Western Norway University of Applied Sciences

Faculty of Engineering and Science Department of Computing, Mathematics and Physics

SensorShip

Extracting and visualizing vessel data

Bachelor, Computing

Department of Computing, Mathematics and Physics

Faculty of Engineering and Science

Submission date: 03/06-2019

Number of words: 11437

John Bruhaug

Adrian Solheim

Petter A. Salberg

Daniel N. Pettersen

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: SensorShip – Extracting and Visualizing Vessel Data	Dato: 03.06.2019
Forfatter(e): John Bruhaug, Adrian Solheim, Daniel N. Pettersen, Petter A. Salberg	Antall sider u/vedlegg: 50 Antall sider vedlegg: 51
Studieretning: Dataingeniør og Informasjonsteknologi	Antall disketter/CD-er: 0
Kontaktperson ved studieretning: Carsten Gunnar Helgesen	Gradering: Ingen
Merknader:	

Oppdragsgiver: Kyma AS	Oppdragsgivers referanse:
Oppdragsgivers kontaktpersoner: Trond B. Kvamme	Telefon: +47 952 65 107

Sammendrag:

Prosjektet handler om å demonstrere hvordan man kan bruke Kyma sitt API, slik at Kyma sine interessenter kan forstå verdien av deres API og dataen den gir tilgang til. Dette ble gjennomført ved å lage en skrivebords applikasjon i JavaScript. Denne applikasjonen benytter Kyma sitt API til å hente data og visualisere disse ved hjelp av grafer. Ved å visualisere dataene viser vi verdien av dem og gjør de lettere å forstå. Etter samtaler med Kyma har vi evaluert prosjektet som en suksess.

Stikkord: JavaScript, Vue, Electron, API, Kyma, Windows.

Preface

This report is a summary of the bachelor project in Computing at Western Norway University of Applied Science, by students John Bruhaug, Adrian Solheim, Petter A. Salberg and Daniel N. Pettersen. The report has been written during the semester where the students have been developing an application for Kyma AS. The report has regularly been updated during the project.

During the last three months the project group has learnt a new programming language and multiple different tools, and applied them to develop a dashboard application, and connected this application to an API. The students encountered numerous bugs and issues that had to be solved along the way.

The students would like to thank Kyma for the opportunity to work on such an interesting project with the ability to choose the technology stack ourselves, and for the office space with a beautiful view that was granted to us during the project's duration.

We would like to give a special thanks to Trond B. Kvamme for checking in on us regularly and giving us constructive feedback along the way. We also appreciate his comments and suggestions on Github.

We would also like to thank our advisor Violet Ka I Pun for her efforts in guiding us through this project and rigorously reviewing our reports and documentation throughout the project.

Term List

REST-API - Representational State Transfer - Application Programming Interface is a way to transfer the state of a programming object through a web service.

JSON - JavaScript Object Notation is a standard format for data. It is a way of sending data stored in objects.

Vessel sensors - Sensors on the vessel collect data and sends it to Kyma. They store it in their data center.

Technology stack - Are all the services, programming languages, and frameworks used to build one single application

DOM- Document object model. It is it defines HTML elements as objects.

SQL - Structured Query Language, used to access and manage some databases.

Front-end - The part of the application that the user interacts with directly

Back-end - the part of the application that is not directly accessed by the user. Workers in the background

Performance data - Data about how the vessel is performing.

Paging - A page is what the user sees on the device screen at a certain point in the application, such as a login page or a home page. Paging means reducing the number of pages in the application.

Scalability - The ability to customize the screen size of the application.

CSV - Comma Separated Values. A form of formatting data when storing as file.

Github - A software development platform.

Granularity - The scale or level of detail in a set of data.

Repository - A data structure used to store metadata for a set of files and/or directories. It also stores the history of changes made to the files and directories.

Sprint - A short time-boxed period where a team works to complete a set amount of work.

RAM - Random access memory, type of storage unit used in computers, which gives access to all the saved data in a random order.

Data center - A building, dedicated space within a building or a group of buildings used to house computer systems and associated components, such as telecommunications and storage systems.

UI - User Interface: Determines if the application is presentable to the users of the application/website.

UX - User Experience: Determines ease of use, a good UX makes sure the user knows what to do at each point.

1. Introduction

1.1 Goal and Motivation

Kyma is a company that provides vessel performance monitoring tools for vessel owners and clients. They make sensors in addition to collecting all the data from the vessels to a data center and making the data available through their API, the Kyma API. Kyma's clients have a hard time understanding what the Kyma API is and how you can use it. Kyma wants a "proof of concept" that illustrates the value and the usefulness of the API.

Therefore the goal of this project is to make an application that will demonstrate how one can use the API so that everyone involved in the industry can understand the value of the Kyma API, even if they have no computer background.

To achieve this we will make a dashboard application that will get data about vessels from the data center through the Kyma API and display it in a way that the information can be understood. Instead of getting loads of numbers for the owners to read, we take those numbers and turn them into graphs that paints a picture of how the vessels performs, and the status of the vessels.

The application is going to be lightweight and easy to use and will quickly fetch the data from the API and store it on the device. Then it will show the data that is the most useful to the owners visualizing it in different graphs. This will be a powerful tool for Kyma to show their clients what can be achieved using the Kyma API.

1.2 Context

Kyma wants an application that they can use to show the value of the data they get from the sensors of the vessels (see Figure 1.1) which can be fetched from the Kyma API. The application will be a prototype to give clients a better understanding of the information that the sensors of the vessels provides, giving them the ability to analyze the data and gain a better understanding of the performance of the vessels.

It is valuable for a vessel owner to get the sensor data for multiple reasons. For instance, they can use the sensor data to check if the vessels conforms to the specifications promised by the shipwright. The vessel might perform worse generating extra costs. They can use the sensor data to prove performance loss and request reimbursement. They can also analyze fuel consumption of different types of fuel to try to maximize the vessel's cost efficiency.

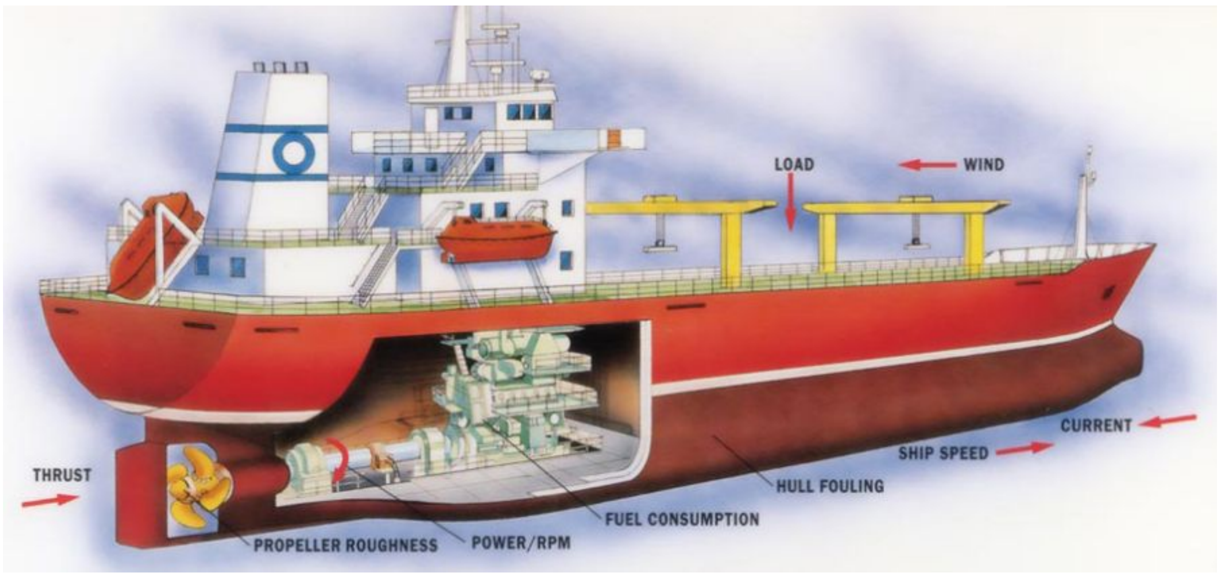


Figure: 1.1 Sensors on a vessel[1].

An API is an interface that allows two software programs to communicate with each other. The API defines the correct way for a developer to write programs that requests services from an operating system or other applications.

"The Kyma WEB-API provides an interface between the customer's own computer systems and the 'Kyma Cloud' which is a repository for the raw logging data received from the Kyma Ship Performance System on-board their vessels."[2]. Kyma allows their users to connect to their API. With this connection the clients have the ability to create their own applications for fetching data on demand. The 'Kyma Cloud' is a data center that contains data provided by sensors on vessels and is updated on fixed intervals.

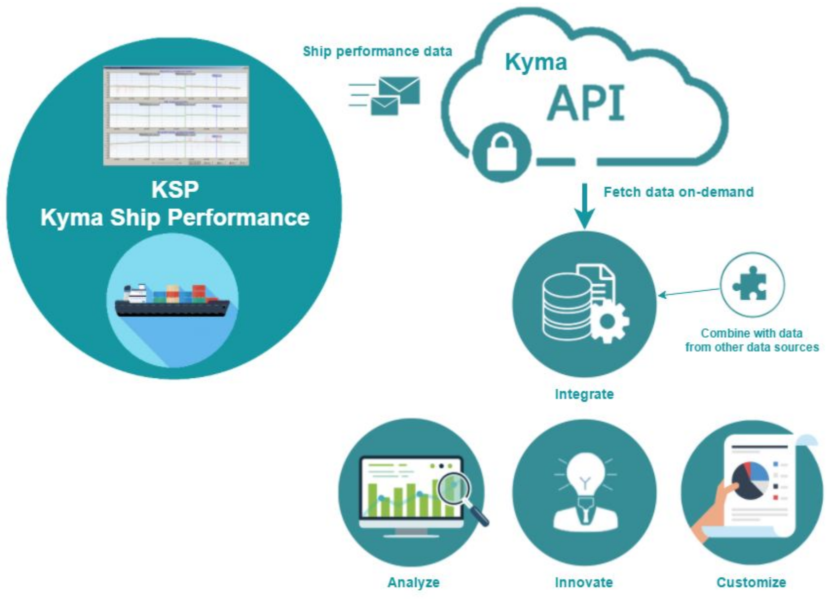


Figure 1.2: The Kyma system[2].

Figure 1.2 shows an overview of the Kyma System where our application would involve the integration part. Our application mainly lies in the integration phase and serves as an interface between the API and the clients, who can analyze and customize for their needs.

1.3 Limitations

The application is going to run on a Microsoft Surface. The hardware on Microsoft Surface is limited in certain aspects. The biggest one being It has less memory than a regular laptop, and we need to keep this in mind when we develop this application.

Lack of experience is a limiting factor considering no one in the project group has any noticeable experience with the technology stacks needed to develop a desktop application, or within such a large project.

Time is our biggest constraint because of the time it takes to not only develop the application, but also learn the necessary tools. In addition, we also need to spend time writing the report and documentation for the project.

1.4 Resources

Resources that Kyma provides to us is the access to a demo environment through their API, office space for the project group, and feedback. Kyma has a lot of data that can be used in a demo environment and this data is considered a resource that Kyma provides. In addition, we have access to relevant expertise from Kyma and they also provide us with a Github repository. This repository will be used to store our project code. Kyma can also follow along with our work on this repository and give feedback.

Official documentation for our chosen technology stacks will be a key resource for us before and during development. This documentation can be found online and will greatly assist us in learning the technology stacks and finding solutions to problems that may arise during development.

Our internal supervisor at HVL is also a resource that will benefit us, regarding the administrative work from HVL including this report.

1.5 Organization of the Report

- **Chapter 1** starts with an introduction to our project. This chapter contains our goals and motivations, context, limitations and resources available to us.
- **Chapter 2** gives a description of Kyma, what previous work our project is built upon and what the project owner`s initial requirements specification are. The chapter also describes our initial solution idea.
- **Chapter 3** explores different approaches to this project where we list what technologies we can use. Further we explain the project specification in more detail and what tools and technologies we have chosen to use for this project. We also go through how the project is organised and planned, how the risk is managed and how the results will be evaluated at the end of the project.
- **Chapter 4** describes in the detail how the application is made and how it functions.
- **Chapter 5** explains in more detail what methods we used to evaluate the results.
- **Chapter 6** describes the project results with regards to the evaluation.
- **Chapter 7** contains a recap where we discuss our experience and results in detail.
- **Chapter 8** contain the conclusions of the project report.

2. Project Description

2.1 Practical background

Kyma has a data center that stores a lot of data about the clients' vessels. They have an API which can be used to retrieve the data from the data center. They want an application that can retrieve this data and show it in an elegant and efficient way to the clients, to demonstrate the value of the data.

2.2 Project Owner

The company was established as a consultancy in marine engineering and naval architecture in 1965. They are specialists in vessel machinery and controls systems, and use computer-based systems for steam turbine plants and motor driven vessels. They have introduced several specialized products to the field of marine performance monitoring

Kyma AS has existed in its present form since 1. Jan 1995, as an independent share holding company. Today the company delivers high quality products for performance monitoring to all types of vessels, mainly freighters and tankers. Their products represent state of the art technology. Quality assurance is continuous and a necessary process for efficient production and development of new products. Kyma has highly qualified staff within all areas of their production, and is located in modern production facilities in Bergen, Norway.

2.3 Previous Work

This bachelor project is highly connected to the data center which is developed and maintained by Kyma AS. Data is retrieved through the Kyma API which is also developed and maintained by Kyma AS.

The API allows users to log in and make request to the data center. The API will then respond with data in a JSON format. The users can use this method to get data about their vessels.

2.4 Initial Requirements Specification

Kyma would like an application that achieves the following:

- The user can login to the application with an API-key.
- The user can use a GUI to choose what data to extract about their available vessels.
- Run on a Microsoft Surface Pro.
- Filter and only show the most useful data.
- Retrieve data and print out CSV file.

2.5 Initial Solution Idea

We imagine an application which can display the data in a meaningful way will make the data more readable and easier to understand; we think this will be a good solution to show the value of the data stored. The solution should have a simple and easy to use interface, making the filtration of data as quick as possible.

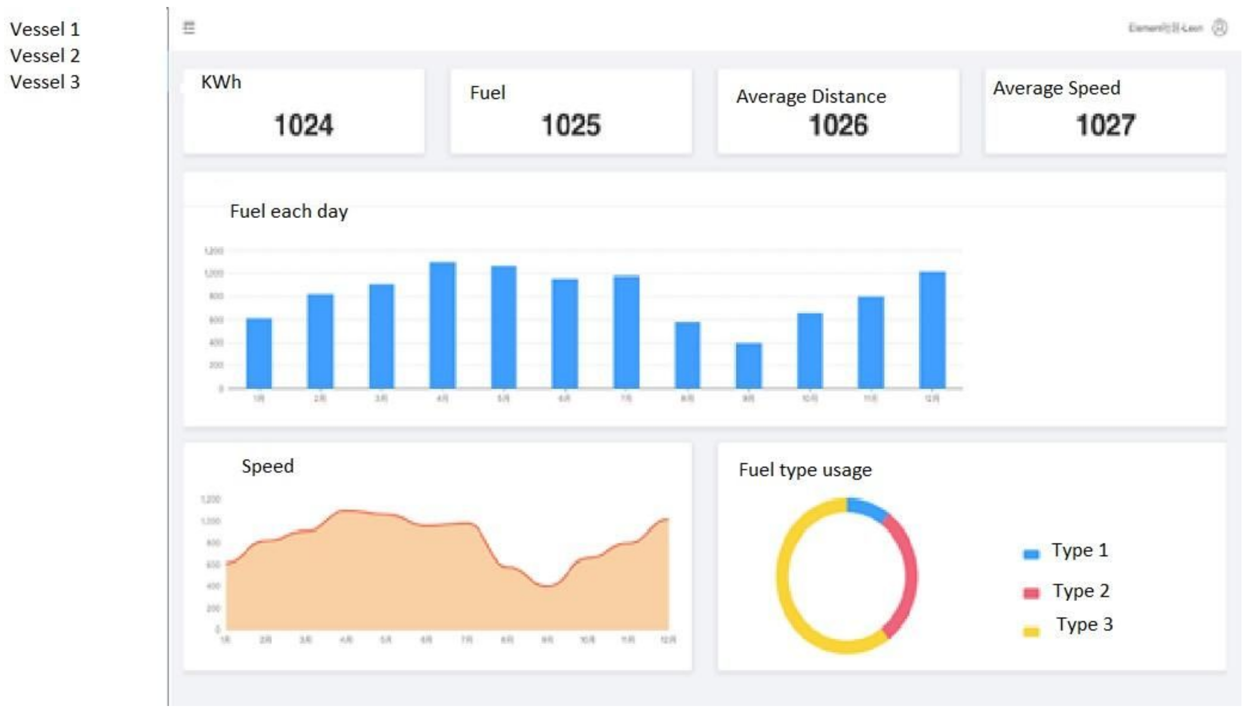


Figure 2.1: Initial idea of what the application should look like. Image is slightly modified from source to fit our project[3].

3. Project Design

3.1 Possible Approaches

There are a several approaches we have considered, each with different strengths and weaknesses. Following is a deeper analysis of these approaches, followed by a discussion.

3.1.1 Alternative Approach – Hybrid Approach with Electron and D3

The chosen approach is working with Electron because the data retrieved by the API is already in JSON format, and to work in JavaScript is the most suitable choice. It is easy to port to a desktop environment, which suits the platform (Windows) that we are aiming for. Electron has high level abstractions, through JavaScript libraries, that will speed up the development process and allow us to produce a prototype in less time. As with React, Electron has a lot of data visualization options to choose from due to the immense quantity of JavaScript libraries. We intend to use the data visualization library D3 in the project. Though all of these great qualities, its high level abstractions could have a negative impact on our runtime efficiency, and applications built with electron are also large in size and may not be able to run on smaller and older systems.

3.1.2 Alternative Approach – Native C++ with Qt

Qt in C++ would offer us a lot of power if that would be necessary for us. It also does include its own data visualization library called Qt Data Visualization which uses OpenGl as its hardware accelerator. Considering the context of our project in terms of running an application on Windows, this would be a perfect fit for us with regards to running an application on Windows. However, we felt that for our timespan it may take too long to develop an application in Qt considering its codebase is in C++. C++ provides the ability for more fine-grained control over how the program should run, but does require writing more code and writing it more carefully.

3.1.3 Alternative approach – Native/Hybrid Approach with React Native

React uses JavaScript and has a strong connection to web application development. However, it does not include the document object model DOM with HTML or styling for its render process. It uses a virtual DOM which could potentially complicate things when importing libraries that makes use of regular DOM. By using JavaScript the application would also be able to support a lot of the libraries that Electron does in Node.js, such as our chosen data visualization library D3. React native can be ported to several platforms with some changes in our code because it is native and does need native elements to be specified for each platform.

3.1.4 Alternative Approach – Native with Xamarin and C#

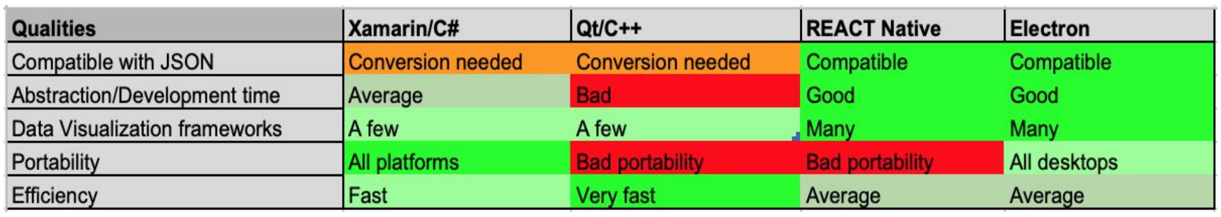
Xamarin is a cross platform framework built on the C# .NET framework. The main appeal of the framework is that it can take advantage of platform-specific hardware acceleration, which provides efficient performance. Our application does not need to be cross platform, but it would facilitate our development process and it would be an extra bonus. Despite all the benefits that Xamarin could provide us with a few downsides, of which the greatest was that it has very high monetary cost. Another downside of it was that it does not do data visualization as well as JavaScript.

3.1.5 Alternative Approach – Data Science Approach with Python and Plotly

Python was another very viable option with libraries like matplotlib and plotly. The Plotly library makes it easy to handle JSON formatted data with extensive methods to plot the data. Plotly is built on top of D3.js, which is a JavaScript library, and is easier to use and more feature rich than matplotlib. However, this only gets us to data visualization. We would still need an underlying framework to create the application to display the data visualization.

3.1.6 Discussion of Alternative Approaches.

The chosen approach for our project was a hybrid approach with Electron-Vue, and D3 as the data visualization library. When choosing the approach, we had to keep in mind a few factors, mainly its compatibility with JSON, omitting the data processing layer. Secondly, the high abstractions of JavaScript allow a higher rate of development over time and allows us to focus on the data visualization, rather than the visual layout and developing an efficient and fast application.



Qualities	Xamarin/C#	Qt/C++	REACT Native	Electron
Compatible with JSON	Conversion needed	Conversion needed	Compatible	Compatible
Abstraction/Development time	Average	Bad	Good	Good
Data Visualization frameworks	A few	A few	Many	Many
Portability	All platforms	Bad portability	Bad portability	All desktops
Efficiency	Fast	Very fast	Average	Average

Figure 3.1: Comparison model of our different tech stack approaches, red is bad and green is good.

3.2 Specification

The application will have to be a desktop application that is able to run on a Microsoft Surface. The user need to authenticate through a login page with an API key, which will redirect into the main dashboard. The dashboard will show on entry a list of all owned vessels linked to that API key. By selecting a vessel one would enter a monitor view with its specific data. This view includes options for filtering, selecting and visualizing the data giving the ability to analyse and gain new insight into the data.

3.3 Selection of Tools and Programming Languages

We are writing the application in JavaScript. JavaScript is a programming language that has a plethora of frameworks and libraries. They are powerful tools which will be extremely helpful in creating our application.

Vue.js

This is a framework which will help with the structure of our code, allow communication between the different parts of our application.

https://vuejs.org/

Node.js

This is a JavaScript runtime environment which executes code outside of the browser. It will help us with the running of our code and making the testing much more effective.

https://nodejs.org/

Electron.js

Electron is a framework, it allows for the development of desktop applications using both front end and back end components. Electron uses Node.js, but has many additional features making it easier to use for development.

https://electronjs.org/

D3.js

D3 is a JavaScript library for producing dynamic, interactive data visualizations in web browsers. It makes use of the widely implemented SVG, DOM elements, and CSS standards. This will help us in making the visual aspect of the application.

https://d3js.org/

Git

Git is the version control software we are going to use. It will allow us to have the latest working version of our code readily available. It makes sharing changes and updating progress easy and also shows a log of what has been done.

https://git-scm.com/

Chart.js

Chart.js is a JavaScript library for drawing graphs in web browsers like D3, but not as powerful. Unlike D3, Chart.js does not manipulate DOM elements. It is therefore a lot easier for us to implement. Chart.js is better suited for what this project needs because we will save a lot of time implementing it and will barely notice the lost functionality of D3.

https://vue-chartjs.org/

3.4 Project Development Method

3.4.1 Development Method

We are going to use the development method Scrum which is an agile approach. It is flexible and allows us to handle problems that might occur at the early stage of the application development. It also allows us to add new parts and features to our running project code in iterations. It helps maintain oversight of the progress and project code, and decreases the chance it gets bloated and riddled with errors. Thus we think this is the ideal development method for us.[4]

It also allows us to split the project into small tasks, which are easy to assign to each member. Since we are 4 in our team, this will prevent us from working on the same parts and getting in the way of each other. It also tracks our progress precisely.[5]

We generally work together in the same room, on different parts of the project. Since we always are in proximity it is easy to ask each other for help and keep team members up to date on how far along we are with our individual work. We can then easily see how far behind or ahead we are of the schedule. This allows us to possibly change the next iteration based upon the current condition. This is the incremental aspect of Scrum which we view as vital to our project's success.

Using such a method we avoid the pitfalls of the waterfall model. In the waterfall model a working system is not implemented until the end of a project. This creates a situation where the system cannot be tested until it is completed. The system is also only functional late in the project's life cycle and any problem is therefore detected very late. Due to our timeframe we want to encounter and tackle our problems early in the life cycle.

3.4.2 Project Plan

In order to get a general idea of what needs to be done, we start with conversations with Kyma, which will give us a clearer picture of what our project goals are going to be as well as Kyma's goals and specifications. In addition to this, we created a Gantt diagram (See Appendix A) to stay on track with our deadlines at HVL and to make sure we keep our own deadlines regarding the project.

To make it clear what the current focus is and to measure our progress, our project is split up into 4 phases:

1. Pre-planning
2. Pre-project and prototype
3. Development
4. Handover and report

During pre-planning phase we choose which project to work on, do some administrative work concerning the bachelor report and establish contact with our employer.

In the phase of the pre-project and prototype, we explore different solutions and technologies, as well as work methodology. We also make a prototype that we can build upon during the development phase.

In the development phase we decided to work iteratively, where each iteration is 2 weeks long and after each iteration, we develop the application further. After each iteration we want to have a fully functional version of the application.

In the phase of handover and report we focus on correcting bugs in the application to produce the final version, and we focus on finishing the report regardless of the level of completeness of the application.

3.4.3 Risk Management

A risk is an uncertain event that can have a negative effect on the project's goals. The risks need to be identified and classified, and we started this work in the pre-project phase, but continued it into later project phases.

To make sure the risks are properly handled, we attempt to create a basic, functional and working prototype the first week and ensure that we always have a working application at all stages of development. Because of our limited experience working with larger projects we'll be taking use of Scrum as a development framework in accordance to the agile framework.

Furthermore, when it comes to version controlling we will also be using git-flow as our work method. It may slow us down slightly due to not having used it before, but that won't be significant since we're new to working as a group together. We will save time over the course of the project as our project has a strong development branch to work in and each pull request being checked over before merged. Since we are working with iterations and incrementally developing the application as a functioning product after all iterations, well know if a risk is handled. Below is a table that shows the risks with the probability it will happen, the impact it has on the project and a description of each risk.

Risk	Probability	Impact	Mitigation
Sickness	0.2	0.3	Impact will depend on severity of sickness Other project members can do the work the sick person should do. Scope of project can be reduced by choosing to not implement some features.
Complications of setting up the different components and developer tools	0.8	0.7	Get this done as soon as possible, so we will encounters possible errors in the early stage. Ask Kyma affiliates for advice.
Architectural problems	0.3	0.4	Get a demo environment running quickly. Test rigorously. Adapt.
Insufficient knowledge	0.7	0.6	Consistently reading and acquiring knowledge about the different technologies. Iterative development cycle. Focus on having running code, minimizing chance of errors we have trouble resolving.
Tiredness, poor group moral	0.2	0.9	Help one another to stay focused, and maintain the energy. Meet often and work together.

Table 3.1: Showing risks with probability and impact and how to tackle it

3.5 Evaluation Method

Having Kyma test our application would be a great way to get very useful feedback on our project. Getting feedback from the vessel owners will also be a good way to evaluate our project. The vessel owners have a lot of knowledge about what data is the most useful and interesting.

From the initial requirements specification (2.4) we have a metric to show the correctness of our project, that ranges from 1-3:

1. No features are implemented.
2. Some features are implemented, those that are implemented work.
3. All features are implemented and working.

The amount of evaluations we can perform are limited because of our time constraint of three months. This means we will perform these evaluations during the end of the project. We will make up for the lack of evaluations by having Kyma help us evaluate our project continuously by receiving feedback on our work.

4. Detailed Design

4.1 Use-Cases

After our initial requirements we decided to have a use-case diagram like shown in Figure 4.1. The *system* is the application called "Kyma API App" and it involves 2 main actors, primary actor: *User*, and support actor: *Kyma API*. The primary actor is the one that interacts with the system and the support actor only takes a responsive part in the system and thus cannot directly influence the *system*.

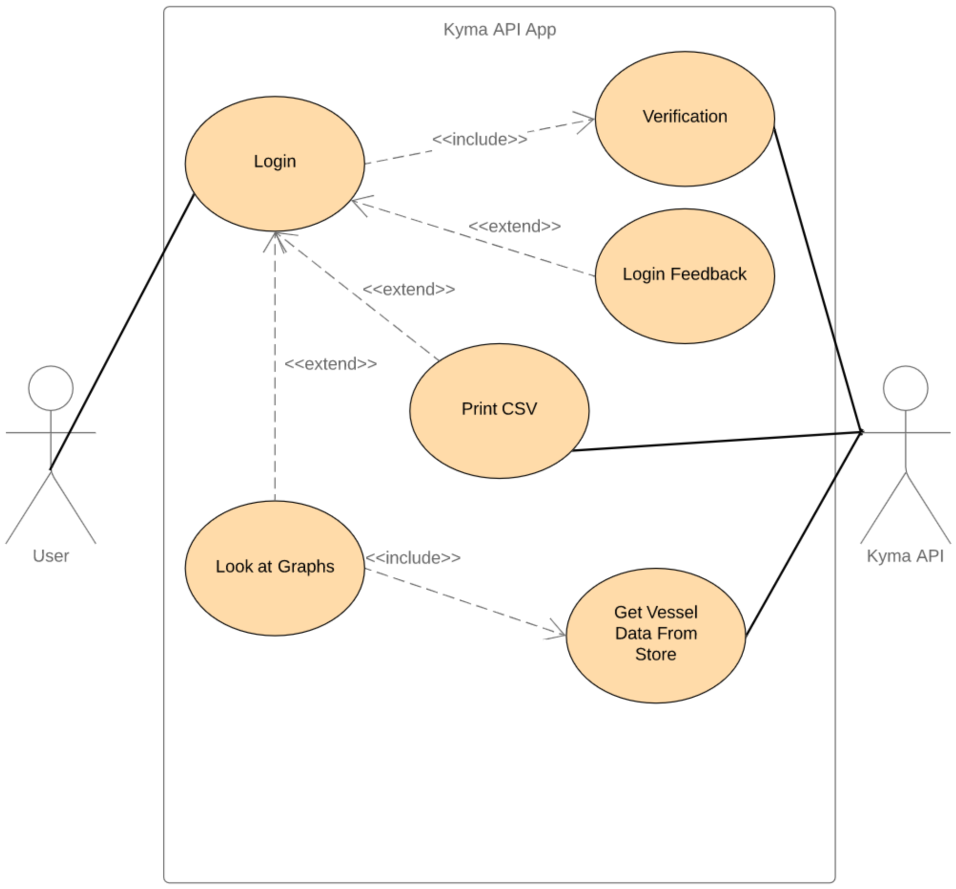


Figure 4.1: Use-case diagram of system Kyma API App

The main entry point of our *system* is through *login* use-case. The *Login* requires another use-case *Verification* that will get in touch with the data center through the *Kyma API*. If however the *login* fails the *Verification, a Login Feedback,* extends *Login* and prints out a suitable message for the user.

After a *successful login* two other use-cases extends *login* to provide a *Print CSV* and *Look at Graphs* use-cases. Through the *inclusion* of *Get Vessel Data From Store* the *User* is able to look at the *graphs* displaying data fetched from the *Kyma API.* The data store includes a predefined subset amount of data that are specified for the application. This showcases the application as a tool to demonstrate the value of data that the API provides.

Unlike the graph use-case, the *Print CSV* use-case functions differently. Rather than getting the data from our store of data we directly fetch it from the Kyma API. There are two reasons for this: first is that *Kyma API* provides a already built in csv data format, but it requires a new fetch of data. Secondly the data that is already fetched is only a small subset of what Kyma stores, and if one wants a specific data type then one has to fetch it from the Kyma API.

4.3.1 Login Sequence Diagram

Figure 4.2 shows the sequence diagram of the login use-case and provides a more detailed view of how this process takes place. As with all systems they must be acted upon by an actor and in our case it is the *user* that starts with the *input* of the user's email and password. The input is then *validated* with a function checking for valid inputs, checking whether the email is the correct format, and if each input field is provided. If this is not the case then an error message will be displayed as feedback to the user.

If all inputs are valid the input will go through an authentication process where the data will be passed to the *Kyma data center* requiring a *200 OK* return message. If the input is not valid then a *401 error* message will be returned instead. Then this will show another error message as feedback for the user. Lastly if all parts are returned with the accepted values the Login.vue will switch the main window to the *Home Dashboard* and we have a successful login.

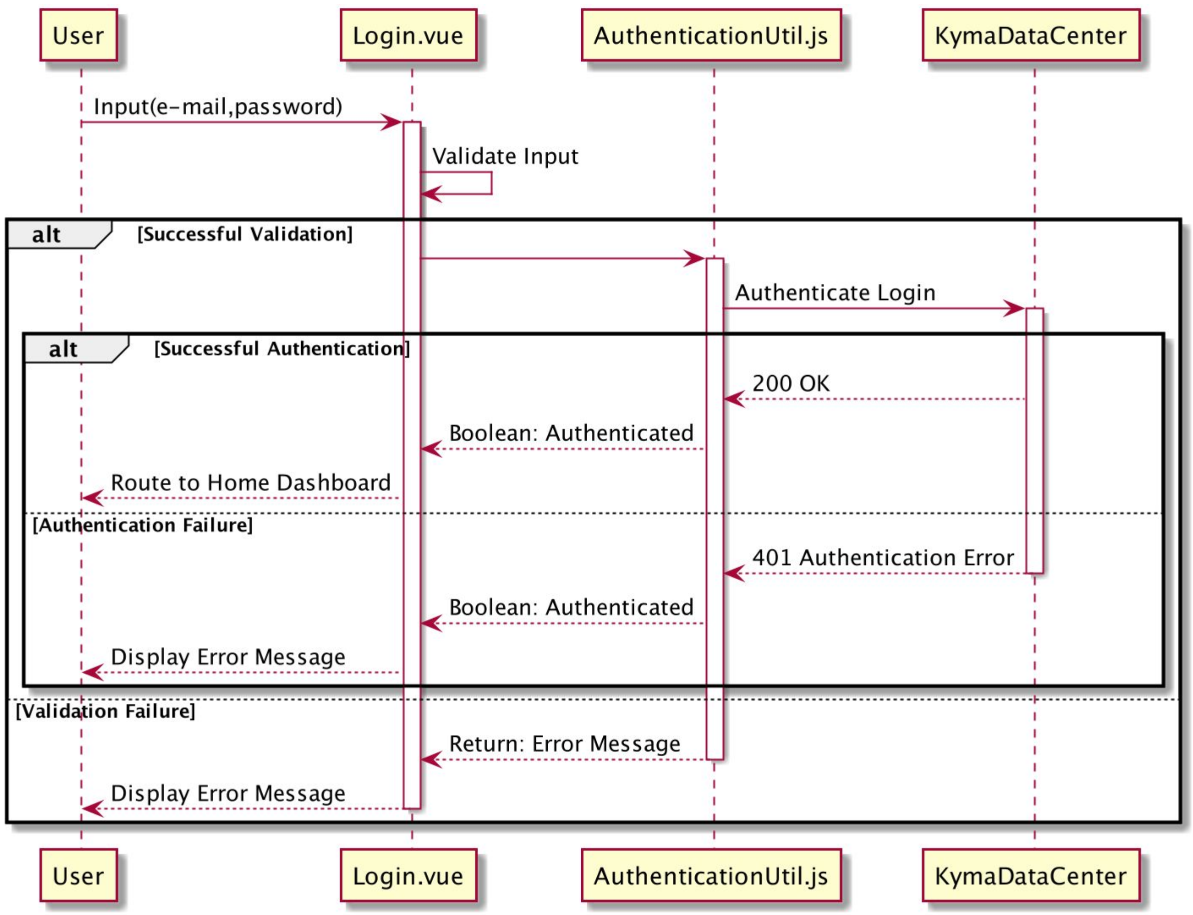


Figure 4.2:Sequence Diagram of Login

4.3.2 Navigation Vessels to Graph Sequence Diagram

Figure 4.3 provides a detailed picture of a user viewing a graph in our application. It shows the use-cases from starting up the application until a *Graph* is shown on the screen. The *Login* as it was shown earlier is now simplified to only show what happens after a successful *login.* The view is passed to the *Router* and routed to the *Vessel.vue* component. This component allows a *User* to click on a list of *vessels.*

When the user clicks on a vessel the corresponding vessel id is passed to the *Router,* the *Router* creates a path with the id and routes to it. The *User* is then presented with the *Vessel.Vue* component which the *Router* has routed to. This is accompanied with the id as an URL parameter so that it can be applied to receive the corresponding correct data for that vessel earlier referenced as dynamic routing. The vessel view will show the correct data for the vessel the *user* has selected.

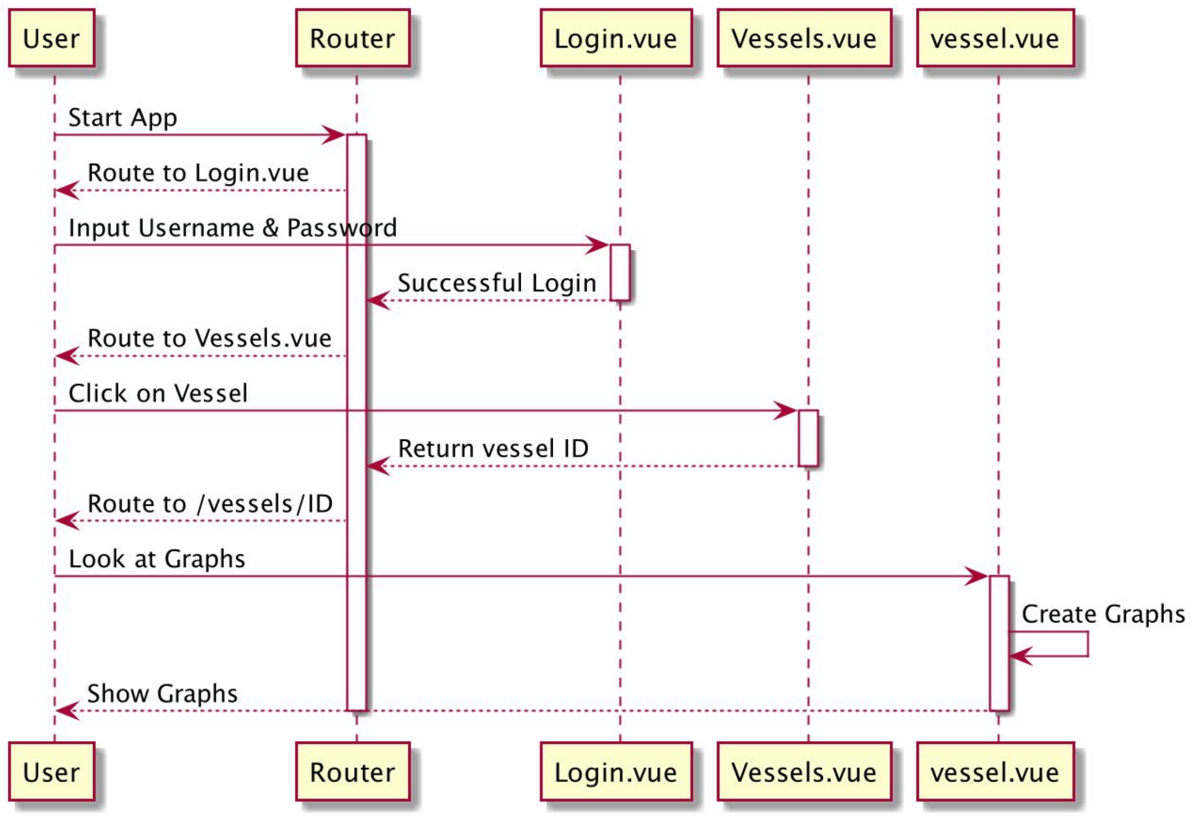


Figure 4.3: Sequence Diagram of Navigation through the Application to viewing a graph

4.3.3 Create CSV file Sequence Diagram

The final use-case of our application, seen in Figure 4.4, will allow the *user* to create a customised CSV file through the use of our *system*. This will take place in the Csv.vue component where the *User* can specify a vessel from the current vessels that are in our data store called Vuex state. The specified vessels will be returned and represented in a *drop down menu* for the *user* to select.

When a *user* has selected a vessel a new get request will be passed onto the data store and will return the *log variables* for this vessel. The user can pick and choose from this list of log variables. After log variables are selected the user can press a button to confirm their choices. The *User* must also pick a granularity. Then the Csv.vue component will send a batch(many variables in one) get request to the *Kyma data center* with a specifier of CSV return and get a reply with the data ready for file output.

The file output needs to be handled on the back-end *main thread* in Node.js because we are working with JavaScript and it can not talk to hardware or file systems without using Node.js. After the file has been created a feedback is returned to the *User.* This is to confirm that a file has been created and to provides the path to where it is located in the filesystem.

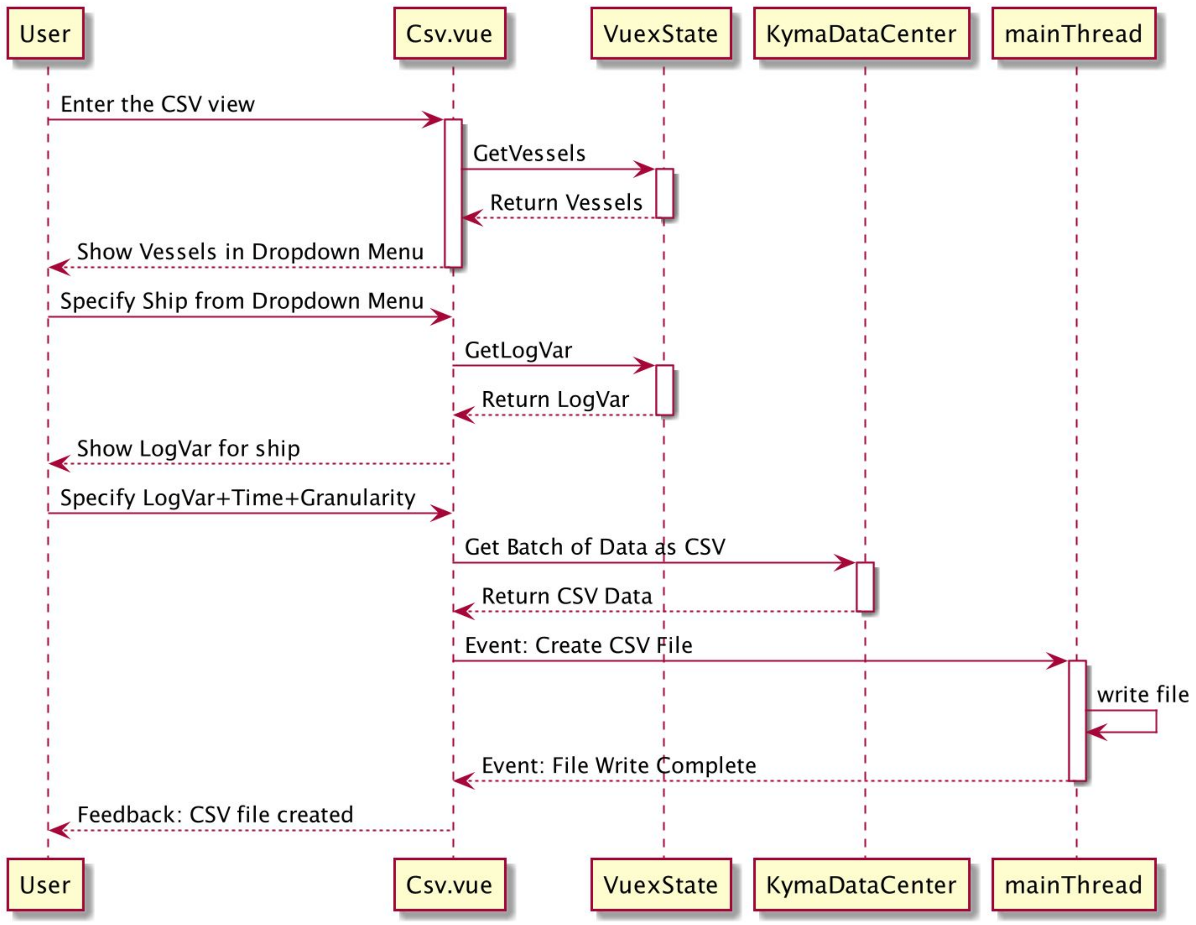


Figure 4.4: Sequence Diagram of creating CSV file

4.2 Architecture

4.2.1 External Architecture

"*REST stands for Representational State Transfer. It's an architectural pattern for creating web services. A RESTful service is one that implements that pattern.*"[6] The Kyma API follows the REST pattern. This architectural pattern is described by Roy Fielding[7] as built on the HTTPS following a number of constraints: Client-Server, Stateless, Cacheable, Uniform Interface, and Layered System. The external architecture of our application consists of a RESTful Server run by Kyma at their data center that takes in data from all the vessels containing their sensors and makes them available to us through their API.[8]

4.1.2 Data Elements

The data format that the API returns is in the form of JSON (JavaScript Object Notation). The actions that are available to us are HTTPS GET requests in the form of three main queries as seen in figure 4.5:

- "/getVessels" provides all vessels with their id and names.
- "/getLogVariables" requires a vessel id and returns the sensor variables for that vessel id.
- "/getLogData" requires a sensor id, granularity and time span as parameters and returns all the data from that sensor in the given time span.

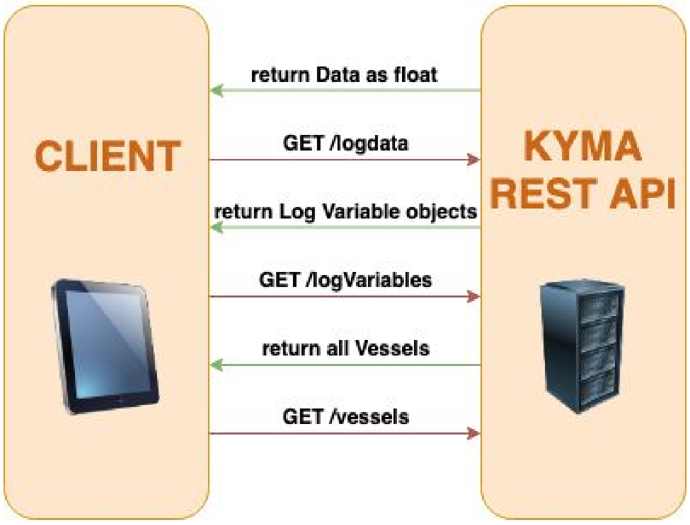


Figure 4.5: Visual Client Kyma communication with HTTPS GET requests. Flow is from bottom to top.

4.2.3 Internal Architecture

The design of our application is composed of the "Model View View Model" (MVVM) design pattern used in each Vue component (see Figure 4.6). This pattern is composed of three parts: a View, a Model, and a View Model communicating in between. This pattern came naturally to our application because we chose the framework Vue.js.

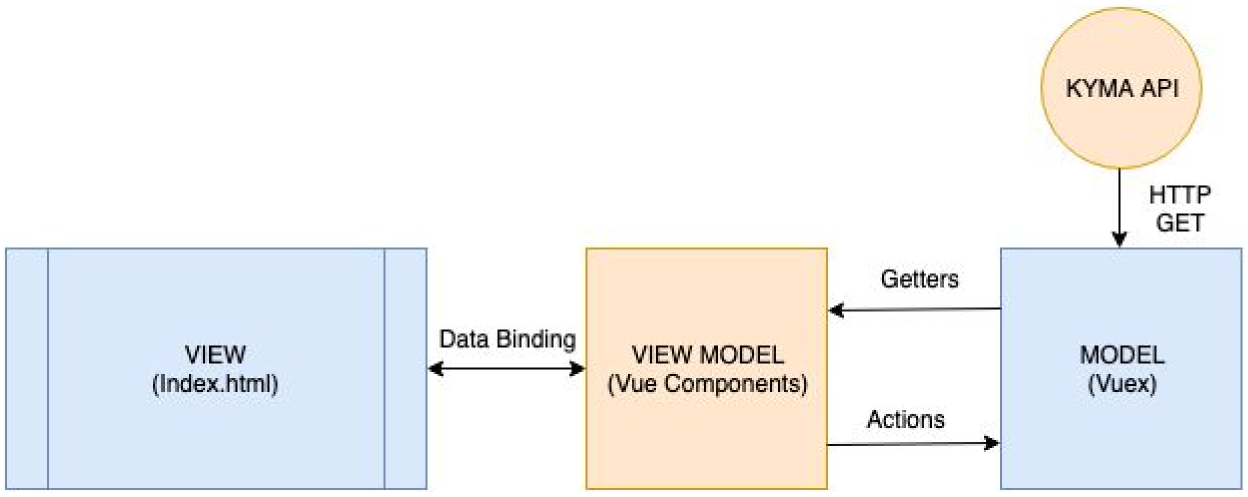


Figure 4.6: MVVM

As seen in the Figure 4.5 we can see our View is composed of DOM elements just like regular websites. Our View Model, as seen in the figure, is the components that we will use for each specific part of our application that will be handling data to be viewed on screen, such as generating graphs. They will be altering the DOM elements in the index.html file.

Our data part or Model, shown in Figure 4.6, of our application consists of a store of data represented using Vuex state management pattern. This will be explained more in depth in chapter 4.3.2. Vuex consists of having data stored as a global state that is available for all components to use. There are layers of watchers both when putting data into state and retrieving them from state so that every part of our application that will use these data will be reactive to changes.

4.3 Modules

4.3.1 Authentication

The authentication part is handled by Kyma's data center. As seen in Figure 4.7 the application sends a regular HTTPS request for JSON data with the email and password the user has typed in, and checks whether a status code 200 is returned. If status code 200 is returned as a response, the user's credentials are valid and the application redirects the user to the home screen of the application.

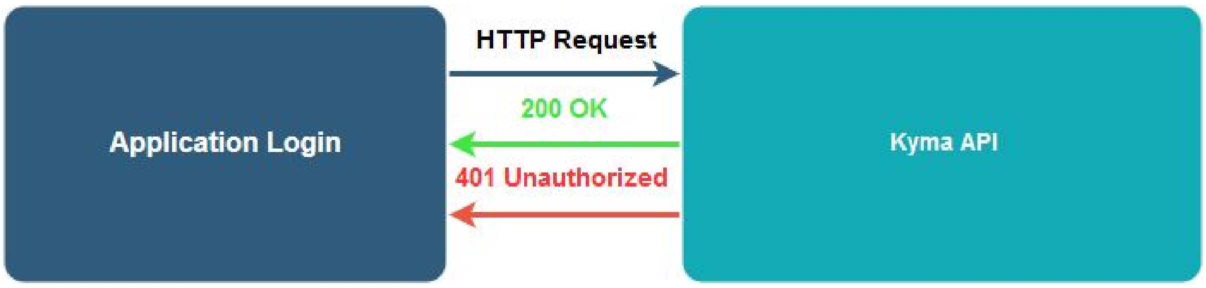


Figure 4.7: Login Authentication returns either 200 OK or 401 Unauthorized

4.3.2 Vuex

Vuex serves as a centralized store for data that components throughout the whole application has access to [9]. It is a source of the *current* 'truth' for the application. Each component in the application has access to this data store. This is why graphs can be located anywhere within the internal structure of the app. All graphs has this store of data available and this data will be up-to-date. Vuex comprises of actions, mutations, state, and getters. Actions and mutations *updates* state. State contains the data that all components in the application can access. Getters provide components with functions to easily retrieve data from state.

Figure 4.8 below shows an overview of the Vuex cycle. Each part is explained more in depth below:

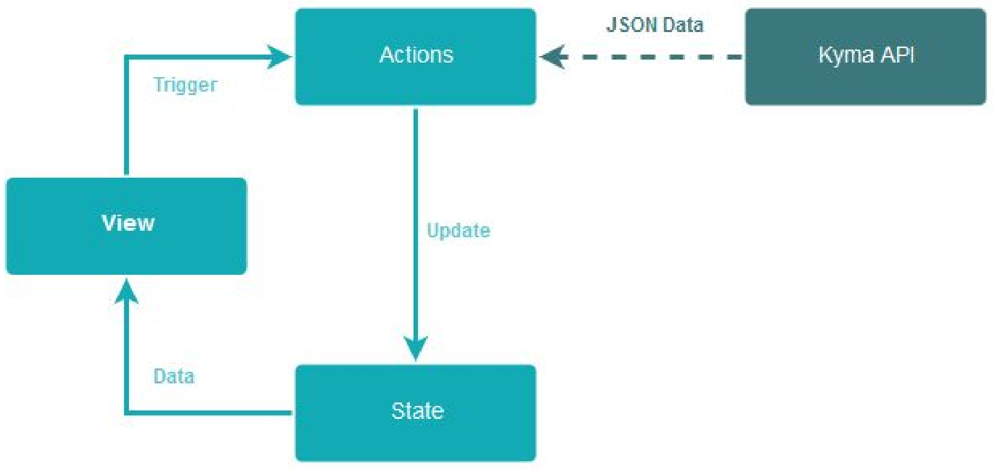
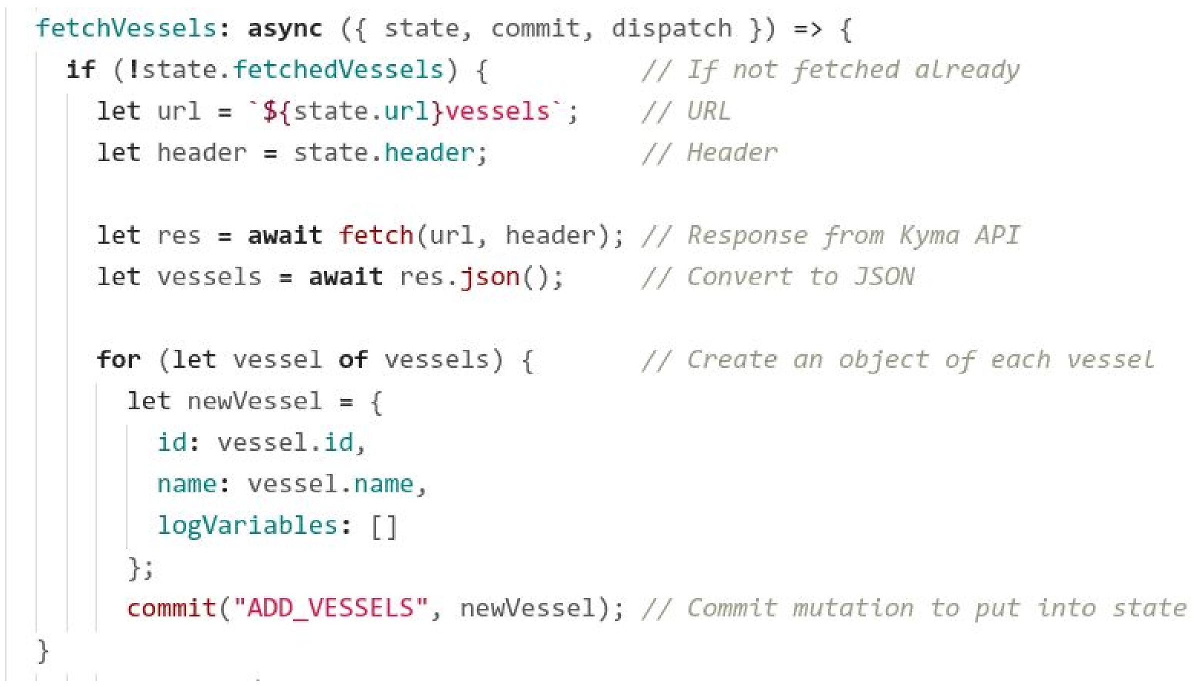


Figure 4.8: How the View uses Actions and States from Vuex store

4.2.3 Actions

Actions contain functions that return data that state should store (see Figure 4.8). This involves updating current state data or fetching new data from an API. Different events within the application will trigger an action, and an action may trigger another action. Upon login, an action is dispatched to fetch data about all vessels associated with the current user credentials. Nested actions are dispatched subsequently to gather all relevant data for the vessels.

Fetching data from the Kyma API is done through custom functions in Actions. These functions take arguments that are concatenated into complete URLs which are then used as API calls (see Figure 4.9).



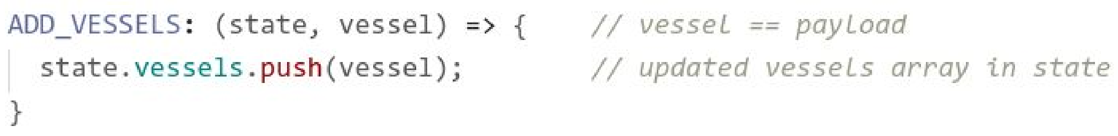
```
fetchVessels: async ({ state, commit, dispatch }) => {
  if (!state.fetchedVessels) {          // If not fetched already
    let url = `${state.url}vessels`;    // URL
    let header = state.header;          // Header

    let res = await fetch(url, header); // Response from Kyma API
    let vessels = await res.json();     // Convert to JSON

    for (let vessel of vessels) {       // Create an object of each vessel
      let newVessel = {
        id: vessel.id,
        name: vessel.name,
        logVariables: []
      };
      commit("ADD_VESSELS", newVessel); // Commit mutation to put into state
}
```

Figure 4.9: Fetch function (Actions)

These API calls return a payload of JSON data. Instead of storing it directly to state the data is committed and sent as a payload to Mutations (see Figure above).



```
ADD_VESSELS: (state, vessel) => {    // vessel == payload
  state.vessels.push(vessel);        // updated vessels array in state
}
```

Figure 4.10: Update state variable with payload (Mutations)

Mutations contain functions that take data as arguments (see Figure 4.10), and store them in state. Mutations serve as an interface between Actions and state to update state in a predictable manner.

4.3.4 State

The application stores vessel data in a global state (see Figure 4.8) after an initial fetching process upon login. This makes it easier to populate graphs in different vessel dashboards with data. Instead of passing data downward into nested child components we have a global singleton, a shared global state of data. This structure is less complex and easier to maintain. No matter how nested a component might be it doesn't rely on the parent component passing the correct data to it, and the parent component doesn't need to manage a lot of data to supply to its child components.

4.3.5 Views

Views are the pages the user can see (see Figure 4.8). Under the hood it's all just one page that can change drastically based on the user's interaction. Views are composed of components. A vessel dashboard (view) consists of multiple graphs (components). Every component is reusable for different vessel dashboards. The only difference is that the data changes to match the current vessel.

4.3.6 Dynamic Routing

All pages, or views, in the application correspond to a path, much like a regular web page URL. The first page upon login is an example of this, which shows a selection of available vessels. This part of the application corresponds to the path '/vessels'. Dynamic routing allows different paths to point to the same route, and object attributes to be part of a path [10]. After clicking on one of the available vessels the new path will look like this 'vessels/1'. The last parameter in the path corresponds to the vessel's id attribute. This makes for a generic vessel dashboard where the current vessel can be identified using the path. No matter which vessel you click on you end up at the same route. However, by reading the route parameter we find which vessel we are currently viewing. This is important, because the id of the vessel can be used to retrieve the correct data and display it in graph components. The vessel dashboards are in actuality just one dashboard component that knows how to retrieve the correct data for each vessel.

Figure 4.11 illustrates how each path routes to the same vessel dashboard component (this is a route), and how the vessel id determines which data will be displayed in that component.

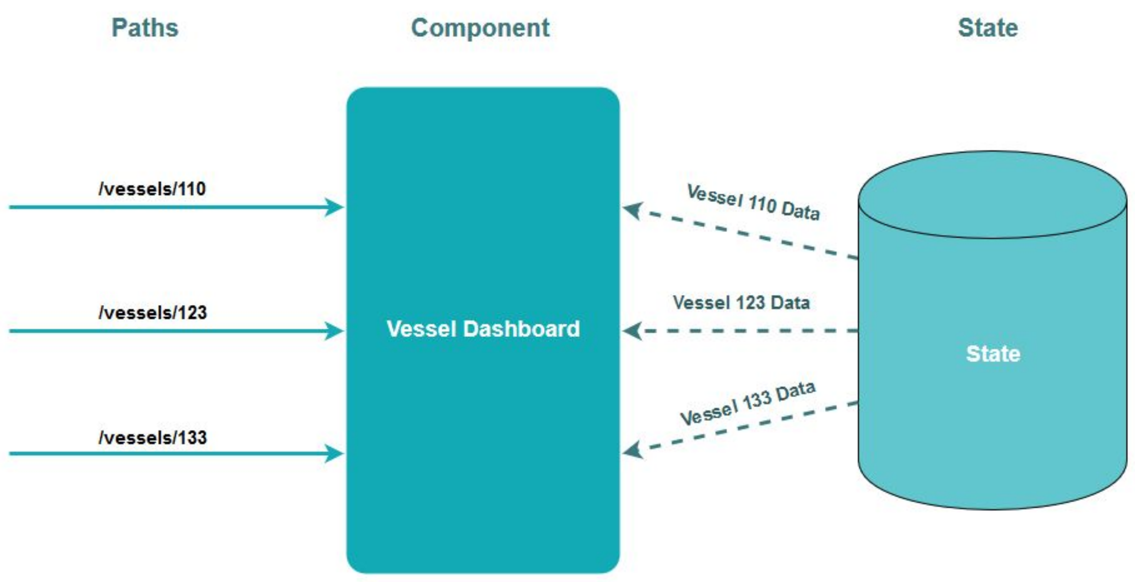
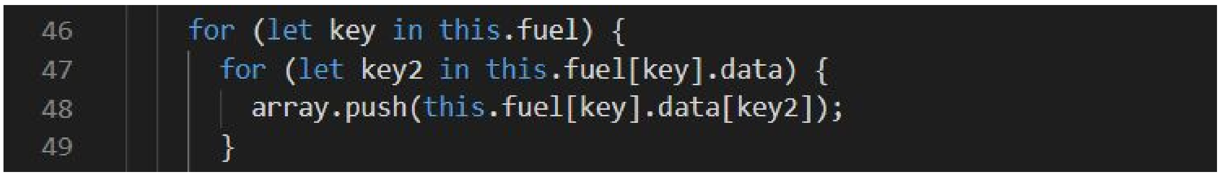


Figure 4.11: Dynamic Routing

4.3.7 Graphs

This chapter will explain in more detail how the graphs get the data it needs from the JSON objects in state and renders the graphs. For reference some additional example graphs can be viewed in the section 4.5 figures 4.21 and 4.22.

Graphs are individual components consisting of different types including line charts, bar charts and pie charts. To get the information needed to make the graphs, the components uses methods from the getters component to gain access to the JSON formatted data that is stored in Vuex state. Graphs iterate through the JSON objects and pushes the necessary data into arrays. It does this for all the data values that are going to be drawn, creating several different arrays.



```
for (let key in this.fuel) {
  for (let key2 in this.fuel[key].data) {
    array.push(this.fuel[key].data[key2]);
  }
```

Figure 4.12: Iterates through a fuel object in a nested for-loop and creates arrays of data points to be plotted.

These arrays now contain the data points for our graphs. The arrays are processed differently for each graph. In the line chart each value in the array are simply plotted and a line is drawn between the value points. For bar and pie charts the arrays are summed up, and the sum of each array is then displayed as a total value. This sum can be the total fuel consumption in a given time. Each bar on a bar chart can represent different fuel types or fuel burners, such as generators or boilers (see Figure 4.13).

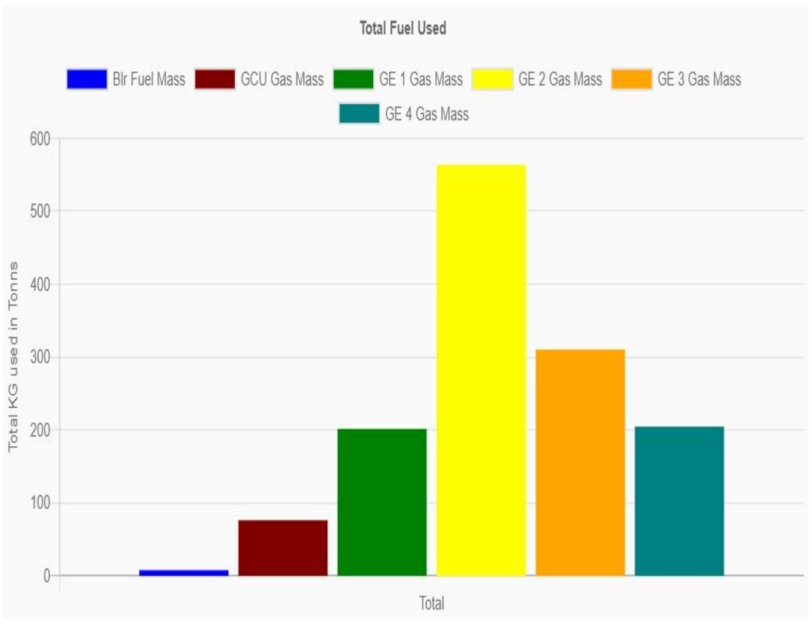
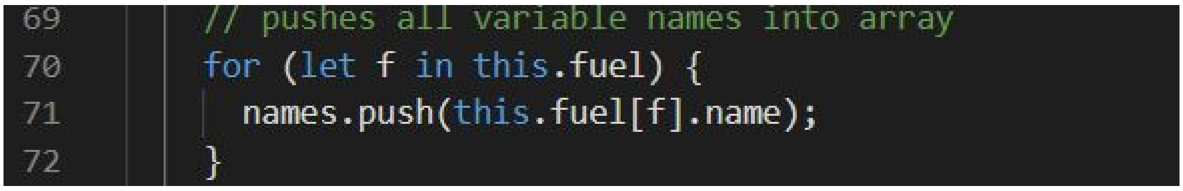


Figure 4.13: Bar chart showing total fuel over a month with variable names on top.

The graphs also contain names of the variables. The names are stored in JSON objects and are retrieved via a getter and pushed into an array. The same way as the data points, only difference being that all the different names are pushed into the same array. The names are displayed on top of the graphs (see Figure 4.13).



```
// pushes all variable names into array
for (let f in this.fuel) {
  names.push(this.fuel[f].name);
}
```

Figure 4.14: Fills an array with variable names

Some of the graphs do not need multiple labels along the x-axis since they only show a sum of data points in a given time frame. However, some of the graphs has time labels along the x-axis. For instance, the line chart shows the speed by time.

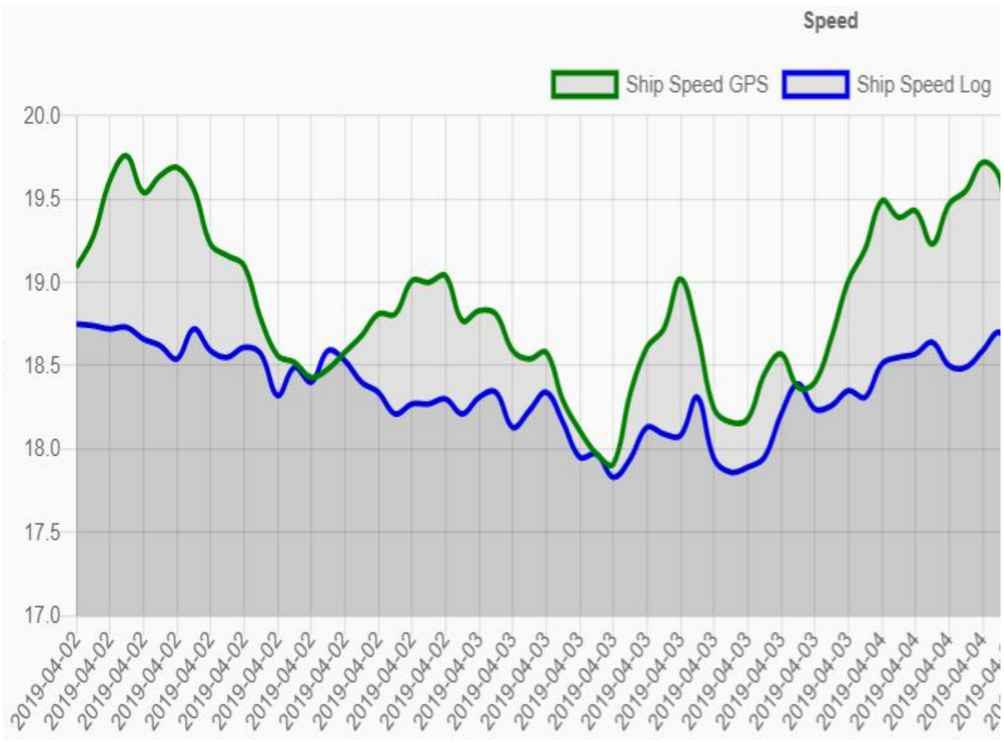
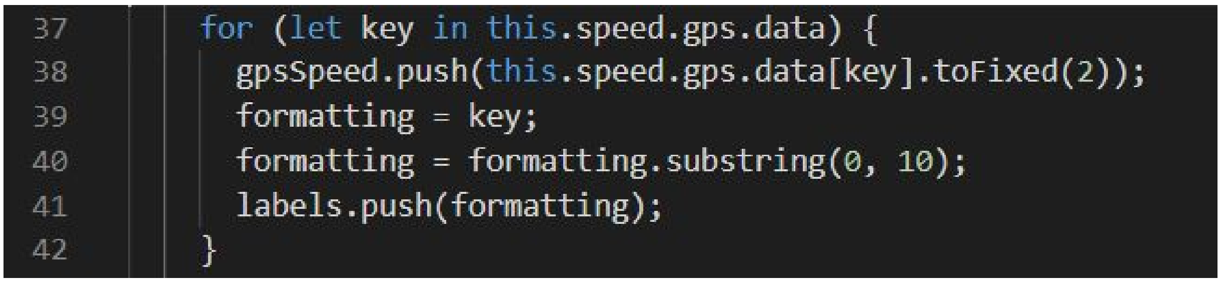


Figure 4.15: Line chart showing the speed of a vessel with dates along the x-axis.

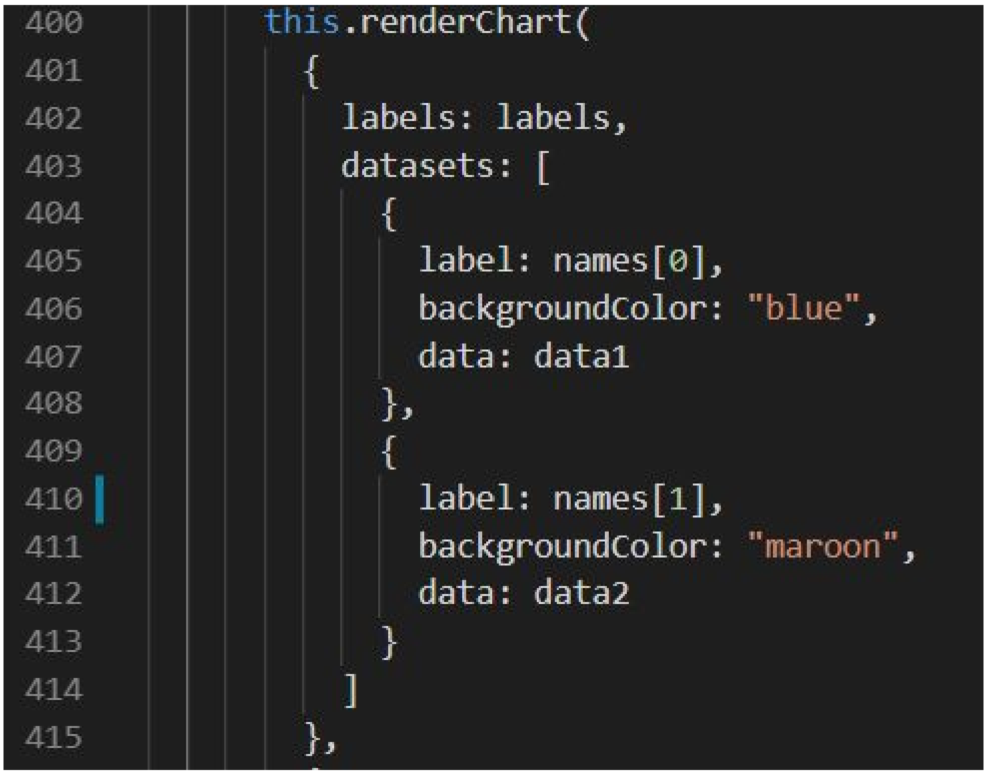
This function iterates through a gps JSON object. Each data point is stored in the gpsSpeed array, and each time stamp, which is the key of each data point, is stored in the labels array.The array containing the time labels will be the x-axis.



```
for (let key in this.speed.gps.data) {
  gpsSpeed.push(this.speed.gps.data[key].toFixed(2));
  formatting = key;
  formatting = formatting.substring(0, 10);
  labels.push(formatting);
}
```

Figure 4.16: Function for getting gps data and labels.

The renderChart (see Figure 4.17) function which render the graphs takes an object as an argument. This object should contain all the necessary variables for the chart to render properly. This includes the data array, label array, and the names of the variables. The data array is plotted on the y-axis. The label array labels the x-axis with the data points' corresponding time stamps. Lastly, the array of names shows which variables the chart is currently displaying. The object contains other options such as coloring for each variable.



```
            this.renderChart(
              {
                labels: labels,
                datasets: [
                  {
                    label: names[0],
                    backgroundColor: "blue",
                    data: data1
                  },
                  {
                    label: names[1],
                    backgroundColor: "maroon",
                    data: data2
                  }
                ]
              },
```

Figure 4.17: Showing renderChart object, including options for coloring

4.4 User Interface Design

Creating a brand new and intuitive UI and UX takes time and knowledge about the subject, and these types of resources are lacking in the project group, as stated before in Chapter 1.3. Therefore, we require an easy and fast method of implementing a good UI and UX. To do this, we looked at Google's material design.

Material design contains a set of principles, guidelines and tools for creating good design. These principles and guidelines are implemented in many applications and devices that everyone uses daily, such as Google's mobile applications and websites like YouTube and Slack. Utilizing material design will create an intuitive user experience for our application thus ensuring a good UX. For instance, the icon at the top of our navigation drawer is called a hamburger icon (see Figure 4.18), which can be seen in almost any application today. This icon is used to indicate that this is a menu and the user can click on it to reveal the menu or minimize it. We also made all the clickable components highlighted when a user hovers the mouse over them, indicating that these components can be interacted with.

We have chosen to implement material design in our application, by using the Vuetify material design component framework for our UI design. Vuetify provides us with all the graphical components we need to make our design and it is made with Vue so it fits our application well[12]. The Vuetify graphical components are also visually pretty in our opinion, thus very little modifications on these components are needed. This means we can implement a good UI in a time efficient manner. It also helps us in regards to paging and scalability [11].

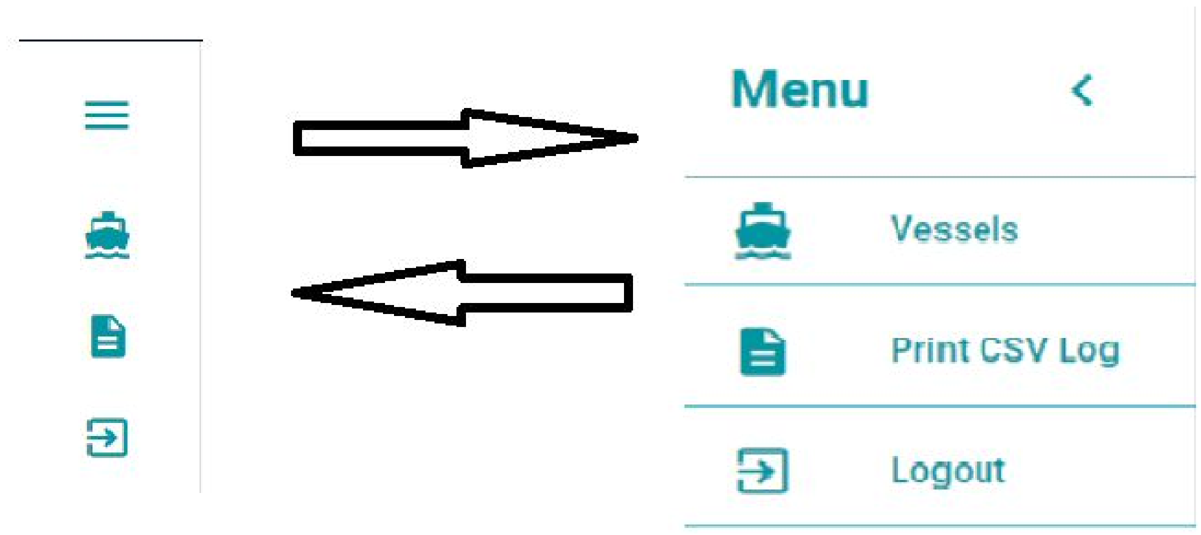


Figure 4.18: Showing the navigation drawer minimized on the left and the navigation drawer extended on the right.

4.5 The Application Workflow

Here we show how the application works, including what the user can do, and the features that the user can interact with.

4.5.1 Login

When the user opens up the application, he/she is greeted by the login screen. The user must be authenticated with a user id and a password. These user credentials are provided by Kyma, where the user id is the username (email account) for the Kyma Online portal. Login happens when the login button is clicked or the enter key is pressed (see Figure 4.19).

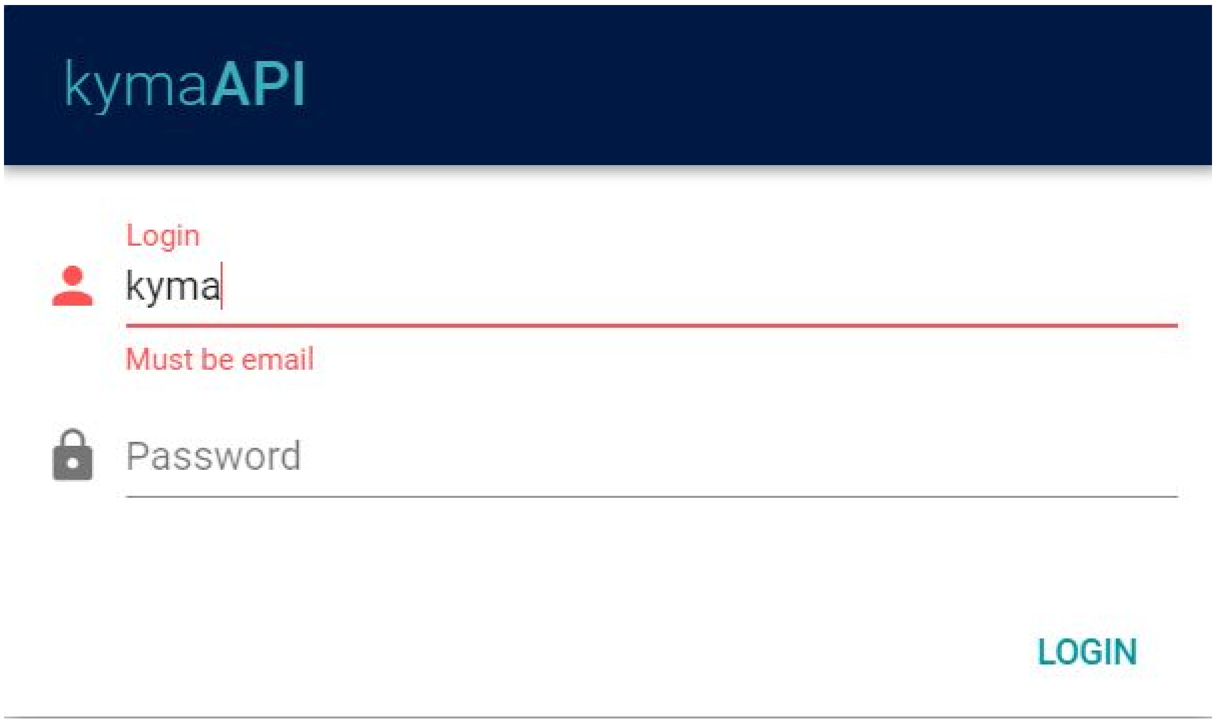
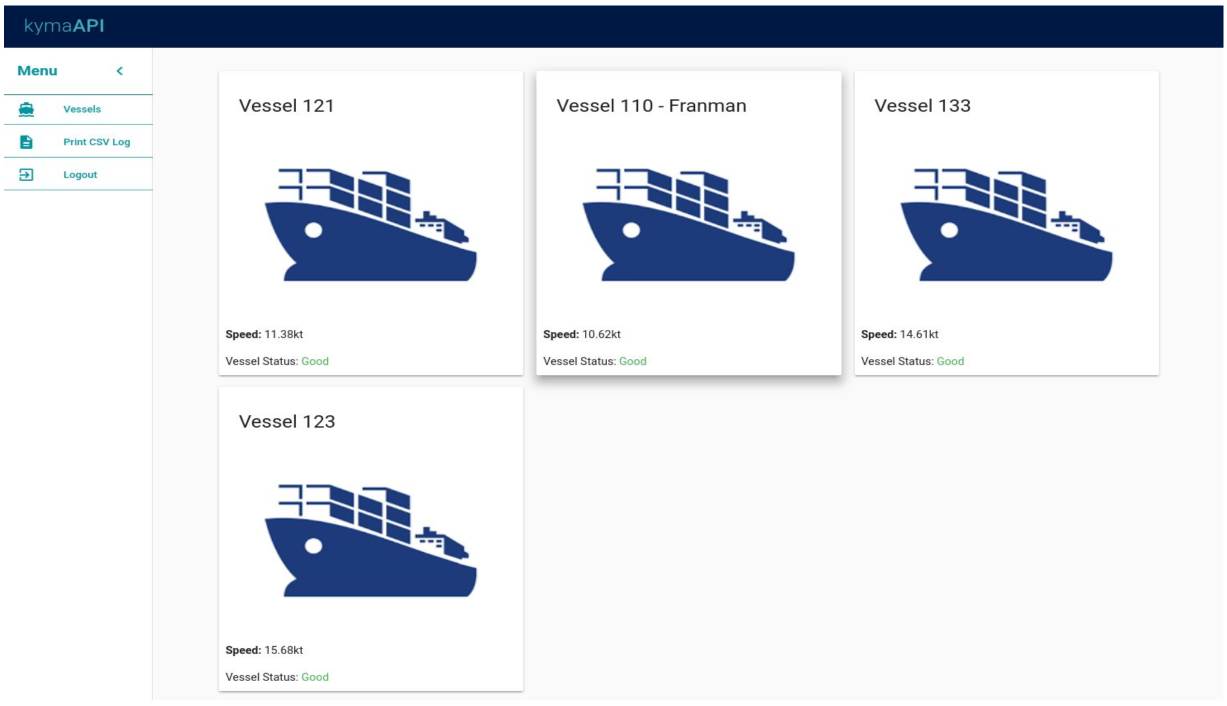


Figure 4.19: The login page

4.5.2 The Home Screen

When the user has logged in successfully they get shown the home screen (figure 4.20) which contains a grid-list of their vessels. From here he/she can navigate to a specific vessel or see current speed about his/her vessels within the vessel cards. The navigation drawer is on the left side of the screen, with three buttons: one to go back to the list of all vessels, one to a page where the user can print out a CSV file of the data, and one to log out of the application. The navigation drawer is visible anywhere on the application, except the login screen, and can be resized by clicking the top icon in the navigation drawer.



Vessel 121
Speed: 11.38kt
Vessel Status: Good

Vessel 110 - Franman
Speed: 10.62kt
Vessel Status: Good

Vessel 133
Speed: 14.61kt
Vessel Status: Good

Vessel 123
Speed: 15.68kt
Vessel Status: Good

Figure 4.20: The home screen

4.5.3 Inside the Vessel

When the user navigates to a specific vessel, they get the page for that specific vessel and a default graph is shown. From here he/she can choose a graph with a drop down menu. The time frame is chosen via the date picker and it specifies the date for all graphs (See Figure 4.23). The graph will then be visible with the specified time frame. If the user wants to hide a variable from the graph, they can click the name of that variable at the top of the graph. By clicking on the *Vessels* button in the navigation drawer the user can go back to the home screen, or do so by clicking the *kymaAPI* logo on the top left corner (See Figure 4.21 and Figure 4.22).

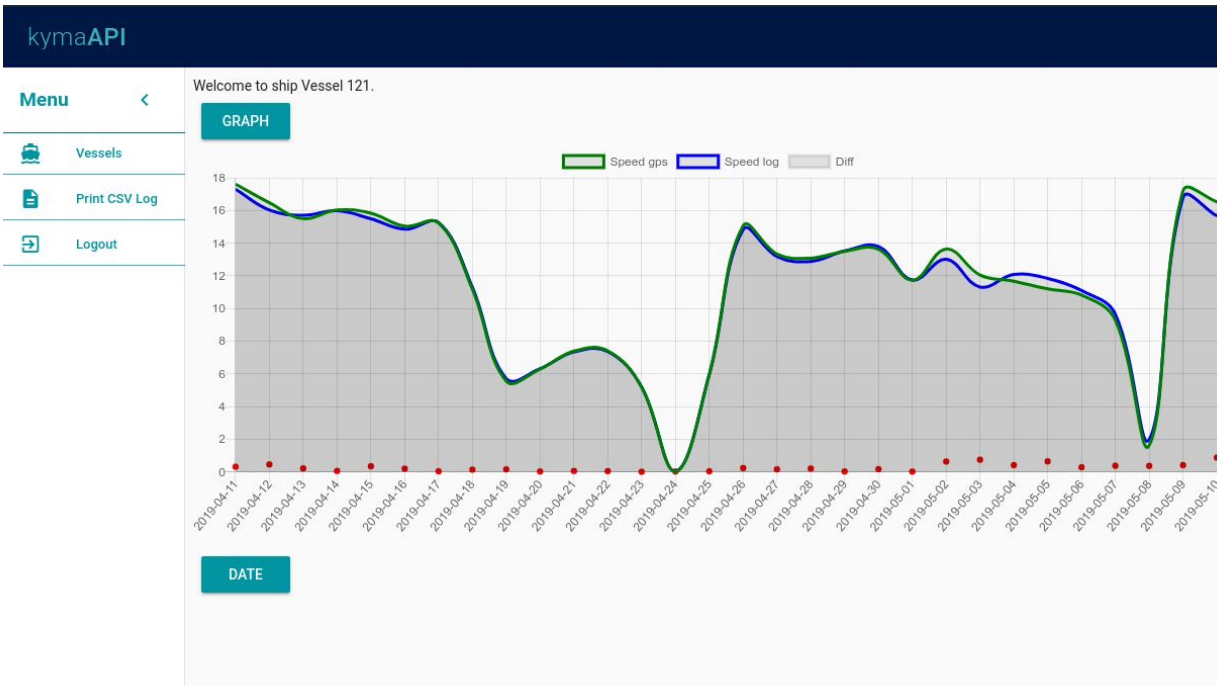


Figure 4.21: Inside a vessel, line chart that shows the speed of the vessel.

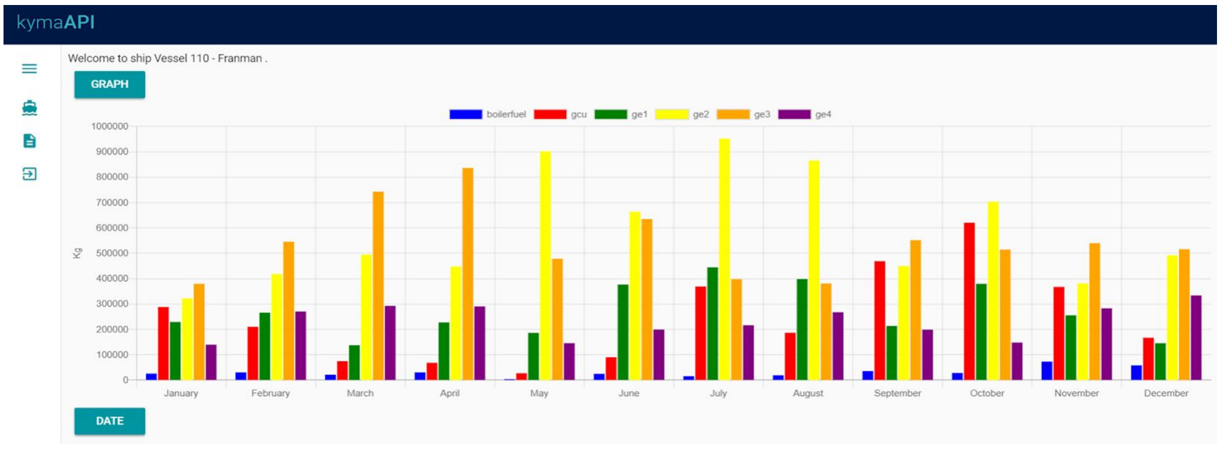
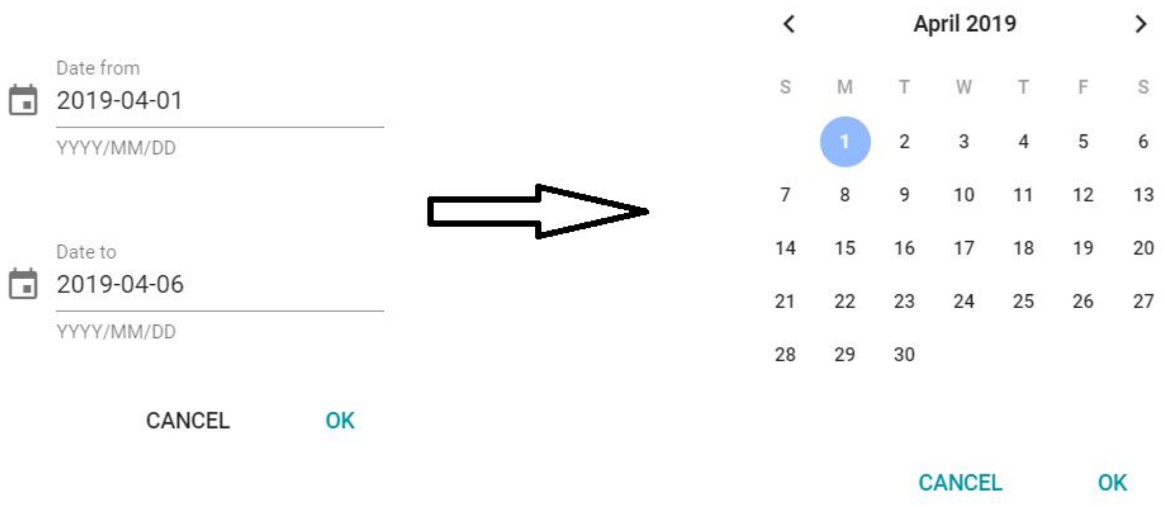


Figure 4.22: Inside a vessel, bar chart that shows fuel consumption.



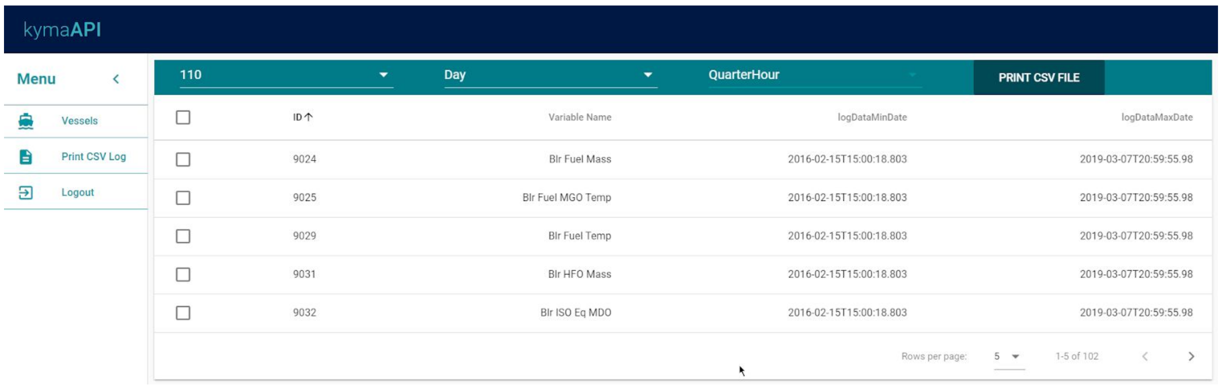
Date from 2019-04-01 YYYY/MM/DD

Date to 2019-04-06 YYYY/MM/DD

Figure 4.23: The Date picker. When the input field is clicked on the left picture the calendar on the right picture is shown.

4.5.4 Print CSV

If the user clicks on *Print CSV Log* he/she is presented with a page where he/she can download a CSV file of the data. The user can choose for which vessel they would like to download data, the granularity and timeframe. The user chooses what log variables he/she wants by clicking the checkboxes to the left, and by clicking the top checkbox they can select all log variables. On the bottom right there are three buttons: the leftmost one enables the user to choose the amount of log variables to show on the page, and the two arrow buttons to the right allows the user to go to the next page or previous page of log variables (see Figure 4.24).



	ID ↑	Variable Name	logDataMinDate	logDataMaxDate
☐	9024	Blr Fuel Mass	2016-02-15T15:00:18.803	2019-03-07T20:59:55.98
☐	9025	Blr Fuel MGO Temp	2016-02-15T15:00:18.803	2019-03-07T20:59:55.98
☐	9029	Blr Fuel Temp	2016-02-15T15:00:18.803	2019-03-07T20:59:55.98
☐	9031	Blr HFO Mass	2016-02-15T15:00:18.803	2019-03-07T20:59:55.98
☐	9032	Blr ISO Eq MDO	2016-02-15T15:00:18.803	2019-03-07T20:59:55.98

Rows per page: 5 1-5 of 102

Figure 4.24: The download CSV page

5. Evaluation

5.1 Evaluation Method

Our intended method of evaluation is an approach where we receive continuous feedback from our project employer and the vessel owners. We wanted to receive the comments after each development sprint and after project completion. The feedback we received would be valuable evaluation data we could use to implement continuous changes. The goal of the evaluation is to use the evaluation data to add improvements and fix issues during the development lifecycle. This would help us develop the right product for Kyma.

We evaluated the product less thoroughly than initially intended. Our method included continuous feedback from our project employer. We had a formal evaluation near the completion of the project where we had an application showcase for Kyma. This is where we got our evaluation data.

On the Github repository, where we store all our project code, we utilized sprints so that our project employer could follow along with our development and give us feedback when we updated the project. Our project employer also provided github issues with features they wished to have implemented in the application. In addition, we had good communication with our project employer in our office space at Kyma, where we could show our application during development when our project employer had time to see it.

5.2 Evaluation Results

Below are the requirement scores we have given the application adhering to the initial requirements in Chapter 3.5.

Requirements:	**Score:**
The user can login to the application with an API-key and password:	2
The user can use a GUI to choose what data to extract about their available vessels:	2
Run on a Surface Pro:	3
FIlter and only show the most useful data:	3
Retrieve data and print out CSV file:	3

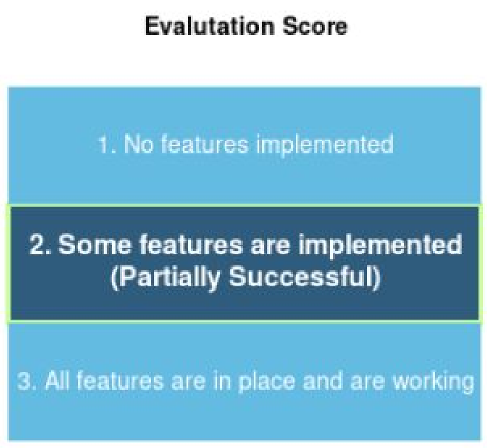

Evalutation Score
1. No features implemented
2. Some features are implemented (Partially Successful)
3. All features are in place and are working

Figure 5.1: Results from our initial specification and evaluation score for the application

6. Results

6.1 Results According to the Project Owners

The project employer, is our main source for evaluating this project. During a showcase in the later stages of the project the product was shown and was well received. We received some feedback for initial requirements that were not met and possible improvements to implemented features.

The positive feedback the product received was praise for us being able to get a working product in a short time. We were able to achieve this because of our choice of frameworks. The frameworks were specifically chosen, despite the lack of experience in it, for the purpose of getting a working product quicker.

Furthermore we received compliments on colorscheme and general design, though that had room for improvements. Specifically on the homescreen we had cards of each vessel with information of current time specific data such as speed, but this currently only works for the demo environment. Also some of their customers had up to 70-140 vessels, with that layout it would not be a good design for the user. It was suggested that a list could be chosen when a user had a lot of vessels. Another improvement would be to sort vessels into categories based different values.

Our application only works for the demo environment. Initially Kyma would have liked the application to work for several environments. A user should have been able to login with a specified API-key and view their vessels and data from the Kyma data center, but we did not succeed in this because we developed it specifically for the demo environment. Why we developed it specifically for the demo environment will be discussed more in depth in Chapter 7.

6.2 Results According to the Project Group

Considering the main goal for the project and the requirements for the application, we feel that we have created a solution that is partially successful. Our solution is a dashboard application designed to work with Kyma's demo environment. We feel this is a good enough solution for Kyma to show the value of their API and data it provides to the vessel owners. The dashboard does not work with other API keys than the demo environment. However, some parts of the application are more dynamic than others. The Print CSV component can work with other API keys with little change to the code. However, the core application will need a lot of rewriting to work with other environments.

The application contains components to visualize the incoming vessel data from the API. The user can select which vessel to view data for, and specify data in between two dates. The application fetches a subset of the available data from the Kyma data center. This is ideal for the application to run on a Microsoft Surface because of a small amount of RAM in comparison to more powerful laptops. However, the fetched variables are restricted to a predefined default set of variables picked by the project group instead of the user being able to pick themselves. Through discussions with Kyma we found that this was a good solution, because it does not prevent them from showing the owners or clients the value of the API and the data it gives access to. This makes the application less flexible, but makes it run smoother.

7. Discussion

7.1 Hybrid

Our reason for going with a hybrid approach, instead of a native or web approach, was rooted in the fact that the development process was limited by time constraints. Hybrid applications run inside a native container, and uses a browser engine to render HTML and process JavaScript. Hybrid applications are generally faster to develop than their native counterparts. A fast and very efficient application wasn't part of the core requirements which also strengthened the choice of a hybrid application. Speed and efficiency of the application is often the tradeoff one has to pay for choosing a hybrid approach.

The team didn't have any experience in developing hybrid applications prior to this project. This led to the initial phase consisting of a lot of reading and doing research. We managed to develop and add all the minimum features (login, graphs to visualize json data, csv download option) in the time frame we were given, and felt we spent a good amount of time producing meaningful code. In conclusion, we agree that this approach was the best approach considering the alternatives and the requirements for the application.

7.2 JavaScript

The project group picked JavaScript as the best choice for developing this dashboard application. This went well with the data that is fetched from a REST-API. The data is in JSON format which is a language-independent data format, but derived from JavaScript. JavaScript therefore has a ton of support for working with and manipulating JSON data. C++, C#, and Python were also discussed considering no one in the project group had any experience with JavaScript. However, JavaScript has a lot of frameworks that we took advantage of to launch the development process and propel the project forward quickly.

7.3 Vue.js

Vue is a framework that makes the code more structured and maintainable. It divides pieces of the program into logical reusable components. This leads to a more organized coding environment that is easier to conceptualize and work with. The use of frameworks like these is the reason JavaScript is a good choice even without a lot of experience in the language.

The Vue philosophy is to contain lean and reusable components. We started out following this development philosophy, but as time constraints grew our focus shifted from following this philosophy to just making features work. As a consequence of this our code got bloated. This caused our code to become harder to maintain, have lower reusability, and scale poorly. If we had more time we would refactor large parts of our code and rewrite functions to have one purpose and one purpose only.

7.4 Electron.js

Electron was the first framework we decided to use. It allowed us to transform the application from a page hosted in the browser to its own desktop application . It took a good amount of time in the starting phase of the project to learn this framework, even though we did not use a lot of its functionality.

7.5 Scrum

We chose an agile development method that is iterative and incremental. The two methodologies we considered were Rational Unified Process (RUP) and Scrum. We ended up with Scrum because of its flexibility and its short sprints. Scrum allowed us to narrow the planning down to a week by week basis. With the team's lack of experience we found it hard to establish a detailed long term plan, and with Scrum's short sprints we had more opportunities to readjust the course of the project.

Scrum gave us the flexibility we needed, but over time things weren't planned ahead as much. The team would have benefited more from a stricter plan, or with a planning strategy with less improvisation involved. The sprints weren't followed as rigorously as they could've been. This could have been fixed by appointing a project leader. The project leader should have delegated tasks and held other members accountable for the task and deadlines they received. This could have lead to the development methodology being enforced more directly and we could have planned ahead more and managed our time more efficiently.

7.6 Data Visualization

From the beginning we were really impressed with the D3.js library and its data visualization capabilities. It seemed to be able to provide all the functionality we wanted to visualize data and a lot more. After sticking to it for a while we noticed how little we accomplished over a fair amount of time. The learning curve was a bit steeper than we expected. To further complicate things, D3 bind data to the Document Object Model (DOM), while at the same time Vue.js uses its own virtual DOM. This complicated things to a point where we agreed we had to look for options.

Chart.js became our data visualization library of choice. It does provide less visualization capabilities, but still way more than we needed. The effects were immediate and suddenly we had some graphs that we were happy with.

7.7 Hard-coding and Dynamic Coding

As programmers we always like to develop code that is dynamic. Dynamic solutions can adapt to changing input without the need for a developer to step in. Kyma wanted a demo application for their demo environment, but was also interested in a solution which could be deployed outside of the demo environment. To achieve this one would have to keep hard-coding to a bare minimum. We started off the project with the intention of producing a dynamic solution that could adapt to and load data from any API key. By developing dynamic interfaces the developer doesn't have to change the code when the input or "environment" changes.

Throughout the development process the ideas and solutions we came up with were all rooted in a dynamic dashboard application. However, as time went on we had to compromise between developing a dashboard that works for other environments outside Kyma's demo environment, or create a better application for the demo environment only. Considering a demo application for the demo environment was the main task, we went ahead and made that our main focus.

To make the dashboard application more dynamic we would have to spend more time in developing dynamic interfaces within the application to handle the different amounts and types of log-variables, and a changing number of vessels. This involves staying within the limit of acceptable RAM use, and having graphs respond correctly to missing variables, as well as extra variables. This is something we could have spent more time hashing out next time early in the project.

7.8 MVVM Design Pattern

There were struggles in getting used to the framework and ways of working with it that we had decided. At first the group had little idea of how the language worked and how things would come together, thus it took a while to get a understanding of the architecture that we would choose. We decided on MVVM because of its easy integration with our chosen framework, Vue. This in general worked well to serve our purpose.

7.8.1 Model

We had a bit of trial and error with how we came to the final iteration of how the model was applied. First we tried to keep separate JavaScript classes being our model. One JavaScript class had control of retrieving the data from the data center, while another JavaScript class had the role of being our data structure in the form of JSON.

We implemented this partially where we had a separate file for storing the API-keys in JSON temporarily to avoid sharing it online during our development. The file getData.js, had responsibility of all our API-fetches. Before we got to the point of creating a fully fletched data structure for the input data, we looked into a more formal data structure framework. We had two possibilities in mind, SQL or a Vuex state store. We looked into SQL with sqlite, but decided against it because of having to perform another step of converting the data from JSON to sqlite. So finally our choice of model fell on Vuex.

7.8.2 View Model

Our view model was made up of Vue components. The components are made up of 3 parts: a template of html for the static elements, a script part of JavaScript for the dynamic elements and data handling, and a style for the css styling of the components. We used mainly the template and script part of the vue components and let Vuetify take care of the styling for us. To create a component went generally very well for the whole team, however to use them correctly to their real purpose was harder to implement. A Vue component are made to be small and to perform well on one task and that task only. If a task is repeated that part is supposed to be refactored out to create more specialised components. After development had been going for a while the project started to have a lot of duplicate code and got bloated. As such it became hard to maintain. In generality we kept the overall arch of the architecture.

8. Conclusion

The goal of this project was to make an application that will demonstrate how one can use the Kyma API so that everyone involved in the industry can understand the value of it and the data it provides, even if they have no computer background.

8.1 Judgement

We feel this goal has been achieved even though the application does not fulfill all the initial requirements completely. We think a person with no computer background will through the use of this application be able to see the value of the Kyma API and the data it gives access to. Therefore we have concluded that we have successfully made a tool that Kyma can use.

8.2 Further use

Kyma stated that they could potentially use parts of the application or code in another solution they're developing. Kyma can also use this application to show to clients or vessel owners.

8.3 Further development

If this project were to be developed further the main focus should be making the application more reactive and dynamic. Kyma wanted the application to work for environments other than the demo environment. This would require an overhaul to some of the core aspects of the code. The code would need to be refactored to make it more scalable and maintainable.

References

[1]Marshipengineering. (2019). [online] Available at: https://www.marshipengineering.com/products/ship-performance-monitoring/kyma-as/kyma-ship-performance-monitoring-and-tools/ [Accessed 08 Apr. 2019]

[2]Kyma.blob.core.windows.net. (2019). [online] Available at: https://kyma.blob.core.windows.net/public/downloads/Kyma%20Online%20%20API%20-%20brochure.pdf [Accessed 30 May 2019].

[3]Zhang, L. (2017). *The image that gave us the idea for our solution*. [image] Available at: https://github.com/njleonzhang/element-ui-pro [Accessed 18 Mar. 2019].

[4]Scrum.org. (2019). What is Scrum?. [online] Available at: https://www.scrum.org/resources/what-is-scrum [Accessed 30 May 2019].

[5]Scrumstudy.com. (2019). Phases and processes in Scrum project| SCRUMstudy. [online] Available at: https://www.scrumstudy.com/whyscrum/scrum-phases-and-processes [Accessed 30 May 2019].

[6]Goebelbecker, E. (2019). REST vs RESTful: The Difference - NDepend Blog. [online] NDepend. Available at: https://blog.ndepend.com/rest-vs-restful/ [Accessed 30 May 2019].

[7]Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

[8]Anon, (2019). [online] Available at: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest [Accessed 30 May 2019].

[9]Vuex.vuejs.org. (2019). What is Vuex? | Vuex. [online] Available at: https://vuex.vuejs.org/ [Accessed 30 May 2019].

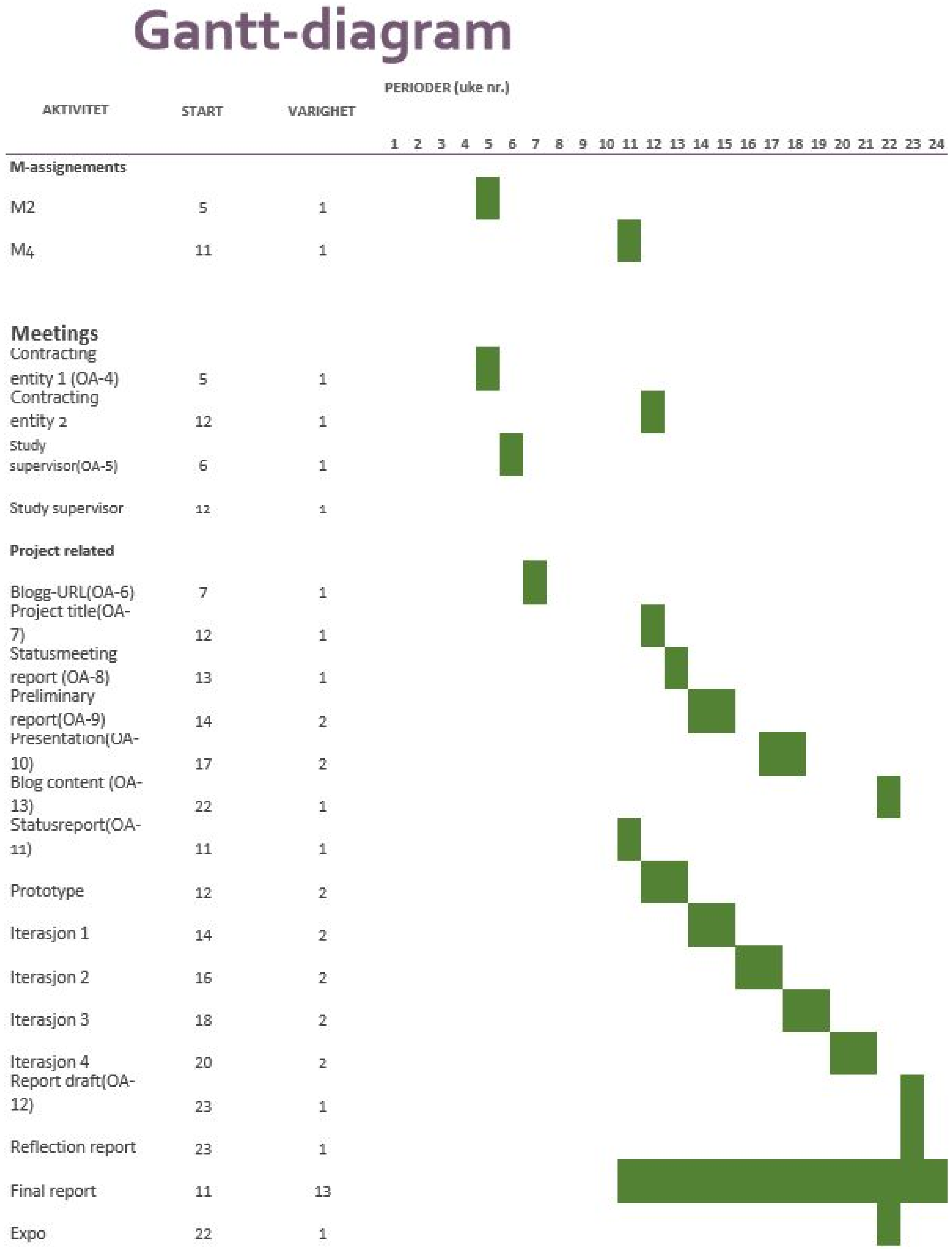
[10]Dynamic Routing (2019). Dynamic Route Matching [online] Available at: https://router.vuejs.org/guide/essentials/dynamic-matching.html[Accessed 30 May 2019]

[11]UX Planet. (2019). Designing for PC Apps. [online] Available at: https://uxplanet.org/designing-for-pc-apps-4554d8a0aa85 [Accessed 30 May 2019].

[12]Vuetifyjs. (2019). Vue.js Material Component Framework — Vuetify.js. [online] Available at: https://vuetifyjs.com/en/ [Accessed 30 May 2019].

Appendix

A: Gantt Diagram



Gantt-diagram

PERIODER (uke nr.)

AKTIVITET	START	VARIGHET
M-assignements		
M2	5	1
M4	11	1
Meetings		
Contracting entity 1 (OA-4)	5	1
Contracting entity 2	12	1
Study supervisor(OA-5)	6	1
Study supervisor	12	1
Project related		
Blogg-URL(OA-6)	7	1
Project title(OA-7)	12	1
Statusmeeting report (OA-8)	13	1
Preliminary report(OA-9)	14	2
Presentation(OA-10)	17	2
Blog content (OA-13)	22	1
Statusreport(OA-11)	11	1
Prototype	12	2
Iterasjon 1	14	2
Iterasjon 2	16	2
Iterasjon 3	18	2
Iterasjon 4	20	2
Report draft(OA-12)	23	1
Reflection report	23	1
Final report	11	13
Expo	22	1