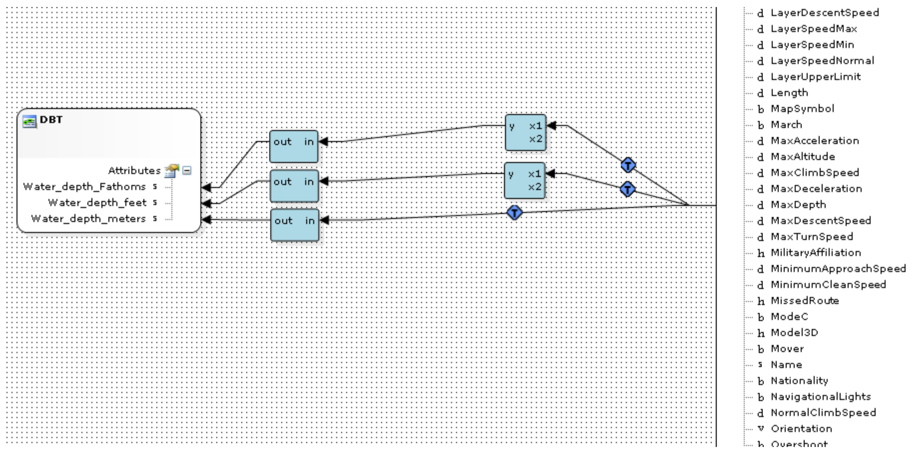
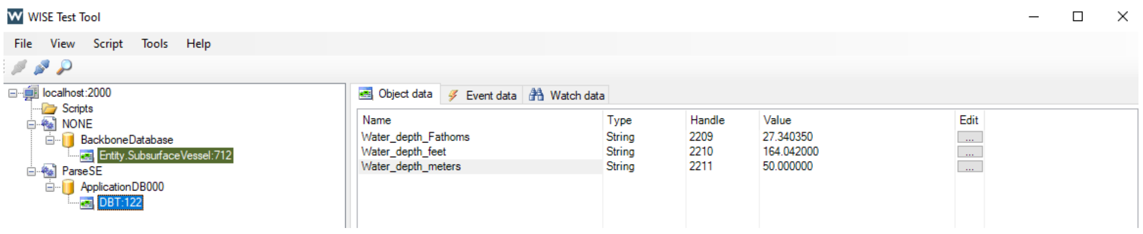


**Figure 5.4:** How a WISE transformation may look like for the generated information model from another information model. Here can be seen a math transformation (the ones with the y-output), where it converts the MaxDepth to the different units in Water_depth for a NMEA DBT object.

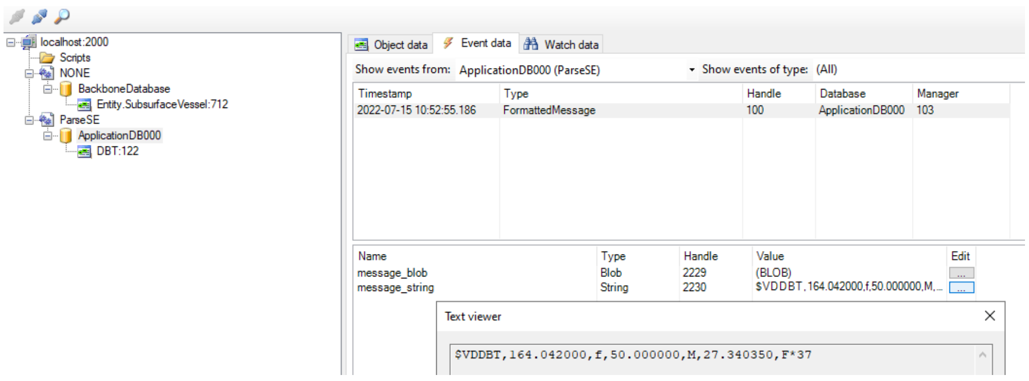
As shown in figure 5.4 whenever a subsurface vessel is created or the vessels MaxDepth attribute updates a transformation will occur which creates a DBT object in the ParseSE database.



| Name | Type | Handle | Value | Edit |
|---|---|---|---|---|
| Water_depth_Fathoms | String | 2209 | 27.340350 | ... |
| Water_depth_feet | String | 2210 | 164.042000 | ... |
| Water_depth_meters | String | 2211 | 50.000000 | ... |

**Figure 5.5:** How it looks like in WISE Test tool when a transformation takes place.

The driver then extracts the attributes from the object created in its database and creates a new message event formatted according to the protocol standards as can be seen in figure 5.6.



Show events from: ApplicationDB000 (ParseSE) Show events of type: (All)

| Timestamp | Type | Handle | Database | Manager |
|---|---|---|---|---|
| 2022-07-15 10:52:55.186 | FormattedMessage | 100 | ApplicationDB000 | 103 |

| Name | Type | Handle | Value | Edit |
|---|---|---|---|---|
| message_blob | Blob | 2229 | (BLOB) | ... |
| message_string | String | 2230 | $VDDBT,164.042000,f,50.000000,M,... | ... |

```
$VDDBT,164.042000,f,50.000000,M,27.340350,F*37
```

**Figure 5.6:** The created event with the message formatted accordingly.

# Chapter 6

# Result

Interviews along the process were helpful to create discussion about where the software was headed and how it related to business needs. Every interview session resulted in useful reflection around architecture or potential functionality. Showcasing the GUI and discussing design resulted in receiving useful feedback.

The evaluation interview at the end was held with the product owner where a set of statements about the fit of the solution were gathered. The GUI was said to feel modern and was deemed to have Saab characteristics. Having the XML as a single source of information on the user side was deemed an advantage. Additionally, the functionality in the test tool was relevant and further development was interesting for the product owner. The final design of the GUI can be seen in Appendix F.

The tests performed using various components were successful. This was decided because of the retrieval of messages with a consistent validation according to the principles by which the validation was designed, and by asserting that the messages transmitted could affect or be displayed in the components that could receive messages. The communications success depended on the messages keeping their integrity from the transmitting part to the receiving part.

Following section contains the questions asked at the start of the project, with answers.

- **How many of all the WISE component can be generated dynamically by code?**
  The drivers could not be generated but could be made to support generic information models such as a generated one. This is due to them being defined in XML-format. The connections between the Data Managers need to be mapped manually.

- **What are the user-interface requirements to consider when generating components?**
  The generation had to be coupled with what was written in XML using names which would show near the generated inputs. There also had to be some form of grouping that

showed if inputs belonged logically. Finally, there had to be error messages shown if the user tried to enter faulty data without override active. Some additional general design concepts were considered as well, such as need for contrast and a coherent design.

- **To what extent can existing functionality in Enterprise Architect be used?**
  The integration to EA was deemed time consuming because of the reliance on third party delivery of documents, which at the was not sufficient to find use cases and common structures of documents. Furthermore, the modelled IRSes in EA are currently only fit for human use and was missing information needed to be a viable alternative. However, EA has the ability to export XML documents from UML objects, thus the parsing can be extended to support the EA format of XML without redesigning significant parts of the existing solution.

- **How does one design the use of component generation to be as need-fulfilling for Saab and as performant as possible while taking into account the limited time frame for the task?**
  With the help of using simpler-to-implement techniques and constraining the delivered solution to include the most important functionality to prove the concept works. It's helpful to make the solution extendable to be integrated into an environment without significant change.

- **How is the balance between dynamic coding and hard coding made based on the time frame?**
  By combining dynamic and non-dynamic coding. First using non-dynamic coding to implement a prototype and afterwards inserting dynamic behavior where necessary. The dynamic behavior needed was the behavior responsible for enabling different protocols to be specified and used with the help of the XML specification.

- **What specification is required to generate a functional and dynamic GUI?**
  To categorize the protocols and sentences, a string for the protocol name and the sentence name was introduced. To generate the inputs, the field type and name with relevant text was introduced. To group the inputs, group tags were introduced. Inputs categorized as dropdown menus needed the different options presented to the user.

- **How does one show the user real time high frequency data in a comprehensive manner?**
  By creating a compact layout with summaries of messages that can be expanded. Thus by creating layers to filter information that is not interesting at first glance and creating an overview before diving deeper. The chat-like layout of the first few designs shown in B were not suitable for high-frequency data. Graphs and data visualization is discussed in conclusions. Graphs could be useful when showing an overview.

- **What other protocols may be relevant to extend for?**
  Any field based protocol with application relevant to the owner of the product. The implementation shows implementation of JSON and NMEA. However, if the solution were to be used for, as an example, testing IOT components, MQTT could be implemented. If there would be a desire to test for example ground vehicles, CAN bus could be extended for.

- **What data is interesting for the user?**
  In this scenario the message values along with validation for the messages. The message header also had relevance.

- **What are the relevant capabilities of WISE?**
  Noteworthy capabilities are:
  Handling of transactions, including rollback in the data managers.
  Driver with the ability to hook into data manager changes.
  Connections with designer tool allowing for mapping through a GUI.
  WISE Connectivity's ability to connect interfaces without having to manually program transmission and retrieval.

## 6.1 Workflow of Solution

The solution as of now is made to be hosted locally and not reachable outside the network. Therefore the assumption is made that setup of the program is done on a machine and thereafter use is repeated on that machine. Therefore is it assumed that most use cases do not involve installation of the program. Furthermore, the GUI is run in a terminal with "npm start" and the WISE components in Visual Studio 2017 or any substitutes for those programs. It is thus up to the developers competence to install the software according to the existing documentation. The resulting start point of the workflow is an installed and run-able program with a dummy XML file baked into the project. If retrieval is to be done from a SE, meaning a simulator, alterations have to be done to the connectivity file by mapping connections in WISE Designer. The following step is to go to "Settings" in the launched GUI to choose source IP, transmit port and multicast-address. To implement or update the list of specified protocols, changes are made in an XML file in accordance with a reference table found in Appendix E and additional documentation containing examples in the repository for the project. The GUI is then reloaded by using the browsers refresh button or F5. This results in an updated GUI with fields for the inputs specified in the XML. The GUI is thereafter ready to transmit. To receive, the "play" button has to be clicked once.

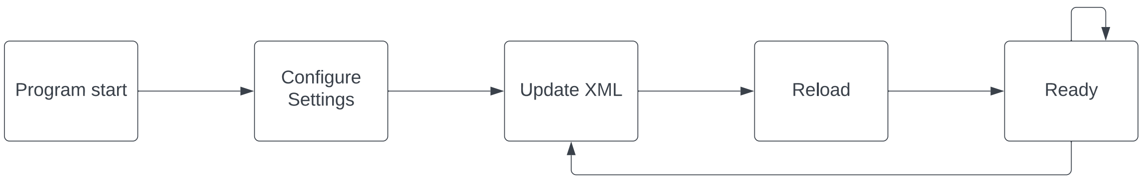


**Figure 6.1:** Chain of events for updating XML.

## 6.2 Result of Experiments

The communication between GUI's and between GUI and the previous test tool was successful from an early stage. This because NMEA-sentences could be sent back and forth. The GUI could correctly classify the messages as valid or invalid. The validation followed the rules set in the XML-file for both structure and value for NMEA and JSON. The result of the test with the INS was retrieval of the full messages with a correct validation for both structure and value-ranges. Any other formats that could be sent by the INS-sensor was received but deemed invalid as expected, because they were not specified in the XML.

As for the MUI component tests, a significant performance degradation was observed. For 100 rendered ListEntry objects, which is the dividers representing messages in the GUI, a debilitating amount of input-latency was apparent when expanding the message dividers. The latency from perceived click to perceived complete expansion was approximated to be at least 500 milliseconds. Whereas the components without MUI accordingly showed no apparent input-latency. The MUI-free versions latency is thus approximated to be below 200 milliseconds. Worth mentioning however, was that useMemo was not used in the test.

The briefly mentioned 100 Hz stress test using the previous test tool resulted in all messages being received, all while the GUI remained stable and responsive.

### 6.2.1 Test tool comparison

| Category | CJEX-sim | Previous tool |
|---|---|---|
| 1 | 10(11) | 10(11) |
| 2 | 7 | 4(7) |
| 3 | 4 | 4 |

**Table 6.1:** Result of test tool comparison where the parenthesis contain a count of even partially or substandard functionality.

# Chapter 7
# Discussion

The starting point of this project was inexperience of web development. We had some experience working with Visual studio and C# external to our time at LTH. This is relevant when discussing the time span of the project. The inexperience of web development meant some extra time was spent reading and trying React hooks and JSX objects. While the experience with C# and its tools meant that those parts could be quickly started.

## 7.1 Process

Because of the small size of the development group the need for a prescriptive or strict process was not apparent. A task list was deemed sufficient because the amount of parallel work in the project never exceeded an unmanageable amount. Reflecting on the use of lists brings to light some similarities to a Kanban board because of the dynamic appending to the list when a need for a functionality was apparent. But also because of the flow of having a list of tasks to do and crossing them off one or two at a time. Informally there were roles emulating a product owner and a scrum master. The product owner was the person with the knowledge of the business need and the scrum master was the one with the notebook containing the task lists.

Reasoning around the process makes it clear that a mix of principles were used that felt natural and easy to implement at the time. The complexity of figuring out the design and eliciting parallel to structuring a process meant the process was not set up at first, but was iterated upon during the project. If this work were to be continued or extended with more developers, the task list would naturally progress to a Kanban board to ensure a more collaborative space. The digitalisation of the task list would also offer better traceability and documentation at the cost of some initial setup- and learning effort.

The iterative implementation of functionality by first introducing the least complex functionality first was convenient to reduce the complexity of the brainstorming, and allowed for design of more complex parts being done when the practical information was maximized. Figuring out the least complex functionality also led to dividing the task into smaller parts and starting with a manageable part.

The use of whiteboards which always were in the vicinity was deemed useful because the information and diagrams from focused brainstorm sessions were easy to find and see. Minimal context switching was needed to access the information, just a turn of the head. This allowed our code decisions to be more frequently influenced by the design decisions that were agreed on.

## 7.2 UX/UI

The GUI follows the Saab design guidelines (Saab AB, 2022). The design guidelines are used for inspiration and to give the GUI a design that is recognisable as a Saab product. However, the design guidelines are not meant to severely restrict the design and do not cover all aspects of the design. Sometimes complementary design choices had to be done, such as what happens when hovering over certain parts. Designs of effects existed but being cautious of overuse of those effects led to some diversification and additional effects.

## 7.3 Technical Choices

A range of technical choices were made in the project, some almost unconsciously.

### 7.3.1 API

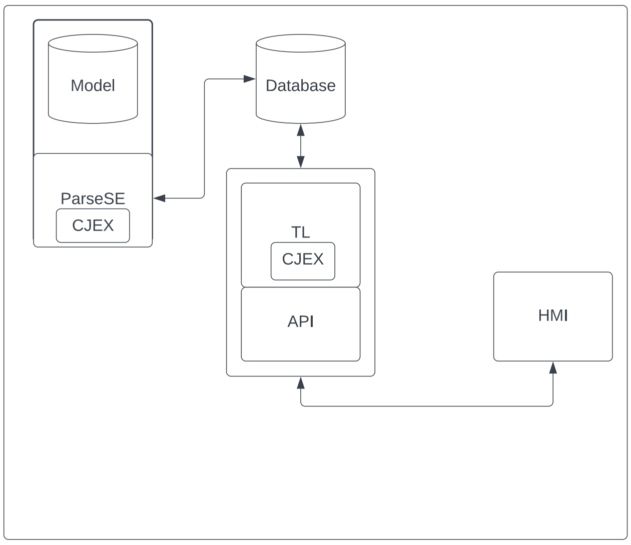
REST was one of those technical decision that was made early on in the project. REST was relatively straightforward to implement and our prior experience with it also helped. Despite this, other solutions than REST were investigated at later on such as WebSocket and long-polling. However, REST was deemed to be sufficient and thus the time required to overhaul it was instead spent on developing other features that contributed more towards the goal of this thesis.

### 7.3.2 GUI

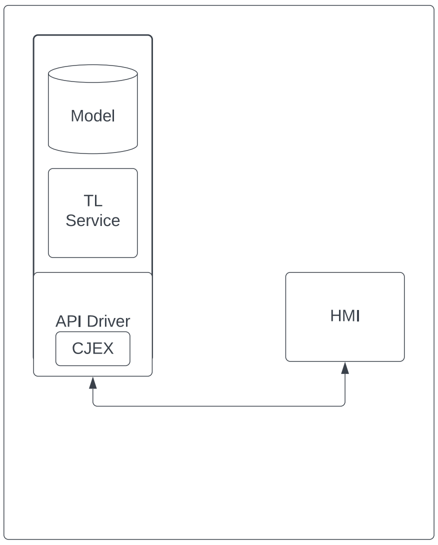
It was decided early on to save the values of inputs, options and so forth in the local storage of the browser. This was mainly a QOL improvement, to prevent the user having to type in the same values multiple times when for example swapping between the different protocols or pages. Local storage was also used to log the messages being fetched from the REST API. Although, this lead to a number of issues, mainly in regards to security and performance. After a deliberation with Marcus and investigating this further it turns out that any website could access the local storage. This was a big issue that had to be addressed. Furthermore by logging the messages sent from the REST API in the local storage it impacted the performance heavily when rendering the information. Local storage is also limited to 5MB which would most likely be exceeded. Local storage was changed for session storage in regards to saving inputs, which is cleared after the session has ended. Moreover, the logging of messages was moved to the REST API that implemented filtering and pages. Instead of sending all the messages for each API Request, a limited number of the latest received messages would be sent.

### 7.3.3 WISE

There were a number of technical choices made related to WISE, all having their drawbacks and advantages. To begin with, it was decided not to separate the API service from WISE as originally was intended, see figure 1.1. This was mainly due to reducing the number of components and complexity. A separate database that is not part of WISE would have to be constructed such as a SQLite database in order to store the data from WISE so that a separate API service could access it. This would also have an impact on the delay, since there would be one more step until the the information can be accessed. Although it would allow saving the session data for later use and analysis which would be deemed beneficial. Another decision that was considered was in regards to combining the API and TL Driver or at least changing the TL Driver to a WISE service attached to the API Data Manager. This decision was also not made mainly due to the decision being deliberated too late in the project stage and it would not be worth the time needed to refactor the drivers. There is also a benefit with splitting the drivers up since it makes it possible for the TL Driver to be located in another network than the API Driver. Similar decisions were made with the ParseSE Driver, to combine it with the other drivers. However, this was also deemed unnecessary for the task.



**(a)** The functionality of the API and TL Drivers are moved outside WISE, while the ParseSE remains a WISE Driver. All of the messages are logged in a database.



**(b)** API Driver combined with the same functionality of the TL driver as a service.

**Figure 7.1:** Two alternative solutions.

There was also a discussion held with the product owner towards the end of this thesis in regards to the purpose of WISE in this solution. There is not necessarily a need for WISE in this particular case, although it has a few benefits as is described above. There is a possibility to partially transition away from WISE towards a monolithic approach as can be seen in figure 7.1b where only ParseSE remains a WISE component.

## 7.4 Frameworks, tools and Libraries

This section lists the most important frameworks and libraries utilized in this project.

### 7.4.1 React

React with JSX allowed for fluid programming without thinking too much about dividing HTML and JS. This allowed for easier implementation of logic where HTML objects had to be returned conditionally. JSX allows for more variations, which means less restraints, in programming which also increased ease of learning.

In this thesis, React was new to us, which meant that the code in this project can be optimized. Something that was missed were the assigning of keys in ternary operators. Adding a key to the HTML objects is necessary to hint to the DOM that the components are different. We did not use React to its full potential, which is advised if this product is to be further developed. However, functional components and state was used which is sustainable.

### 7.4.2 MUI

MUI provides well-rounded components especially fitting when there is a need for more complex logic. Having more complex logic behind components generally means the component takes longer time to create, which in turn means the potential reward of using MUI is increased. Furthermore, using MUI means flexibility of the UI is increased in later stages of development due to the simple theme changes of MUI components compared to CSS. However, since MUI components carry a certain performance weight imposed by its implementation which can not be changed by any developer, certain applications involving a frequent or high amount of rendering can be slowed down. MUI solves this by exposing hooks which can be utilized to implement a lightweight version of a component. However, in this project with limited knowledge of React, it was deemed riskier to use these hooks than to implement components ourselves. The performance degradation with MUI components can most likely be mitigated by using useMemo on the individual expandable accordion components. Correct use, which means assigning a unique key to components using UseMemo, can most likely result in smooth performance. Combined with optimizations done with a production build, MUI could be viable for future use.

### 7.4.3 NMEA

Some libraries existed for implementing NMEA messages. However, they were either poorly documented or too complex for our use case. When seeing the risk of time waste studying existing complex libraries or the risk of using libraries not optimized for our use case, it was decided that NMEA was to be implemented from the ground up. Having a fairly straightforward implementation, the need for an existing implementation was not great enough.

### 7.4.4 XML

Some alternatives exist for XML in the use of serialization. The ones compared were Protobuf and JSON. The advantages of XML over protobuf is beginner-friendliness. Protobuf has a more complex syntax which were different from previous experience with markup languages such as HTML. Another option for serialization was JSON. However, XML was deemed to have more human-readable tags and has the option to add attributes to tags directly. Furthermore, Enterprise Architect has the ability to export XML from models, which means that for a coupling with EA, XML was required lest another conversion to JSON would need to be added to integrate into the solution. Additionally, there is the benefit of being able to use an XSD, which also is integrated in .NET that is able to generate XML-reading-classes from an XSD file. All of this coupled with XML being deemed sufficient led to the choice.

### 7.4.5 react-json-viewer

react-json-viewer is a component in React which can display JSON-formatted messages with a tree view containing expandable nodes. This component saved time and decreased risk implementing JSON messages in a readable format.

### 7.4.6 ESLint

ESLint(OpenJS Foundation, 2022) is a tool used to flag errors in JS code. Usage of ESLint is recommended to keep a clean environment. Visualising errors in the terminal and keeping an overview is motivating to fixing those errors. When the code is completely free of detected errors there is a green text indicating this. This probably proves as a reward for the brain (Schiffer et al., 2014) which in turn encourages the behaviour of creating cleaner code.

## 7.5 Tests

The MUI component test with MUI Accordion and custom build components did not use further optimization techniques. However, using further optimization techniques could perhaps result in acceptable input-latency, if coupled with fewer objects rendered in the list. Although, this would impose greater limits on the solution and was deemed to be disadvantageous. The cause for the greater performance with the custom build components was the lightweight approach. MUI Accordion is made to be a general component and thus contains more content to be implementable in many scenarios. Additionally, MUI Accordion contained an animation when expanding which gives the GUI a more living appearance. However, with our use case, a lightweight approach was considered more advantageous due to the commercial potential use of the solution. In commercial scenarios, productivity is prioritized and therefore the faster expansion is more aligned with the prioritization. Showing more components on screen at once is also better for giving the user a better overview of data when needed.

## 7.6 Comparison

Upon looking for other programs similar to this solution, no other program with the same characteristics were found. Several tools with the ability to view data and do some kind of health check of network was found. But none which could send specified protocol messages at the same time, which makes the solution in this thesis unique. Furthermore, a disproportionate need of an in-house tool because of security and confidentiality concerns exist for Saab. These concerns makes an in-house tool essential for the types of operations needed.

### 7.6.1 SailSoft's NMEA/AIS Simulator

The closest competitor found was SailSoft's NMEA/AIS Simulator. The tool has routes which is a variation of the future development of scenarios for the developed tool in this thesis. The additional use of this thesis solution would be the dynamic specification, with the ability to specify even more custom messages and extend for more behavior. Sailsoft's solution is missing the part with validation that is present in this solution. It is also strictly limited to the NMEA 0183 protocol, while the solution presented in this thesis is more of a general approach that is able to support a wide array of different field-based protocols such as JSON. This comparison credits this thesis' solution with some uniqueness.

### 7.6.2 Wireshark

If one were to compare our solution with Wireshark (Wireshark, 2022), which is a packet sniffer program to investigate network protocols and communication, one would see that Wireshark is a popular program with broad investigative functionality. However, all of that functionality is used with the help of a certain syntax for filtering and the user gets a wide range of data shown. The solution shown in this thesis is a more specific tool that filters out relevant information with the help of a GUI with buttons and the like. Firstly, this helps lower the competence needed to use the program and secondly, because of the specific nature, the relevant information is gathered with less clicks or steps and therefore faster. This speeds up the process. Another attribute of the solution is that it is in-house to Saab Kockum's, which means that any future development can be tailored to Saab's needs, which can further improve the speed of the testing and integration process. Controlling the software development can allow easier addition of triggers when certain values are gathered. Take for example the future development of scenarios mentioned later in future development in this report, where one would have to create a new program since Wireshark lacks a robust API to use the data or add triggers to it. This means any attempt to automate testing procedures would not work better with Wireshark.

Wireshark also lacks the ability to send any information or to simulate a sensor. Wireshark is intended to look at data, not create and send it. This is where this thesis' solution has the upper hand when considering Saab Kockum's needs, since it is made with those needs in mind.

### 7.6.3 Previous Test Tool

The previous test tool has the flaws relating to the process in which it is created. Those flaws such as repeated development led to longer time to change or implement sensor testing. Furthermore, the development of the test tool was further from the users in terms of amount of parties involved and any distance from the user can result in increased steps the user-needs need to be communicated, which can increase risk of miscommunication or missed-communication.

The test tool comparison gives some estimation of a comparison. However, one has to consider that the list of functionality that is compared can be flawed by containing more functionality that is contained in one tool compared to the other. This bias was mitigated by looking for all existing functionality and include it in the list. Meaning a superset of the union between each test tools functionality. There is still possibility for bias through the assessment of whether or not the functionality exists. Because it's the developers of one test tool deeming if functionality exists. Having more knowledge around one test tool could cause some misses in the other tool. This was kept in mind during the comparison process and discussion were had among developers to maximize the fairness in the comparison.

The result of the comparison seems to indicate that the test tools are not significantly different when it comes to amount of pure functionality. There were some functionality where one performed better and vice versa. This is seen as positive for the new web-based test tool as it holds more potential going forward. This is reasoned forth by the fact that the new test tool is built on a foundation which allows for more re-use and flexibility, and the use of more modern frameworks. The new tool is designed to be dynamic, which means that any re-implementation can be done at a fraction of the time using XML. The XML is an advantage because it requires less competence to use than to implement an entire application from scratch. The XML acts as a source for the entire program's dynamic behaviour, which improves the traceability since any change made to the XML will have an immediate synchronizing effect. If a coupling to EA is implemented in future versions, the machine-readable parts of the IRS for the new test tool would be guaranteed to be followed by the test tool, whereas the previous test tool cannot be guaranteed to follow the latest IRS. Also the new test tool is developed in-house, meaning the lead time to obtain and use said test tool is assumed to be significantly shorter.

## 7.7 Source Evaluation

When searching for information on how to solve programmatic problems there is a focus on gathering many examples with speed to broaden the problem-solving horizon. Therefore google is frequently used for hastily finding many sources. The problem with google is that anything can be shown, including old or incorrect information. Especially with ever evolving products and frameworks. However, because of the programmatic nature of the problems, they can be evaluated quickly by checking if the program compiles and analyzing fitness by reflecting on the soundness of the solution in line with personal knowledge. Therefore, finding incorrect information is usually quickly detected as incorrect resulting in the cost being low for finding faulty information. Several sources are visited to compare solutions and which solution is most up to date. Using Google to search has the benefit that the most fitting answer most often is at the top. Regarding React, the sources mentioning hooks were preferred instead of those containing ComponentDidMount. This because ComponentDidMount is deprecated and hooks are more recent. Moreover, when researching frameworks, the documentation stemming from the developers of the framework is preferred as long as it is up to date.

Some sources can be validated by us with the use of other sources that back up claims. An example of a source that has been validated is (Rietman, 2008). The reference is a walk through of how to calculate NMEA checksums. The principle set forth is widely used to calculate checksums for other protocols. Calculating checksum in line with this method resulted in the NMEA-sentences being validated as correct when sent from an IMO-certified INS outputting NMEA-sentences. The INS is identical to the ones used in real scenarios. Additionally, the INS outputs the same message as a LOG-sensor would according to sources at Saab. Others can be validated by their source of origin, ones which are regarded as especially trustworthy includes sources from governments, companies or well known journals such as IEEE. For example (IBM Cloud Education, 2021), IBM is a well established actor with more than a century of experience and operation within the tech domain with many inventions and patents. Hence it could be considered a reliable source. In instances where the origin may not be verified and source may be suspect, other sources can be used to back up the claims of the unverified one if found. If not, the unverified source would be disregarded.

## 7.8 Ethical aspects

This work has been done during weeks containing no more than 46 hours, and most often under 40 hours. A work/life balance has been encouraged and no personal conflicts have appeared during this work. Collaboration has been encouraged with compromises being made when developers have had different opinions. Because of the software living in office computers, weekend-work has for the most part not been possible. Therefore allowing for recovery during weekends. The writers are confident that the work done has been completed under ethically sound work conditions.

The purpose of this solution is to increase the productivity of developers developing or testing communication across systems. As a general and dynamic toolset, there is no shortage of use cases.

With such a general tool, one can wonder where the responsibility is of its use. Is the engineer responsible for how the product is used? If a football is constructed, is the constructor responsible for the football not being used to harm anyone? An example would be if a player was struck by the ball in the head. Is the maker at fault or the user? Is it the intention of the maker or the result of the makers actions? The same goes for the user, is it the intention of the user or the result of the users actions that dictate morality? This dilemma is called dual-use (Miller and Selgelid, 2007). Perhaps everyone carries responsibility to at least reflect on ethical aspects.

The solution is not exclusive of other industries. However, working with Saab, a company in the defence sector, results in the discussion naturally progressing towards aiding technical military development. Aiding in military development might not intuitively be considered ethical. However, military development has had many benefits for civil use. As an example, GPS originated from the US military which is widely used today on the planet to navigate (Federal Aviation Administration, 2022). Another example is the positive development the military has had on cryptography, which saw a paradigm shift during world war two that also contributed to the invention of modern computers (Tresorit team, 2022). Cryptography is crucial in today's society in protecting sensitive information, such as bank details, from malicious third parties.

Furthermore, halting production of military material is not effective due to it opening up opportunities for other companies in the same market. Another company can simply replace the production down the line if systemic and regulatory inhibitions are missing. What a company can do to have a healthy ethics, is align itself and comply with regulations set up by government branches influenced by democracy. Today, Saab is complying with regulations set up by the Inspectorate of Strategic Products (ISP) (The Board of Directors of Saab Aktiebolag, 2021). Which in turn is influenced by the ruling party, decided by a democratic process. Therefore the responsibility of limiting export is passed to ISP. The subject of national defence is more easily justified in a democratic society where accepted values of the people are defended and not repressed. The question thus becomes if democracy is worth defending, which is supported with the concept that ethics can be the art of rationalising what is best for a majority of people. Since democracy is for the majority, democracy is conceptually aligned with what makes ethics. As opposed to communism, which is also for the masses, democracy has been proven to be rather stable.

The solution can possibly be used to gain more funding through sales and to deliver more. Therefore the solution can have an impact on keeping the company afloat. Keeping Saab afloat can result in the 18000 people employed keeping economic stability and further developing technology. Employments are good for the economy and good for keeping peace. (The United States Institute of Peace, 2022).

Finally, transparency is important when working with confidential material. Transparency from authors of work based on confidential material can often be lacking in information required to evaluate the article. Which creates a need to trust the authors words with subpar evidence. It is in such moments that the authors need to hold themselves accountable to report accurately and to be honest. The possibility to make up information and when asked to prove it, claim confidentiality, is not ethically sound as it can fool people into believing faulty information, which then can jeopardize their performance at work or generally in life. Therefore, authors may to be held to a certain standard to ensure minimal inaccuracy can occur. The reader is encouraged to further reflect on this subject.

# Chapter 8
# Conclusions

To conclude this thesis, a solution was delivered with functioning communication according to the specifications of the product owner. The product owner saw potential, value and real use cases for the product. Exception was the exclusion of Enterprise Architect, for which the program instead was designed to handle with minimal change. The solution delivered has potential to be a unique program aligned with Saab Kockum's needs.

A comparison was made to show that the difference in functionality between the two is not significant. Although, the foundation of the new test tool is considered to be more modern, reusable and dynamic than the previous test tool. The solution is advised to be further developed to create more value. The solution can be used in its current state for viewing communication with validation and filtering functionality.

This work brings light to what data and metadata is required to generate a functional GUI, and proposed techniques and formats to possibly be used. Interesting to further investigate would be if Enterprise Architect can sufficiently provide this metadata in a worthwhile manner and what advantages and drawbacks would exist in such enterprise workflows. Furthermore, modelling standards for Enterprise Architect needs to be investigated.

## 8.1 Future Development

The preliminary idea for the project was to allow generation of software components from Enterprise Architect. However, the increased complexity and inconvenience of working with EA, and therefore third party consultants which held the EA competence, led to the feature to be left out for this project. This means the option to extend the software to handle XML from Enterprise Architect instead of standalone XML exist and therefore it is a realistic possibility.

### 8.1.1 XML

Currently XSD is not used to verify the structure of the XML file, instead some manually error checks were implemented. This is not ideal since not all of the possible cases may be covered and it is more of a flexible approach that relies on the user making no mistakes. It is strongly suggested to instead implement XSD.

### 8.1.2 WISE

The main limitation with WISE is that it is session based. To overcome this a database such as a SQLite or similar could be implemented as proposed in section 7.1a.

### 8.1.3 Security

The GUI could implement user authentication to limit the users access rights to specific IRSs. This could be done with a login which also can be used to save information related to the user.

### 8.1.4 Scenarios

When sending messages to the GUI, there could be a defined scenario to respond with certain information, such as an acknowledgement. This concept allows for automated test-plans directly in the GUI. This functionality could be extended to send information back and forth to simulate certain control systems.

### 8.1.5 Data Visualization

#### Display trends

Data Visualization could show trends of messages and graphically visualize information sent. Because of the fact that the system logs information, trends can be shown. More data points such as received message frequency could be measured and reported through the GUI. Latency could be measured by comparing the WISE time stamp with the time the message is rendered on the GUI. Graphs of various types could be shown and/or exported to summarize certain behaviour of the messages.

#### More filtering

With more data comes the need for more filtering functions. Filtering functions can be extended to filter for protocol type and to filter for specifically invalid messages. It could also be extended to sort the messages in different chronological orders. More filters that could be implemented is filtering for specific error messages.

### 8.1.6 Testing

The Azure DevOps pipelines could contain more automated tests to further integrate with Saab's practice of continuous integration. Testing gives confidence to automate. More specifically, implement automated tests for the GUI and the communication. Currently only the CJEX library is partly tested.

### 8.1.7 Communication

#### More complex protocols

Currently only field-based protocols are supported that can be sent either via serial port or UDP multicast. This could be extended to also support more advanced protocols with more complex chain of sequences, such as acknowledgement.

#### Allow to listen/send on multiple ports

There was a late desire from the product owner to be able to send and/or listen to multiple different systems or sensors at the same time. Those wishes were not able to be fulfilled due to time constraints. This could possible be achieved by replacing the TL driver with WISE services in the API Driver.

#### REST API

REST-API works great for many cases, however it has a major flaw in regards to real-time communication since there is a need to continuously poll. This means that a request will be sent regardless if new information is available or not from the back-end service. This will have an impact on the performance for the browser, server and cause unnecessary strain on the network. Another downside to polling is the delay between the requests. For example if new information is available on the server and client recently polled, client will have to wait for the information until next polling request which causes a delay. This can be mitigated by increasing the polling frequency, although this will further degrade the performance.

One way to combat this issue is to switch from a REST API to WebSocket which allows a two way interactive communication session to be opened between a client's browser and a server (Mozilla Corporation, 2022) (Herwig et al., 2015). The WebSocket API makes it possible for the client to send and receive event-driven responses without having to continuously poll the server. In the article (Herwig et al., 2015) the authors compare the energy consumption for mobile devices between REST and WebSocket. They found that WebSocket is better suited for real-time communication since the connection to the server is always open and active which improves performance. They also found that the latency was improved by not having to send the HTTP-Headers for each request. However, the major challenge with WebSocket is that it is mainly a protocol for communication and not for defining the structure itself (jfarcand, 2012).

Another way is to use a variant of polling called long-polling (Javascript.info, 2021) (Herwig et al., 2015). It is much simpler to implement than WebSocket and shares some of its benefits such as being able to receive messages without delays. The way it works is that the client sends a request to the server, which does not close the connection until it has new information available. Once it does, the server responds immediately to the request and the client makes a new request to the server. However, in comparison to WebSocket it still has the drawback of having to establish a new connection after each request is responded to which has a negative affect on the performance compared to the open and active connection of a WebSocket (Herwig et al., 2015).

The best of both worlds is to combine REST over a WebSocket. By implementing an application level protocol on top of the WebSocket. This is achievable with a library such as SwaggerSocket (jfarcand, 2012) (jfarcand et al., 2019). In our case the WebSocket may be used for the real-time communication while the REST for initialization of the web GUI and other related settings.

### 8.1.8 UX

#### Embedded IRS

Connected IRS could be generated and embedded into the web GUI when generating said GUI. This could be done by linking to a .PDF on a server or locally. The GUI could then have a switch between different embedded IRSs and get the belonging collection of protocol messages.

#### Custom behaviour

The react application could contain custom coded behaviour for more complex NMEA messages. For example the DBT message contains water depth in different units. A custom behaviour in this case could be automatically converting between the different units when one of the fields is filled in the web GUI. Alternatively, only the SI units could be displayed in the web GUI and the rest could be converted in the back-end away from the user.

#### Combine protocols

The solution could contain behaviour to clump together similar messages into one interface. An example is the HDT and HDM -NMEA messages which differ by the value in one field. Switching that field could automatically switch protocol from one to the other.

#### Drag and drop of XML definition file

The GUI could contain drag and drop functionality of XML files to send file path to the back-end instead of having to assign a file path programmatically.

#### Tutorial

A tutorial could be implemented at the first visit to the site which explains functionality and any required information.

### 8.1.9 UI

#### Better accessibility

The GUI could contain keyboard-focus-able elements and tags that can be read by screen-writers. Material UI has support for this type of behaviour with Aria-labels.

#### Experimental design

Experiment with more modern appearance by deviating from Saab Design Guidelines.

#### Different themes

Theme switching can be implemented with the choice of light and dark mode. Currently only dark mode is implemented.

#### Expanded MUI usage

The components can be ported to MUI or extended in functionality. The options can be implemented to color code components based on errors. MUI can allow easy popups to be implemented which can be used to give the user information. Popups can also be used at the first visit to the site to explain how the site works. MUI can also be used to further align with Saab's goal to use Google's Material Design. MUI comes with both extra functionality but also some limitations. The main limitation found in this project included poor optimization when rendering many components. Although this could be further investigated using different optimization techniques such as UseMemo() and by evaluating a production build. Production builds in React are greatly optimized because of bundling of software resources to allow for use of resources only when relevant.

# Chapter 9

# References

Alpert, S., Abramov, D., and Florence, R. (2018). React today and tomorrow and 90% cleaner react with hooks. `https://youtu.be/dpw9EHDh2bM?t=411`.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). Manifesto for Agile Software Development. `https://agilemanifesto.org/iso/en/manifesto.html`.

Federal Aviation Administration (2022). Satellite Navigation - Global Positioning System (GPS). `https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/gps`.

Hannay, J. E., Dybå, T., Arisholm, E., and Sjøberg, D. I. (2009). The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, 51(7):1110–1122. Special Section: Software Engineering for Secure Systems.

Herwig, V., Fischer, R., and Braun, P. (2015). Assessment of REST and WebSocket in regards to their energy consumption for mobile applications. *International Conference on Intelligent Data Acquisition and Advanced Computing Systems*, 1:342–347. doi: 10.1109/IDAACS.2015.7340755.

IBM Cloud Education (2021). REST APIs. `https://www.ibm.com/cloud/learn/rest-apis`.

Javascript.info (2021). Long polling. `https://javascript.info/long-polling`.

jfarcand (2012). Introducing SwaggerSocket: A REST over WebSocket Protocol. `https://blog.wordnik.com/introducing-swaggersocket-a-rest-over-websocket-protocol`.

jfarcand, elaktio, fehguy, webron, JanxSpirit, and starlightknight (2019). SwaggerSocket: A REST over WebSocket Protocol. https://github.com/swagger-api/swagger-socket.

Khare, R. and Rifkin, A. (1997). Xml: a door to automated web applications. *IEEE Internet Computing*, 1(4):78–87. doi: 10.1109/4236.612222.

Meeks, T. M. (2015). How one project at sandia labs is using sparx enterprise architect to create model-driven requirements and documents.

Meta Open Source (2022). React A JavaScript library for building user interfaces. https://reactjs.org/.

Meta Platforms (2022). Introducing jsx - react. https://reactjs.org/docs/introducing-jsx.html.

Microsoft (2022). Azure DevOps. https://azure.microsoft.com/en-us/services/devops/.

Miller, S. and Selgelid, M. J. (2007). Ethical and Philosophical Consideration of the Dual-use Dilemma in the Biological Sciences. *Science and Engineering Ethics*. doi: 10.1007/s11948-007-9043-4.

Mozilla Corporation (2022). The WebSocket API(WebSockets). https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.

MUI (2022). MUI: The React component library you always wanted. https://mui.com/.

OpenJS Foundation (2022). Eslint. https://eslint.org/.

Raymond, E. S. (2018). NMEA Revealed. https://gpsd.gitlab.io/gpsd/NMEA.html#_dbt_depth_below_transducer.

Rietman, G. (2008). How to calculate the NMEA checksum. https://rietman.wordpress.com/2008/09/25/how-to-calculate-the-nmea-checksum/.

Saab AB (2022). Digital Design. https://brand.saab.com/brandcenter/en/saab/digital-design.

Sailsoft (2014). Sailsoft AIS and NMEA Simulator Software. https://www.sailsoft.nl/.

Schiffer, A.-M., Muller, T., Yeung, N., and Waszak, F. (2014). Reward activates stimulus-specific and task-dependent representations in visual association cortices. *Journal of Neuroscience*, 34(47):15610–15620. doi: 10.1523/JNEUROSCI.1640-14.2014.

Systems, S. (2022). ENTERPRISE ARCHITECT. https://sparxsystems.com/products/ea/index.html.

The Board of Directors of Saab Aktiebolag (2021). Annual General Meeting of Saab AB on 13 April 2021. https://www.saab.com/globalassets/corporate/corporate-governance/annual-general-meeting/2021/en/board-statement-re-proposal-from-sw-peace-and-arbitration-society.pdf.

The United States Institute of Peace (2022). Employment Generation. https://www.usip.org/guiding-principles-stabilization-and-reconstruction-the-web-version/sustainable-economy/employment-g.

Tresorit team (2022). The history of encryption: the roots of modern-day cyber-security. https://tresorit.com/blog/the-history-of-encryption-the-roots-of-modern-day-cyber-security/.

Vuksanovic, I. and Sudarevic, B. (2011). Use of web application frameworks in the development of small applications. *34th International Convention MIRPO*, pages 458–462. https://www.researchgate.net/publication/221412761_Use_of_web_application_frameworks_in_the_development_of_small_applications.

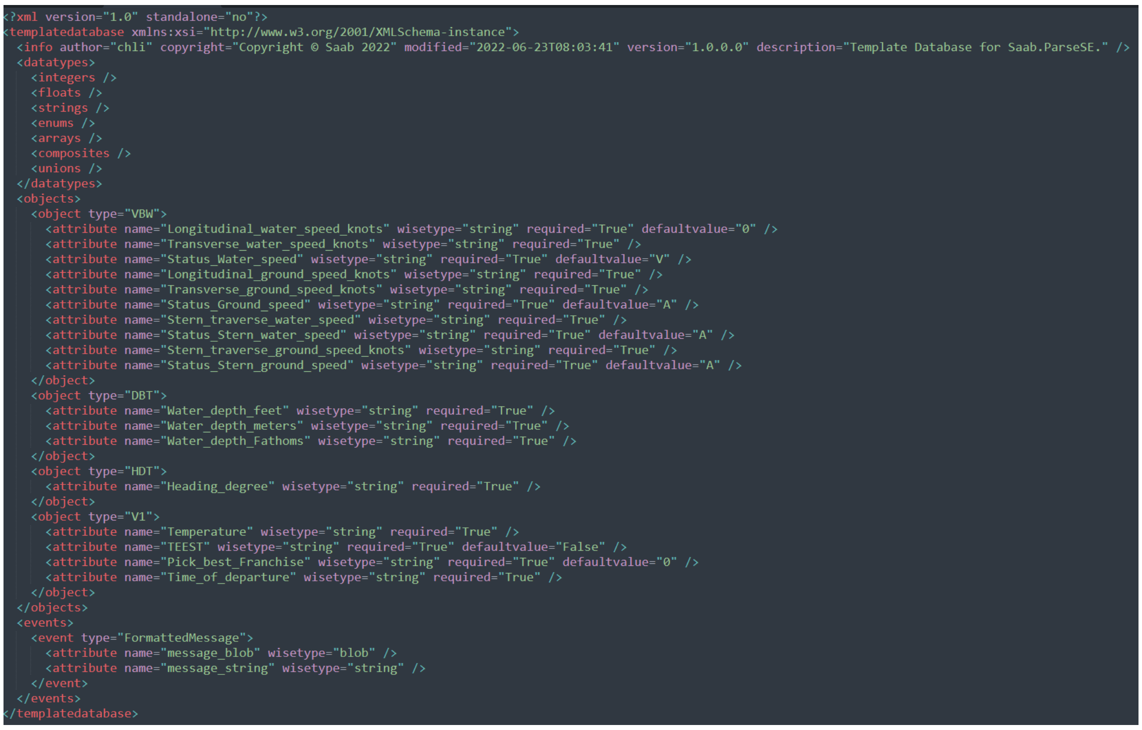
W3 Schools (2022). XML Schema Tutorial. https://www.w3schools.com/xml/schema_intro.asp.

Wireshark (2022). Wireshark. https://www.wireshark.org.

# Appendices

# Appendix A

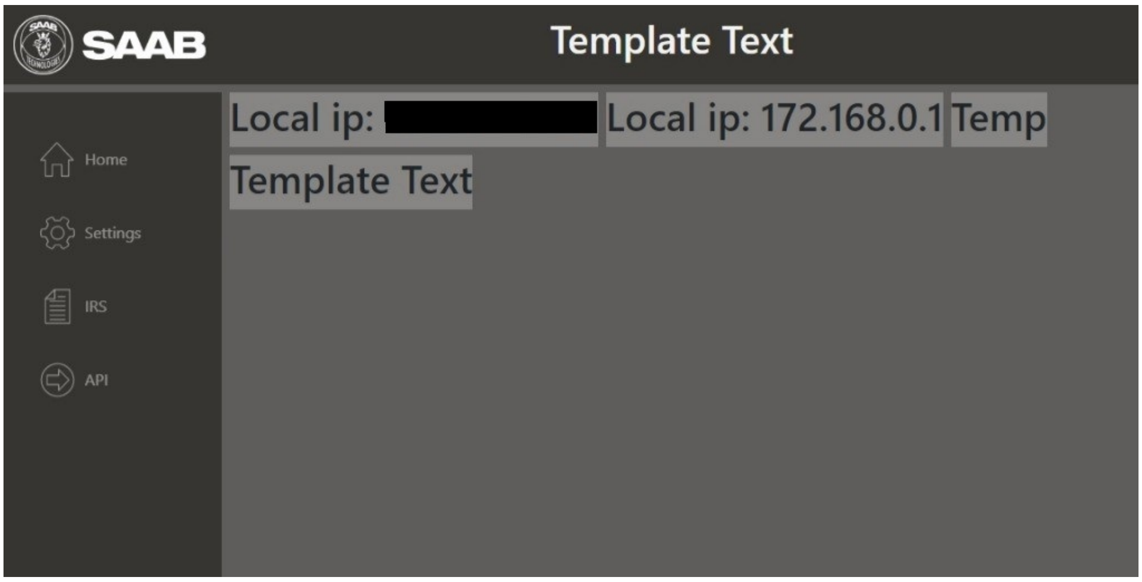
# Information Model Definition



```xml
<?xml version="1.0" standalone="no"?>
<templatedatabase xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <info author="chli" copyright="Copyright © Saab 2022" modified="2022-06-23T08:03:41" version="1.0.0.0" description="Template Database for Saab.ParseSE." />
  <datatypes>
    <integers />
    <floats />
    <strings />
    <enums />
    <arrays />
    <composites />
    <unions />
  </datatypes>
  <objects>
    <object type="VBW">
      <attribute name="Longitudinal_water_speed_knots" wisetype="string" required="True" defaultvalue="0" />
      <attribute name="Transverse_water_speed_knots" wisetype="string" required="True" />
      <attribute name="Status_Water_speed" wisetype="string" required="True" defaultvalue="V" />
      <attribute name="Longitudinal_ground_speed_knots" wisetype="string" required="True" />
      <attribute name="Transverse_ground_speed_knots" wisetype="string" required="True" />
      <attribute name="Status_Ground_speed" wisetype="string" required="True" defaultvalue="A" />
      <attribute name="Stern_traverse_water_speed" wisetype="string" required="True" />
      <attribute name="Status_Stern_water_speed" wisetype="string" required="True" defaultvalue="A" />
      <attribute name="Stern_traverse_ground_speed_knots" wisetype="string" required="True" />
      <attribute name="Status_Stern_ground_speed" wisetype="string" required="True" defaultvalue="A" />
    </object>
    <object type="DBT">
      <attribute name="Water_depth_feet" wisetype="string" required="True" />
      <attribute name="Water_depth_meters" wisetype="string" required="True" />
      <attribute name="Water_depth_Fathoms" wisetype="string" required="True" />
    </object>
    <object type="HDT">
      <attribute name="Heading_degree" wisetype="string" required="True" />
    </object>
    <object type="V1">
      <attribute name="Temperature" wisetype="string" required="True" />
      <attribute name="TEEST" wisetype="string" required="True" defaultvalue="False" />
      <attribute name="Pick_best_Franchise" wisetype="string" required="True" defaultvalue="0" />
      <attribute name="Time_of_departure" wisetype="string" required="True" />
    </object>
  </objects>
  <events>
    <event type="FormattedMessage">
      <attribute name="message_blob" wisetype="blob" />
      <attribute name="message_string" wisetype="string" />
    </event>
  </events>
</templatedatabase>
```

How a Information Model may be defined in XML, in this example a generated one can be seen.

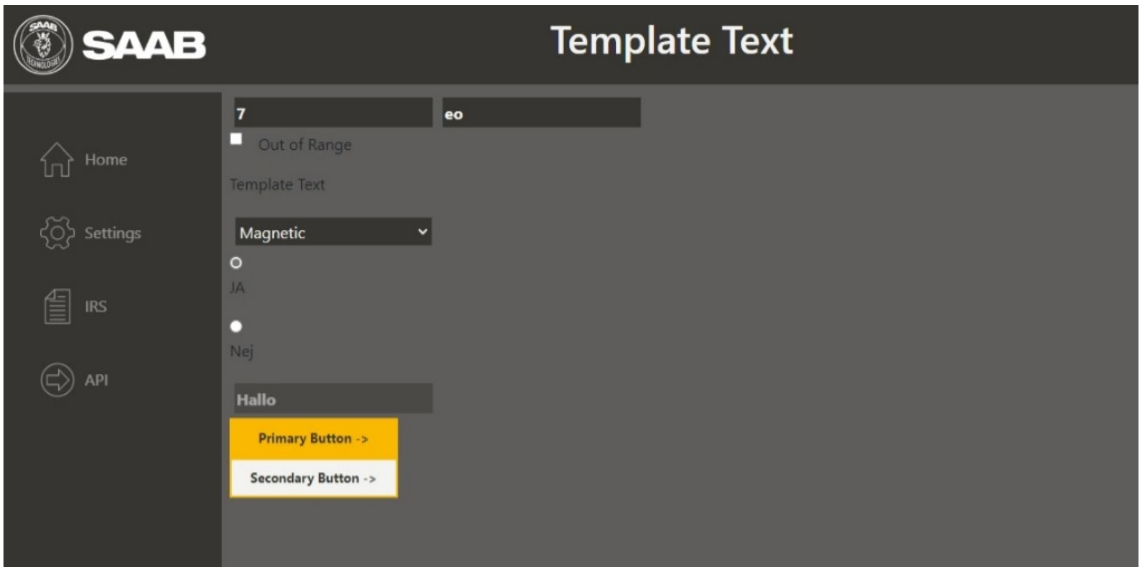
# Appendix B
Iterations of the prototype



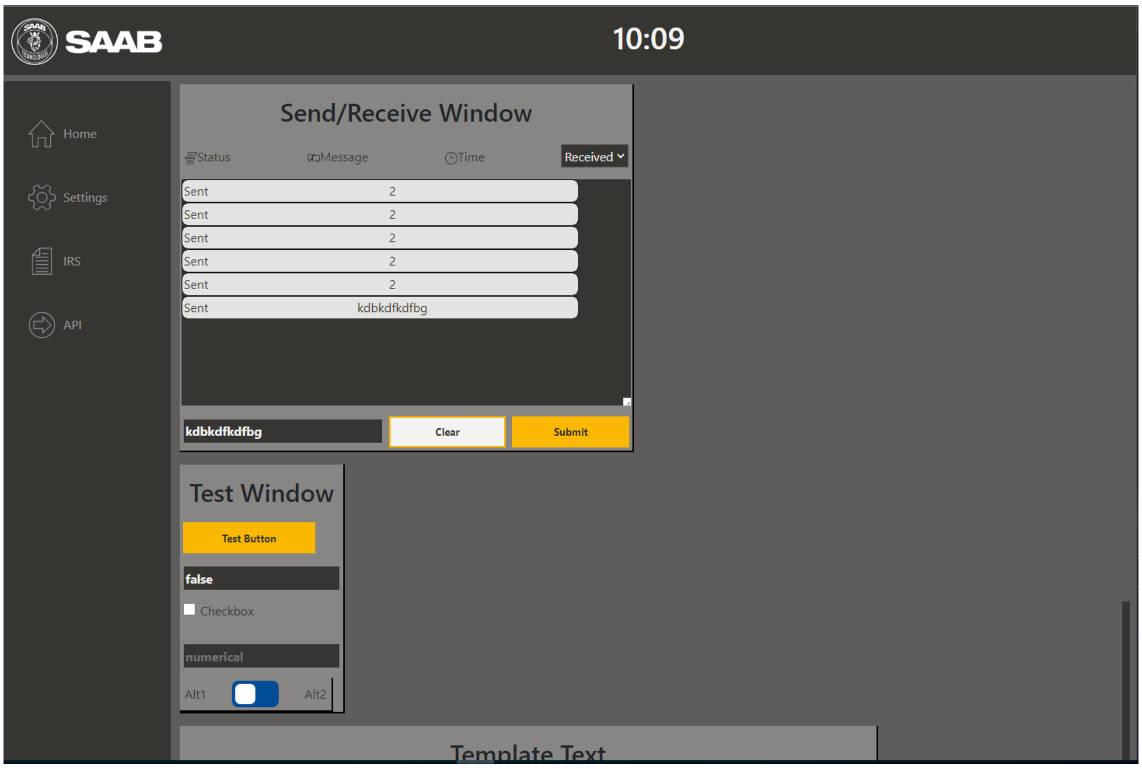
Local ip: [redacted] Local ip: 172.168.0.1 Temp

Template Text

Early layout of the GUI with some API functionality.



Experimenting with different input types.

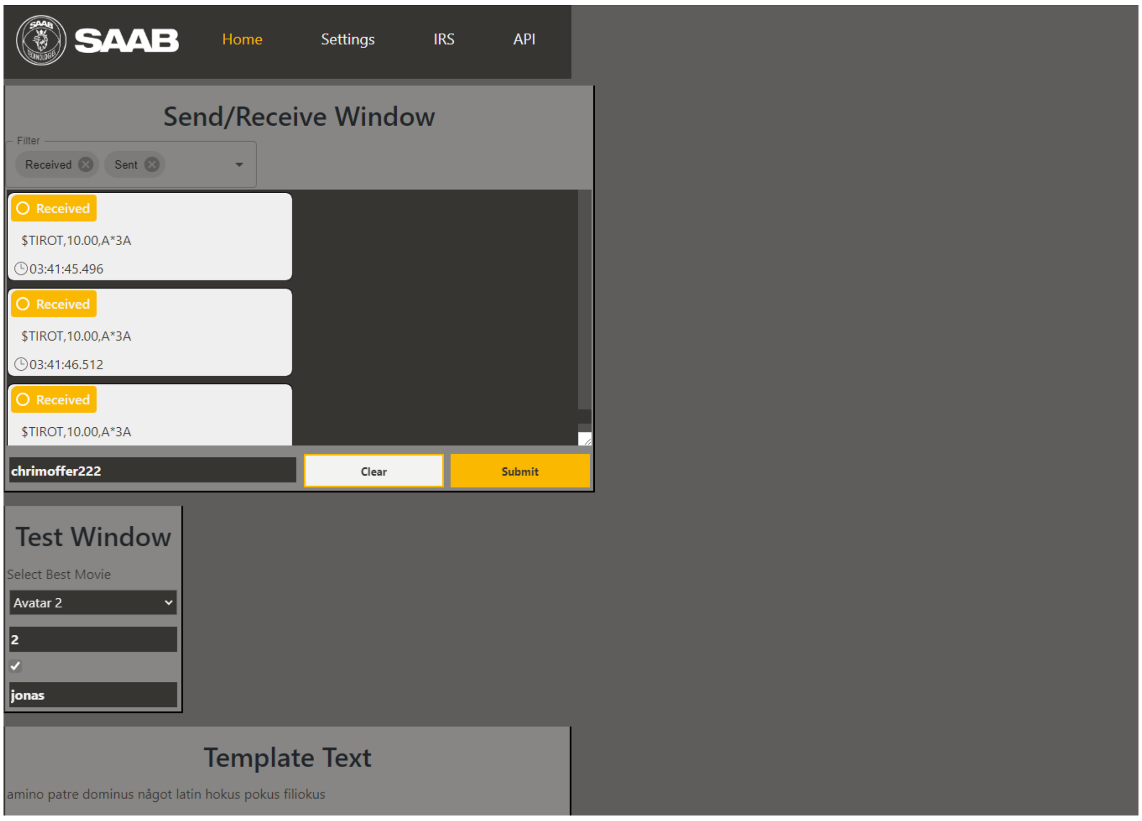


| Status | Message |
|---|---|
| Sent | 2 |
| Sent | 2 |
| Sent | 2 |
| Sent | 2 |
| Sent | 2 |
| Sent | kdbkdfkdfbg |

kdbkdfkdfbg

false

Initial design of the message viewer.



$TIROT,10.00,A*3A
03:41:45.496

$TIROT,10.00,A*3A
03:41:46.512

$TIROT,10.00,A*3A

chrimoffer222

2

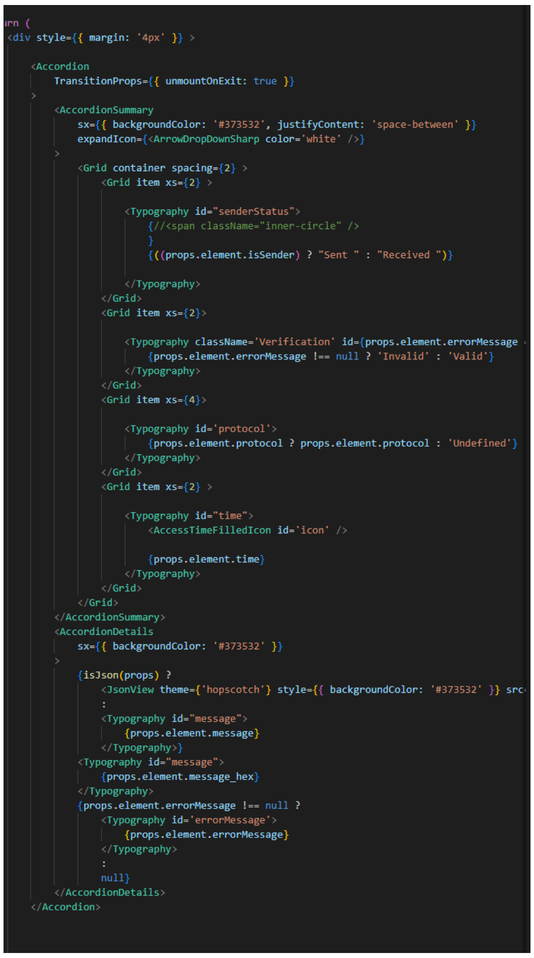
✓

jonas

amino patre dominus något latin hokus pokus filiokus

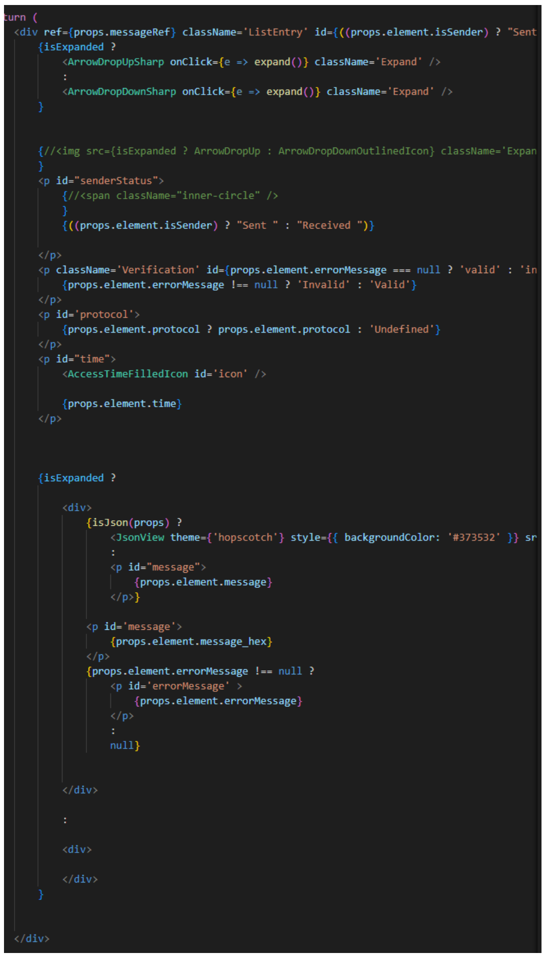
Experimental layout for the navigation bar, updated design for the message window and generation of inputs from dummy-data.

# Appendix C
# MUI comparison



```
urn (
 <div style={{ margin: '4px' }} >

    <Accordion
        TransitionProps={{ unmountOnExit: true }}
    >
        <AccordionSummary
            sx={{ backgroundColor: '#373532', justifyContent: 'space-between' }}
            expandIcon={<ArrowDropDownSharp color='white' />}
        >
            <Grid container spacing={2} >
                <Grid item xs={2} >

                    <Typography id="senderStatus">
                        {//<span className="inner-circle" />
                        }
                        {((props.element.isSender) ? "Sent " : "Received ")}

                    </Typography>
                </Grid>
                <Grid item xs={2}>

                    <Typography className='Verification' id={props.element.errorMessage
                        {props.element.errorMessage !== null ? 'Invalid' : 'Valid'}
                    </Typography>
                </Grid>
                <Grid item xs={4}>

                    <Typography id='protocol'>
                        {props.element.protocol ? props.element.protocol : 'Undefined'}
                    </Typography>
                </Grid>
                <Grid item xs={2} >

                    <Typography id="time">
                        <AccessTimeFilledIcon id='icon' />

                        {props.element.time}
                    </Typography>
                </Grid>
            </Grid>
        </AccordionSummary>
        <AccordionDetails
            sx={{ backgroundColor: '#373532' }}
        >
            {isJson(props) ?
                <JsonView theme={'hopscotch'} style={{ backgroundColor: '#373532' }} src
                :
                <Typography id="message">
                    {props.element.message}
                </Typography>}
            <Typography id="message">
                {props.element.message_hex}
            </Typography>
            {props.element.errorMessage !== null ?
                <Typography id='errorMessage'>
                    {props.element.errorMessage}
                </Typography>
                :
                null}
        </AccordionDetails>
    </Accordion>
```

MUI version using Accordion and Typography.



```
turn (
 <div ref={props.messageRef} className='ListEntry' id={((props.element.isSender) ? "Sent
    {isExpanded ?
        <ArrowDropUpSharp onClick={e => expand()} className='Expand' />
        :
        <ArrowDropDownSharp onClick={e => expand()} className='Expand' />
    }


    {//<img src={isExpanded ? ArrowDropUp : ArrowDropDownOutlinedIcon} className='Expan
    }
    <p id="senderStatus">
        {//<span className="inner-circle" />
        }
        {((props.element.isSender) ? "Sent " : "Received ")}

    </p>
    <p className='Verification' id={props.element.errorMessage === null ? 'valid' : 'in
        {props.element.errorMessage !== null ? 'Invalid' : 'Valid'}
    </p>
    <p id='protocol'>
        {props.element.protocol ? props.element.protocol : 'Undefined'}
    </p>
    <p id="time">
        <AccessTimeFilledIcon id='icon' />

        {props.element.time}
    </p>



    {isExpanded ?

        <div>
            {isJson(props) ?
                <JsonView theme={'hopscotch'} style={{ backgroundColor: '#373532' }} sr
                :
                <p id="message">
                    {props.element.message}
                </p>}

            <p id='message'>
                {props.element.message_hex}
            </p>
            {props.element.errorMessage !== null ?
                <p id='errorMessage' >
                    {props.element.errorMessage}
                </p>
                :
                null}


        </div>

        :

        <div>

        </div>
    }


 </div>
```

Lightweight custom version.

# Appendix D
# Solutions functionality

## The solution's functionality includes:

- Sending specified field-based protocols.
- Receive specified field-based protocols.
- Dynamic and lightweight specification of protocol.
- Validation of messages according to specified protocol.
- Error description.
- Filter between received and sent messages.
- Filter between valid and invalid messages.
- Show messages in Hexadecimal format.
- Show messages in ASCII format.
- List view.
- Latest message view in fields.
- Multi-cast UDP.
- Serial communication.
- Time received measurement.
- Network settings.
- Display PDF of choice.

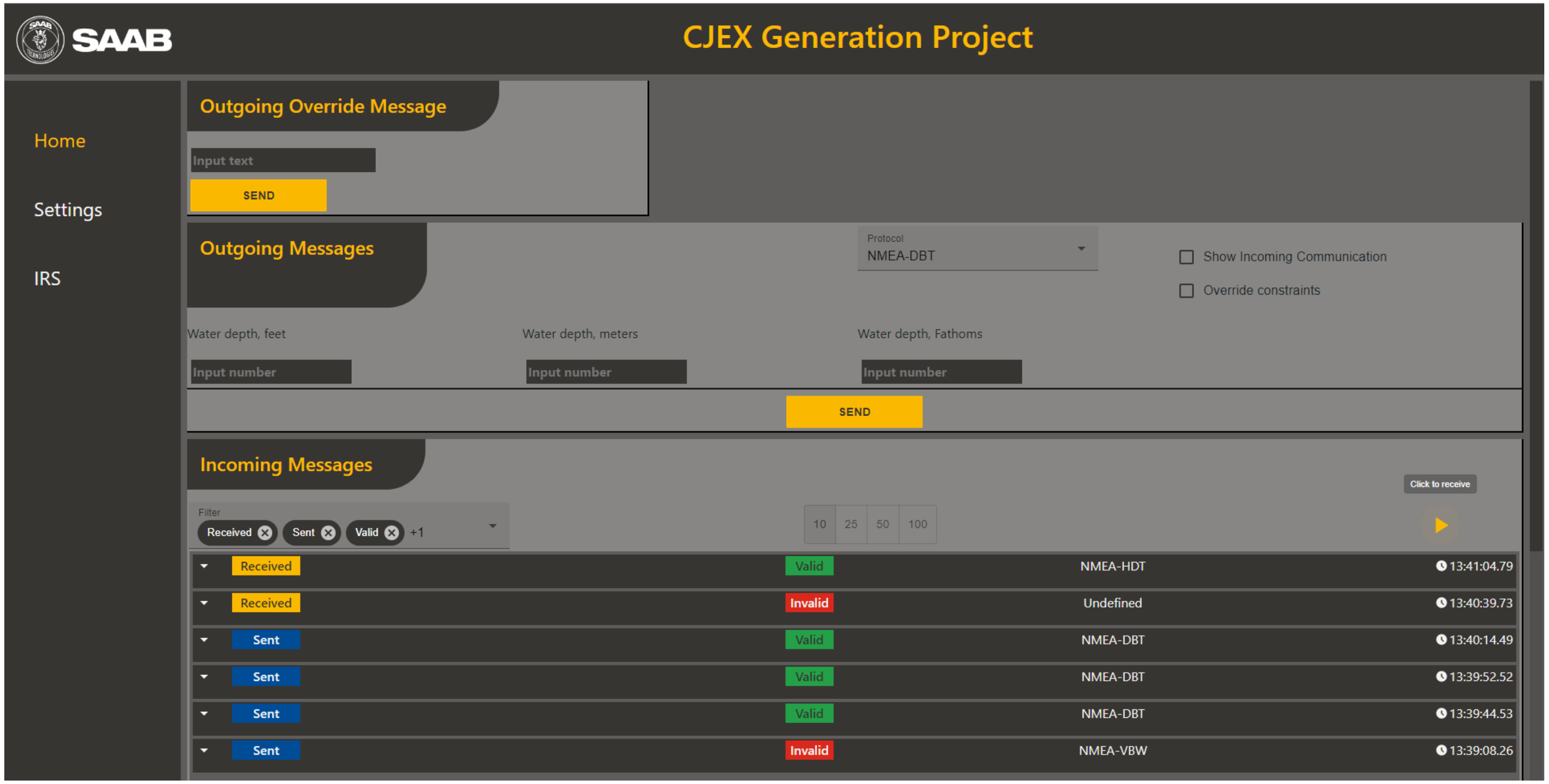
- Real-time edit of specified protocols.
- Dark Mode.
- Allows user to send faulty data if specified(e.g 361 degrees).
- Stops user when sending data outside of specified ranges.
- Modern appearance (according to product owner).
- Expandable windows.
- Expandable message.
- Ability to stop retrieval.

# Appendix E
# XML reference

| Tag | params | Example | Values | Description |
|---|---|---|---|---|
| &lt;Protocol&gt; | type | type="NMEA" | Any | |
| | Children | &lt;Protocol&gt; &lt;Sentence/&gt; &lt;/Protocol&gt; | | |
| | | | | |
| &lt;Sentence&gt; | id | id="VBW" | Any | |
| | talkerid | talkerid="VD" | Any | |
| | Children | &lt;Sentence&gt;&lt;Group/&gt; &lt;/Sentence&gt; | &lt;Group&gt; | |
| | | | | |
| &lt;Root&gt; | id | id="V1" | Any | Can replace &lt;Sentence&gt; |
| | title | title="Template" | Any | |
| | | | | |
| &lt;Group&gt; | Children | &lt;Group&gt; &lt;Field/&gt; &lt;/Group&gt; | Fields | Used to show group in GUI. |
| | | | | |
| &lt;Field&gt; | type | type="float" | float,unit,bool,int,string | Required. |
| | name | name="water depth" | ASCII, A-Z, space and comma. | Required, Unique. |
| | id | id="waterdepth1" | ASCII | Required, Unique. |
| | min | min="0" | float, int, time | minimum allowed field input. Default none. |
| | max | max="360" | float, int, time | Maximum allowed field input. Default none. |
| | minLength | minLength="5" | int | Minimum allowed characters in field. &gt;0. default: "1". |
| | maxLength | maxLength="8" | int | Maximum allowed characters in field. &gt;0. default "inf". |
| | value | value="10" | Same as type | Default value in field.Default none. &gt;0. |
| | required | required="true" | bool | Specifies if field must be filled by user. Default "true". |
| | Children | &lt;Field&gt; &lt;Option&gt; &lt;/Field&gt; | &lt;Option&gt; | |
| | | | | |
| &lt;Option&gt; | label | label="Star Wars" | Any | Used to create a dropdown with specified options in the parent field. |
| | value | value="1" | Number Unique | |

**Table E.1:** Reference for tags and variables for the XML file.

# Appendix F
# Final Design

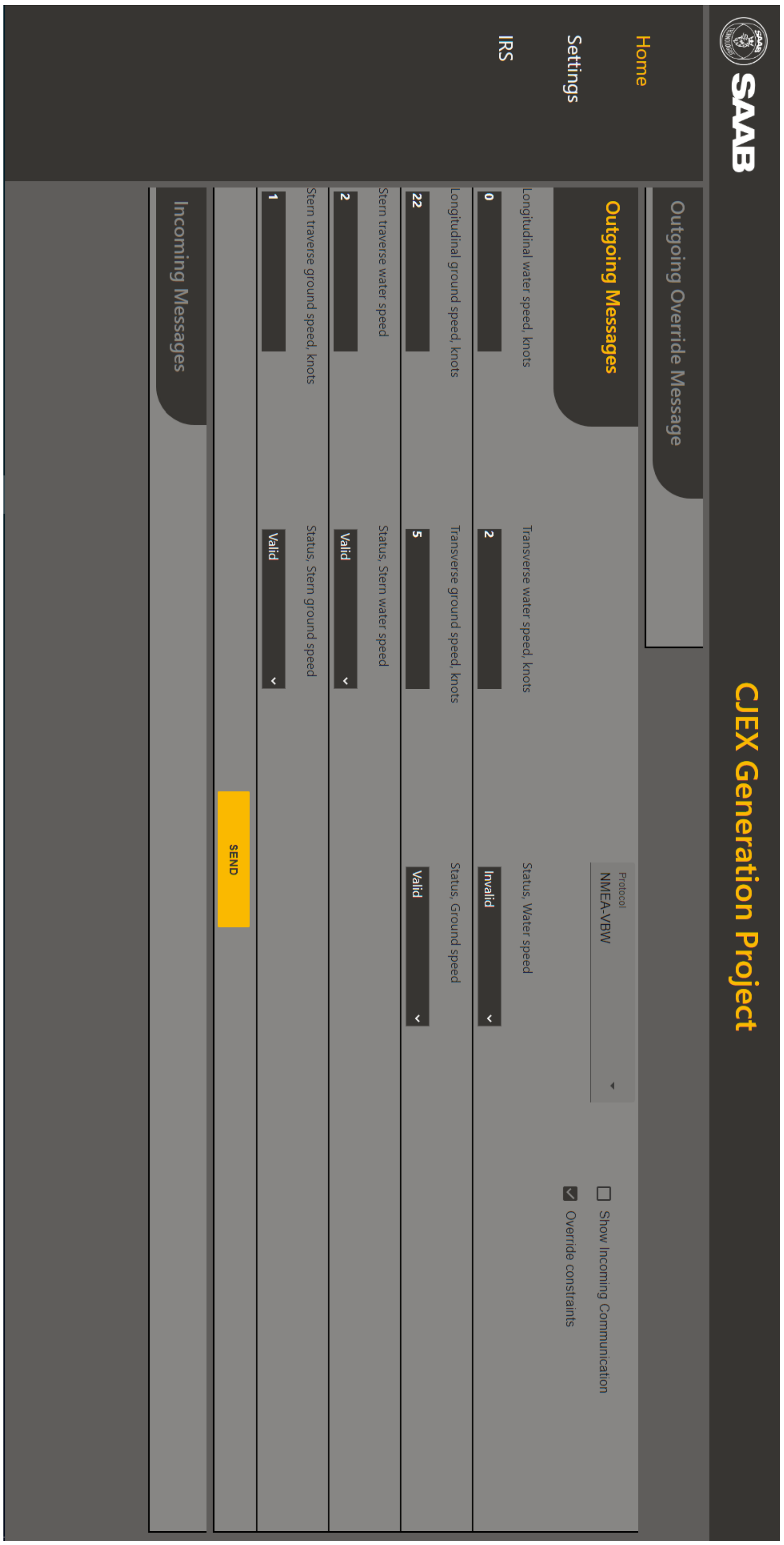


Protocol: NMEA-DBT

Filter: Received, Sent, Valid +1

| | | | |
|---|---|---|---|
| Received | Valid | NMEA-HDT | 13:41:04.79 |
| Received | Invalid | Undefined | 13:40:39.73 |
| Sent | Valid | NMEA-DBT | 13:40:14.49 |
| Sent | Valid | NMEA-DBT | 13:39:52.52 |
| Sent | Valid | NMEA-DBT | 13:39:44.53 |
| Sent | Invalid | NMEA-VBW | 13:39:08.26 |

Final design of the homepage.



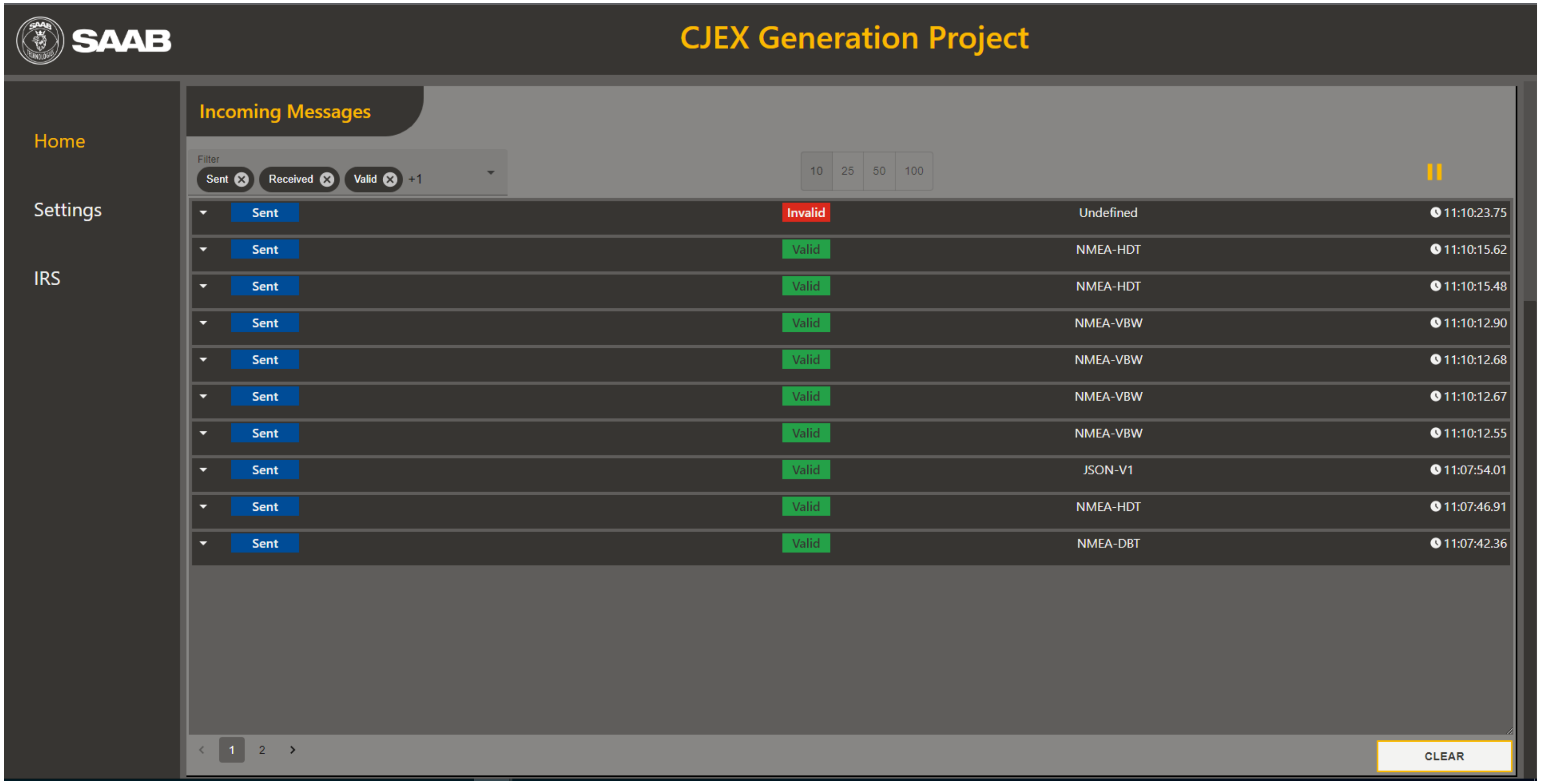
Protocol: NMEA-VBW

[ ] Show Incoming Communication

[✓] Override constraints

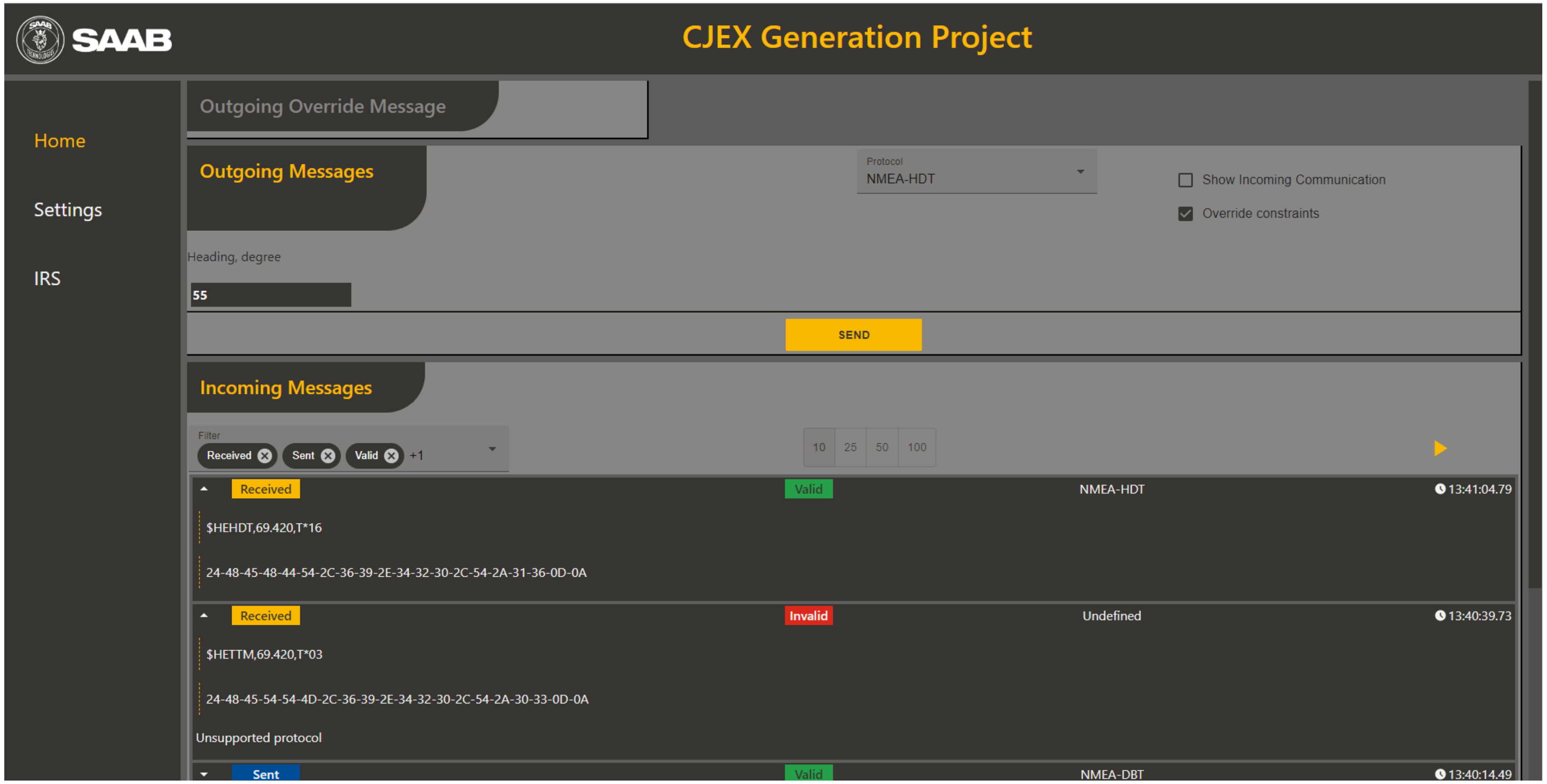
| | | |
|---|---|---|
| Longitudinal water speed, knots: 0 | Transverse water speed, knots: 2 | Status, Water speed: Invalid |
| Longitudinal ground speed, knots: 22 | Transverse ground speed, knots: 5 | Status, Ground speed: Valid |
| Stern traverse water speed: 2 | Status, Stern water speed: Valid | |
| Stern traverse ground speed, knots: 1 | Status, Stern ground speed: Valid | |

Selecting another protocol.



| | | | |
|---|---|---|---|
| Sent | Invalid | Undefined | 11:10:23.75 |
| Sent | Valid | NMEA-HDT | 11:10:15.62 |
| Sent | Valid | NMEA-HDT | 11:10:15.48 |
| Sent | Valid | NMEA-VBW | 11:10:12.90 |
| Sent | Valid | NMEA-VBW | 11:10:12.68 |
| Sent | Valid | NMEA-VBW | 11:10:12.67 |
| Sent | Valid | NMEA-VBW | 11:10:12.55 |
| Sent | Valid | JSON-V1 | 11:07:54.01 |
| Sent | Valid | NMEA-HDT | 11:07:46.91 |
| Sent | Valid | NMEA-DBT | 11:07:42.36 |

Final design of the Message viewer that shows a number of validated and one invalidated message.



Protocol: NMEA-HDT

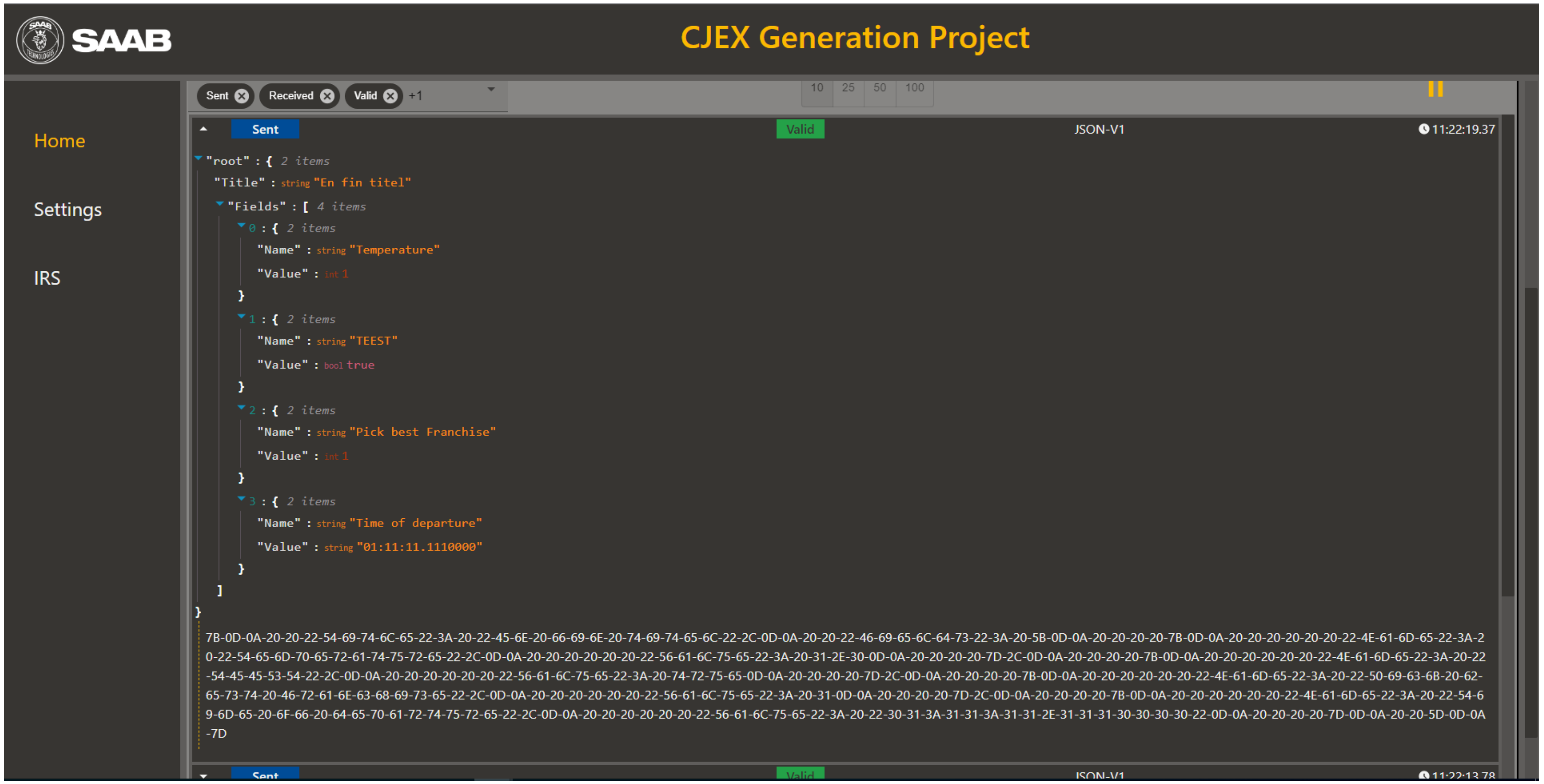
☐ Show Incoming Communication
☑ Override constraints

Heading, degree: 55

Filter: Received, Sent, Valid +1

```
Received    Valid      NMEA-HDT     13:41:04.79
$HEHDT,69.420,T*16
24-48-45-48-44-54-2C-36-39-2E-34-32-30-2C-54-2A-31-36-0D-0A

Received    Invalid    Undefined    13:40:39.73
$HETTM,69.420,T*03
24-48-45-54-54-4D-2C-36-39-2E-34-32-30-2C-54-2A-30-33-0D-0A
Unsupported protocol

Sent        Valid      NMEA-DBT     13:40:14.49
```

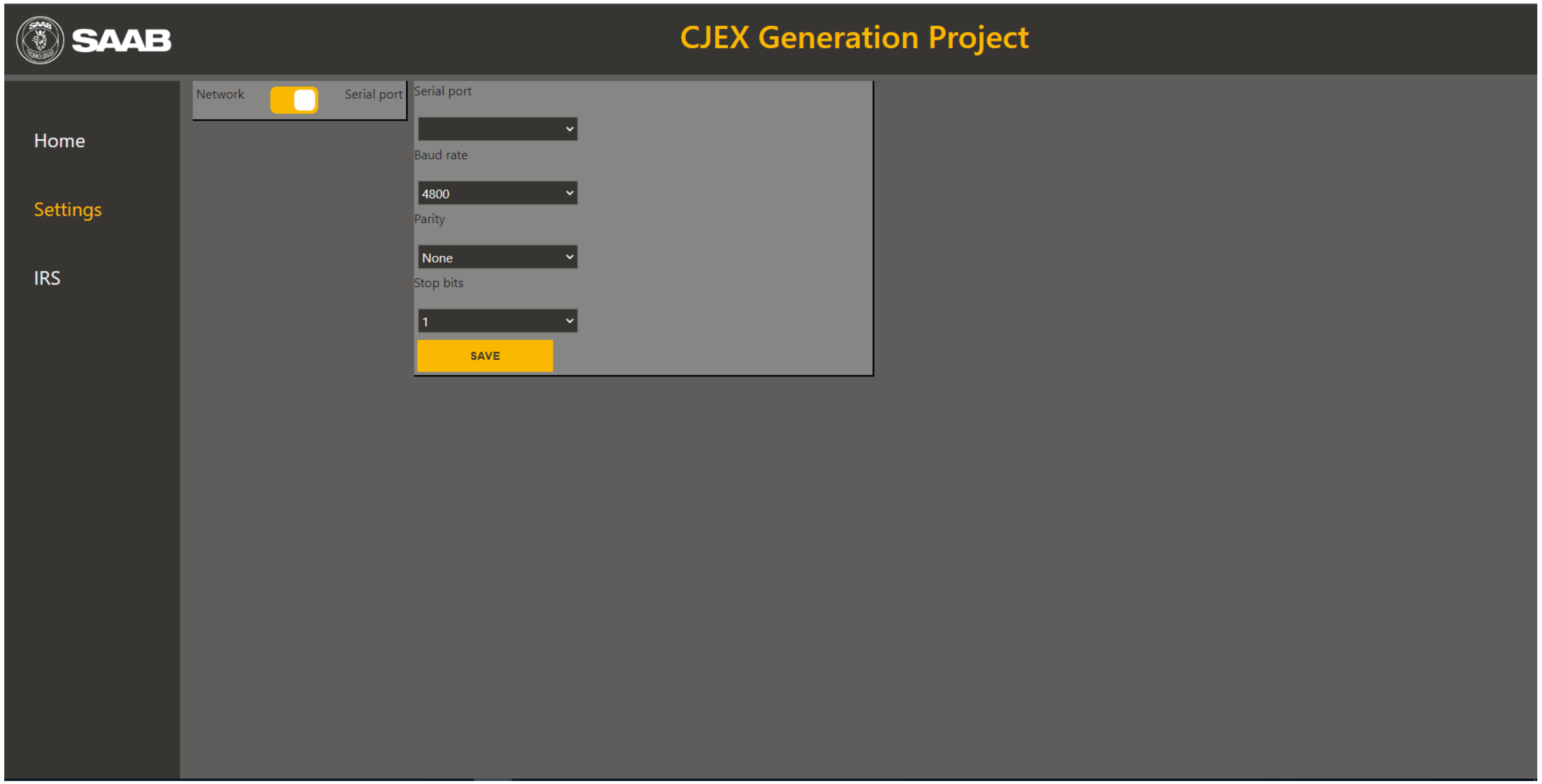
Expanded messages with the data and hexadecimal value, also shows an error for the invalidated ones.



```
"root" : { 2 items
  "Title" : string "En fin titel"
  "Fields" : [ 4 items
    0 : { 2 items
      "Name" : string "Temperature"
      "Value" : int 1
    }
    1 : { 2 items
      "Name" : string "TEEST"
      "Value" : bool true
    }
    2 : { 2 items
      "Name" : string "Pick best Franchise"
      "Value" : int 1
    }
    3 : { 2 items
      "Name" : string "Time of departure"
      "Value" : string "01:11:11.1110000"
    }
  ]
}
```

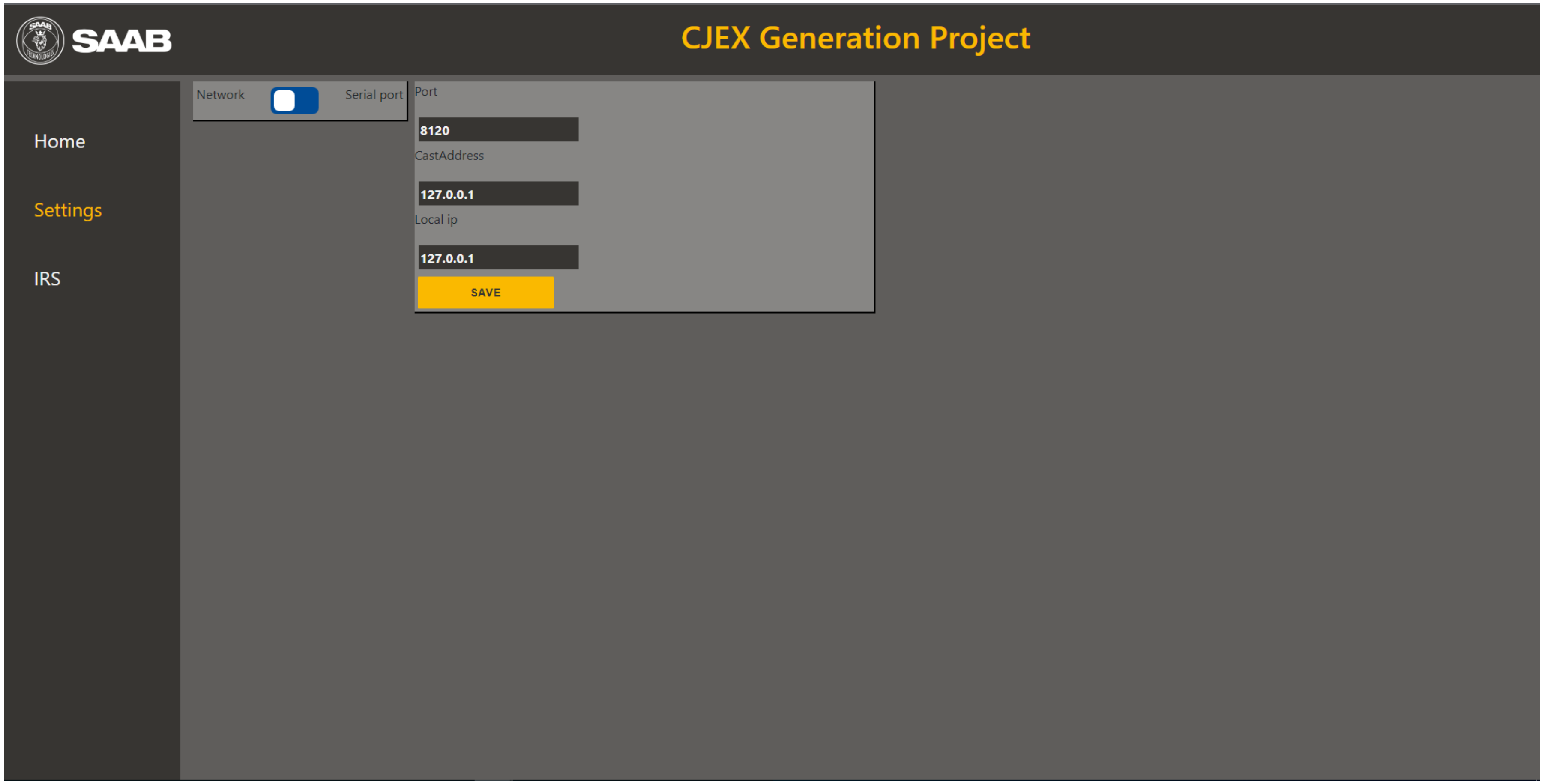
7B-0D-0A-20-20-22-54-69-74-6C-65-22-3A-20-22-45-6E-20-66-69-6E-20-74-69-74-65-6C-22-2C-0D-0A-20-20-22-46-69-65-6C-64-73-22-3A-20-5B-0D-0A-20-20-20-20-7B-0D-0A-20-20-20-20-20-20-22-4E-61-6D-65-22-3A-2
0-22-54-65-6D-70-65-72-61-74-75-72-65-22-2C-0D-0A-20-20-20-20-20-20-22-56-61-6C-75-65-22-3A-20-31-2E-30-0D-0A-20-20-20-20-7D-2C-0D-0A-20-20-20-20-7B-0D-0A-20-20-20-20-20-20-22-4E-61-6D-65-22-3A-20-22
-54-45-45-53-54-22-2C-0D-0A-20-20-20-20-20-20-22-56-61-6C-75-65-22-3A-20-74-72-75-65-0D-0A-20-20-20-20-7D-2C-0D-0A-20-20-20-20-7B-0D-0A-20-20-20-20-20-20-22-4E-61-6D-65-22-3A-20-22-50-69-63-6B-20-62-
65-73-74-20-46-72-61-6E-63-68-69-73-65-22-2C-0D-0A-20-20-20-20-20-20-22-56-61-6C-75-65-22-3A-20-31-0D-0A-20-20-20-20-7D-2C-0D-0A-20-20-20-20-7B-0D-0A-20-20-20-20-20-20-22-4E-61-6D-65-22-3A-20-22-54-6
9-6D-65-20-6F-66-20-64-65-70-61-72-74-75-72-65-22-2C-0D-0A-20-20-20-20-20-20-22-56-61-6C-75-65-22-3A-20-22-30-31-3A-31-31-3A-31-31-2E-31-31-31-30-30-30-30-22-0D-0A-20-20-20-20-7D-0D-0A-20-20-5D-0D-0A
-7D

How a JSON message is shown when expanded.



Serial port: 
Baud rate: 4800
Parity: None
Stop bits: 1

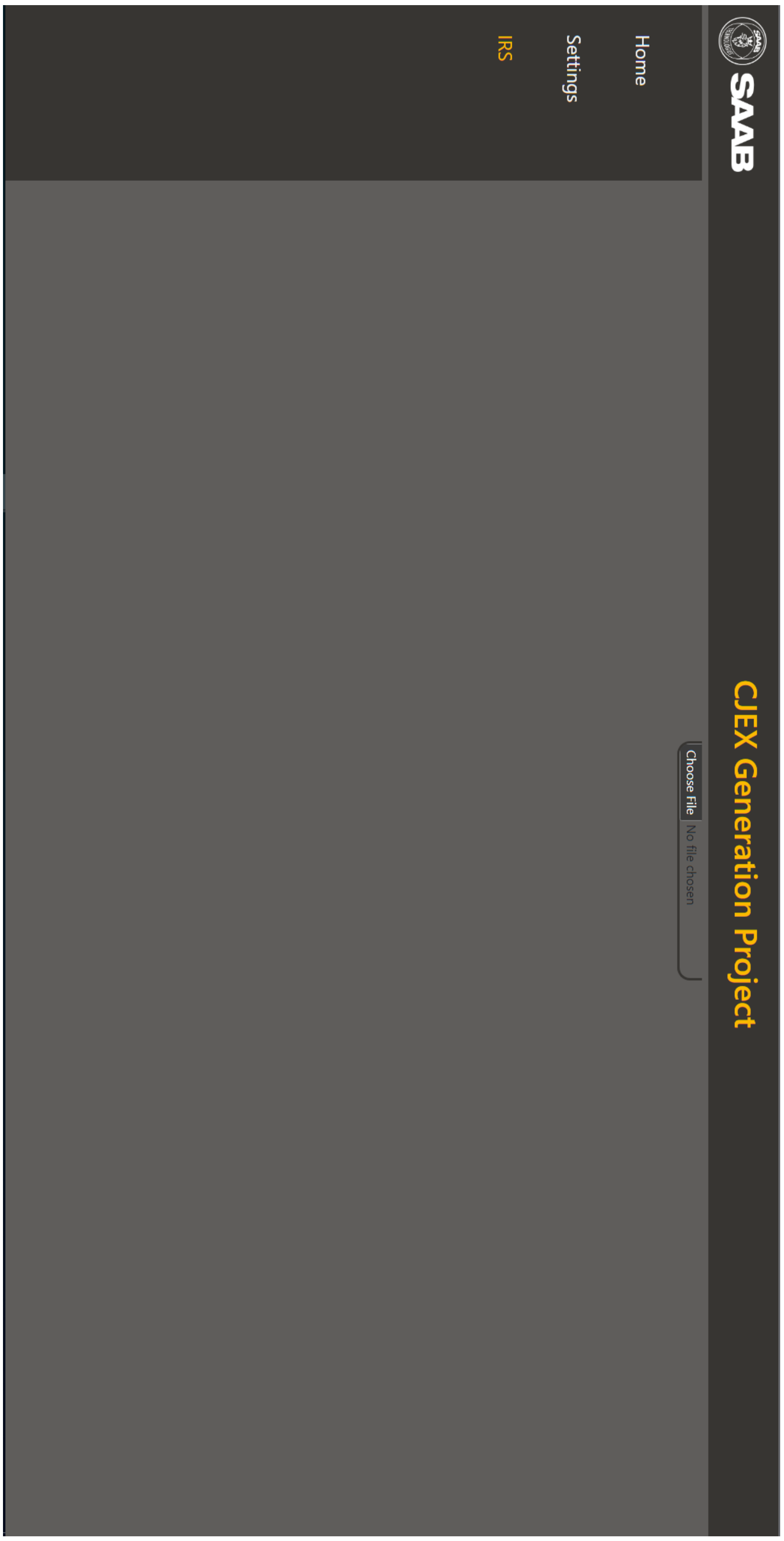
Settings page for serial-port, currently no serial-port could be found in this instance since the PC running it has none.



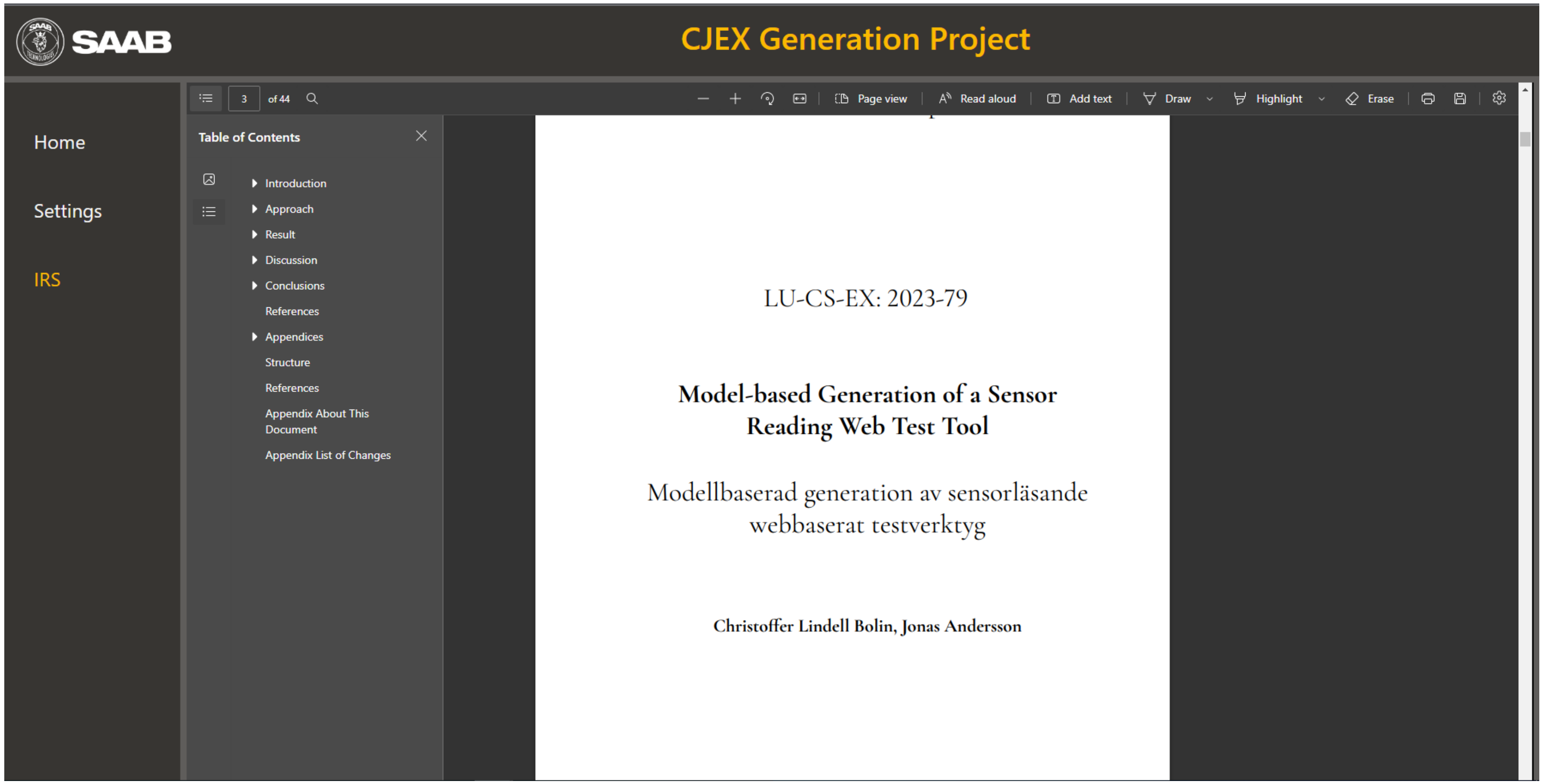
CJEX Generation Project

Port
8120
CastAddress
127.0.0.1
Local ip
127.0.0.1

Settings page for multicast UDP.



PDF selection for IRS.



LU-CS-EX: 2023-79

Model-based Generation of a Sensor Reading Web Test Tool

Modellbaserad generation av sensorläsande webbaserat testverktyg

Christoffer Lindell Bolin, Jonas Andersson

An Example PDF being displayed.

FACULTY OF ENGINEERING | LUNDS TEKNISKA HÖGSKOLA | PRESENTED 2022-08-23

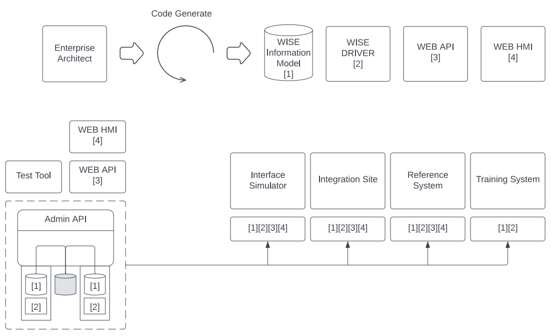
**EXAMENSARBETE** Model-based Generation of a Sensor Reading Web Test Tool
**STUDENTS** Jonas Andersson, Christoffer Lindell Bolin
**SUPERVISOR** Marcus Klang (LTH)
**EXAMINER** Jesper Öqvist (LTH)

# Dynamic Sensor Reading Web Test Tool

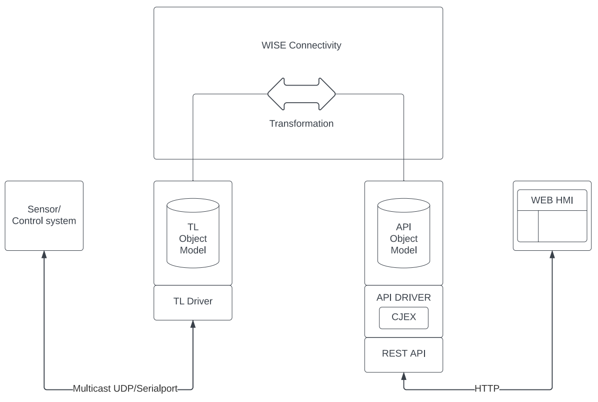
POPULAR SCIENCE SUMMARY **Jonas Andersson, Christoffer Lindell Bolin**

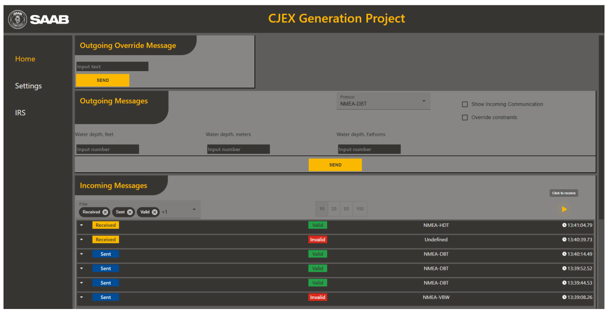
Competence for software can be rather expensive, you do not want to waste work on duplicate, redundant software. This thesis aims to remove some repeated work for Saab Kockum's engineers and save time. This is achieved with a test tool that can handle various field-based protocols specified in an XML-file by the user.

Saab's current test tool is implemented from scratch by a third-party for each new protocol and sensor. This is problematic because it wastes time that could be spent on more innovative tasks. This project sets forth a solution which uses XML to edit generated components to adapt to one or more specified protocols to be tested for. The XML file allows for specification of a field-based protocol which alters not only the communication supported through Saab Kockum's WISE connectivity, but the inputs and outputs generated in a web-based Graphical User Interface (GUI). While the previous test tools needs weeks to re-implement, the new solution simply require the user to change the XML. This allows for lightweight and rapid adaptation.



The solution was quantitatively compared to the previous test tool and there were not significant difference in amount of functionality, and with a more flexible foundation, there is more potential to further develop, which is advised using the documented possibilities.





Protocol: NMEA-DBT
☐ Show Incoming Communication
☐ Override constraints
Water depth, feet: Input number
Water depth, meters: Input number
Water depth, Fathoms: Input number
Filter: Received, Sent, Valid +1

| | | | |
|---|---|---|---|
| Received | Valid | NMEA-HDT | 13:41:04.79 |
| Received | Invalid | Undefined | 13:40:39.73 |
| Sent | Valid | NMEA-DBT | 13:40:14.49 |
| Sent | Valid | NMEA-DBT | 13:39:52.52 |
| Sent | Valid | NMEA-DBT | 13:39:44.53 |
| Sent | Invalid | NMEA-VBW | 13:39:08.26 |