Bachelor's thesis

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

Knotten, David Rise
Aglen, Einar Andreas

Time-Series Cache

Bachelor's thesis in BIDATA
Supervisor: Strazdins, Girts
May 2022

NTNU
Kunnskap for en bedre verden

Knotten, David Rise
Aglen, Einar Andreas

Time-Series Cache

Bachelor's thesis in BIDATA
Supervisor: Strazdins, Girts
May 2022

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

NTNU

Kunnskap for en bedre verden

Abstract

In the modern age of technology, an increasing number of devices get connected to the so-called Internet of Things. This makes it possible for data typically kept in a device to be transmitted to centralized servers for quick access anywhere. Typically, this data is time-series data, which is a collection, or stream, of datapoints of variables across time.

The project is based on a task given by Seaonics. The goal of the Time-Series Cache project is to fit Seaonics' needs for troubleshooting and monitoring of equipment which logs time-series data. In this project the group also focuses on making access and storage of the data more efficient in terms of time to retrieve, and the amount of data transmitted across low bandwidth internet connections. In addition to the service itself, the project also includes the development of a web-based interface for visualization of the captured time-series data. The purpose of this tool is to aid in troubleshooting with features such as side-by side comparisons and manipulation.

The resulting system consists of three separate parts:

- A caching service which stores data either locally or in a cloud increasing the data's availability by storing it on more powerful hardware and higher bandwidth networks. The caching service uses a pub/sub interface between locations, or vessels, and other services which need the locations information. This service is the core of the system and can be used on its own as a minimized package since all non-cache functionality has been purposefully implemented elsewhere.
- A REST service to extend the caches' functionality to provide an HTTP interface for web-based clients
- A web interface where signals can be discovered and monitored both in the past and in the present.

The result of this is a complete system with minimal need for configuration and that improves the availability of data for Seaonics needs. It is also not exclusive to Seaonics and can provide functionality for most typical scenarios where time series data is collected. Within this solution there is also the visualization component, which simply plugs into the main caching system and provides a web-interface for visualisation of the collected data with downsampling. This web-interface introduces some control to make comparing signals easier and massively improves the experience compared to looking over the raw data.

Sammendrag

I den moderne teknologiens verden er der en økning av enheter som er tilkoblet tingenes internett. Denne koblingen mot internettet har åpnet for sentralisering av tids-serie data som gjør at denne dataen, som typisk har vert lokalisert på enhetene selv, kan bli lagret i skyen for enkel tilgang når som helst, hvor som helst. Denne dataen, typisk tids-serie data, er en samling datapunkter lest av enheter og danner en serie over tid

Prosjektet er basert på en oppgave gitt av Seaonics. Formålet med Time-Series Cache oppgaven er å utvikle ett sett tjenester som kan bli brukt til feilsøking og overvåkning av utstyr, som for eksempel sensorer som lager tids-serie data. Gruppen har også fokusert på å forbedre tilgangen og lagringen av dataen effektiv med hensyn på tiden det tar å hente ut data, samt å minimere mengden av overført data. I tillegg til cache tjenesten inkluderer oppgaven også utviklingen av tjenester for å tilgjengeliggjøre ett web-grensesnitt hvor brukere kan hente ut og visualisere data gjennom cachen

Det resulterende systemet består av tre tjenester:

- Cache tjenesten selv, som bruker pub/sub grensesnitt mellom lokasjoner, eller fartøy, eller andre ting som holder på tids-serie data. Denne tjenesten er kjernen av systemet og kan bli brukt selvstendig som en minimert pakke. All funksjonalitet som ikke er direkte knytt til caching har blitt ekskludert for denne grunn.
- En REST tjeneste som utvider cache tjenestens funksjonalitet til å inkludere ett web grensesnitt for web-baserte klienter.
- Ett web grensesnitt hvor signal kan utforskes og overvåkes i fortid, samt i sanntid.

Dette resulterer i et system som krever minimalt med konfigurering og øker tilgjengeligheten for data i henhold til Seaonics behov. Systemet er også bygget for at andre enn Seaonics kan ta det i bruk i de fleste typiske scenario hvor tids-serie data blir dannet. Løsningens web-grensesnitt blir plugget inn i kjernesystemet og visualiserer dataen ved bruk av komprimering. Grensesnittet har forskjellige kontroller for å manipulere signal-dataen og forbedrer brukervennligheten stort sammenlignet med å se over rå-dataen

Preface

The Time-Series Cache task comes from Seaonics. Part of why the task seemed interesting for the group is its aspect of modernizing and improving processes that with older methods have been solved in inefficient ways. Through this project, the group is therefore able to improve important systems by introducing new ideas, and outside perspective, applying what the group has learned during their degrees. It is also a project that is likely to be used for important monitoring and troubleshooting for a long time going forward, thus giving the group a feeling of reason. It also includes a lot of what has been learned after these three years at NTNU, such as databases, web development, designing and creating web-app interfaces, but also adding some twists, such as learning the use of new communication protocols

The group would like to thank:

- Tom Jørann Giske, Erik Espenakk and Stig Espeseth for the task, the help, and the feedback we have gotten along the way
- Girts Strazdins, supervisor from NTNU, for guidance and acquiring the resources we have needed.

Assignment Text

The Time-Series Cache assignment is to develop a system for cloud-based caching of time-series data. The system must be based on the pre-existing API provided by Seaonics current MVP (Minimum Viable Product) of a vessel-side client with communication through the MQTT protocol. In addition to caching data, a web-based solution to visualize the cached data must be developed.

For more formal requirement specifications please look at the requirement documentation attachment

Table of Contents

Table of Figures

Table of Tables

1 Introduction

This chapter gives an overview of the task the group solves, and the structure of the report

1.1 Background

The time series cache project was given to the group by Seaonics AS. Seaonics is a local company in Ålesund that develops technology for ocean exploration and management of ocean resources. Over time as digitalization has reached most if not every aspect of industry, the development has converged toward the IoT (internet of things). As more devices, machines and equipment has joined the IoT, the amount of time series data has increased, and its use has spread to practically every place in the world where humans exist or travel.

With this development there comes the issue of how to handle and store this data, as it might lie behind volatile and slow internet connections such as satellite-based internet connections. The type of data also has the tendency to grow into massive datasets. The task given by Seaonics is specifically handling these cases where internet connections may be limited. This issue is quite typical within the Internet of Things and can be mitigated by caching, or storing, this data elsewhere so that these low-bandwidth internet connections usages are minimized. The group has thus been tasked to create a cloud-based solution to cache the data retrieved, reducing network load by eliminating duplicated requests of the same data across the satellite connections. In addition to the main issue, Seaonics have requested the creation of a tool, using the aforementioned caching-solution where Seaonics' engineers can quickly load the time series data to visualize and analyse it.

1.2 Problem description

The main goal of the project is to create a system for Seaonics that reduces usage of weak internet connections through caching, whilst improving the ease-of access of remote data. Additionally, a web-based interface should be developed where users of the system can easily fetch the data and visualize it, thus eliminating the reliance on other tools.

Even though the main goal is for the system to work within an existing software domain, the group hopes to create a service which is general enough to be used by anyone in similar scenarios with similar issues.

1.3 Scope

Since caching, which is the main issue at hand is an already massive subject, the group will limit the scope to use different pre-existing technologies or services, with the focus being to create a versatile cache. Seaonics already has a vessel-side prototype with its own interface, which the group will use to model the vessel-cache interface, instead of creating this service from scratch. Since the service collecting the data itself is not included in the project, anyone wishing to use the service will first have to create services providing this interface for the cache to use.

1.4 Structure

The thesis consists of 6 sections, with this section being the first. This structure acts as an introduction to where the issue came from as well as it should provide some insight into the issue as well.

The second section is about the theory behind the issue as well as theory behind its solution. In addition to this, the section includes information about the different technologies and methods used to solve the problem.

The third section of this document outlines the methods and methodologies the group has used throughout the project.

The fourth sections' purpose is to show the results of the combination of the theoretical aspects and the methodology.

The fifth section will discuss the results in context of the actual goals at the beginning of the project. Where goals were not fulfilled, this chapter will discuss why the discrepancies are, and potentially how they could have been averted. Part of this chapter will also be the discussion of the methods used throughout the project.

The sixth chapter will conclude this thesis by looking at the original issue in light of the results and following discussion. In addition to this, there will be some pointers towards where the project potentially could go, if further developed.

1.5 Target group

The target group for this project mainly consists of service engineers and offshore personnel working with equipment created and sold by Seaonics. The equipment in question is a variety of electric cranes and winches as well as launch and recovery systems. All expected to perform with minimal down time. And with the help of our solution these service engineers and offshore personnel can observe and analyse the various sensor data while this equipment is in use, as well as historic data to troubleshoot potential issues and discrepancies.

Other groups that also will benefit using the solution are in-house engineers working for Seaonics, with ease of access to historic log data and on demand preview, they can make new and improved products based on the performance of earlier versions.

While the primary target group is Seaonics,' and associated, personnel, the group has also focused on creating a service that is as general as possible. This way, other companies or people with similar requirements can use the system for their own benefit.

1.6 Acronyms

MQTT	Message Queueing Telemetry Transport
MQ	Message Queue
DB	Database
SQL	Structured Query Language
ORM	Object Relational Mapping
DBMS	Database Management System
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
JS	JavaScript
TS	TypeScript
NPM	Node Package Manager
JSON	JavaScript Object Notation
API	Application Programming Interface
MVP	Minimum Viable Product
IoT	Internet of Things
SSR	Server-Side Rendering
SPA	Single Page Application
UI	User Interface
IDE	Integrated Development Environment
TLS	Transport Layer Security
SSL	Secure Sockets Layer
CA	Certificate Authority
VM	Virtual Machine
OOP	Object Oriented Programming
TDD	Test Driven Development
MMI	Man-Machine Interaction
HCI	Human-Computer Interaction
XML	Extensible Markup Language
JSX	JavaScript XML
TSX	TypeScript XML
SEO	Search Engine Optimization
HTML	Hypertext Markup Language
MVC	Model View Controller

2 Theory and materials

This chapter's purpose is to give an overview of the theoretical aspects needed for understanding the problems solved in the project and the basis of the solutions produced. This includes concepts such as standards and protocols, types of architectural patterns, security considerations, and types of tools used throughout the process.

2.1 Caching

Caching is a very typical issue in computer science and can therefore be seen all over the spectrum of software engineering. It can be everything from caching a website, to caching data for data analysis and even computer architecture. Caching is, in layman's terms, storing subsets of data in a manner that makes it more readily available than the main data.

While caching can become highly theoretical, with things like the principal of locality, the groups focus is to apply existing techniques and theory for the purpose of solving Seaonics' problem of availability.

2.2 Communication standards and protocols

Since the system is, in part, a web-application, but also includes the cache service, multiple different communications protocols have been used, where HTTP and WebSocket has been used for the web components, or web-facing components, and MQTT has been used for the cache, and other back-end components, as it was a requirement from Seaonics.

2.2.1 Duplex

When talking about «duplex» in the context of communication, the terms half- and full duplex often come up. For some precursory knowledge, or understanding of the term, below are descriptions and examples of both.

Half-duplex communication is communication where data flows one way at once, meaning that when one entity transmits to another, the other entity will not transmit back at the same time, like sending a ball through a pipe, you can only send balls in one direction at once.

Full-duplex communication opens up for dialogue to go both ways, meaning that when the first entity is transmitting, the other entity can also transmit. With the ball-through-pipe example this is comparable to having two pipes, where one pipe can be used to push from a to b, and the other from b to a.

2.2.2 HTTP

HTTP is an application layer protocol which is also the backbone of the world wide web [1]. It is a half-duplex, synchronous protocol which relies on a request/response data flow. Since it is the protocol of the web it must support a lot of different functionality, which you get through its different methods, such as GET, POST, PUT, etc. In addition to these methods, HTTP uses headers to add context to requests, such as authentication credentials or tokens.

HTTP is the application-layer protocol which is the most widely used and in some ways IS "the internet", it should come as no surprise that this is the protocol the group has chosen to go with for the website and website-REST communication.

2.2.3 MQTT

MQTT is another application layer protocol like HTTP. MQTT does however work in a vastly different way to HTTP. While HTTP, as previously mentioned, supports synchronous request/response communication, MQTT provides asynchronous publish/subscribe communication. The way this works is that clients, as well as servers, connect to an MQTT broker. This broker allows clients and servers to subscribe to and publish to topics. While this scheme is quite simple, it is also flexible. MQTT operates with three levels of service, or "Quality of Service". The levels are numbered 0 through 2 and offer different semantics when it comes to the robustness of message delivery [2].

- 0: At most once, here, the MQTT broker will make a best effort to deliver the message, but it will make no guarantees. Thus, the broker will not make any efforts to re-transmit the message if the client does not acknowledge it has received the message.
- 1: At least once, here the MQTT broker guarantees that the message will be delivered at least one time. In the case where a message is sent, but no acknowledgement from the client has been received, the broker will retransmit the message. With this QoS, the message will be sent to all subscribers of the message topic, unlike with QoS 2.
- 2: Exactly once. When QoS 2 is used, the broker will select a single subscriber, and through a 4-way handshake ensure that delivery happens once, and that the single selected subscriber will have successfully received the message

Since caching data is relevant in both high and low load environments the idea of distributed services is important to consider. While it has not been considered much in the development during the bachelor project, the different QoS'es will be central when it comes to a re-worked cache with the goal enabling the service to support distributed environments.

2.2.4 WebSocket

The WebSocket protocol is a communication protocol designed for use on the web. Whereas HTTP is a synchronous protocol, WebSockets allow servers and clients to communicate asynchronously both ways, making it a full duplex protocol [3]. One can think of it as a full-duplex extension of HTTP. WebSocket connections are started by using HTTP Upgrade calls, which shows that the full-duplex extension analogy is fairly accurate.

Since HTTP is stateless and does not allow the server to initiate communication with a client, websockets were brought in for real-time streaming of data, allowing clients to get a live stream of the data they want to monitor.

2.2.5 REST

REST is an architectural design pattern used to create web APIs and is often used with HTTP as its application layer protocol. Its purpose is to provide a simple to use client-server interface for accessing and or manipulating the state of an application [4]. Since REST is only a design pattern, and not a protocol, it can be hard to find what makes REST, REST, but the main takeaway is its ability to provide the state in a manner where clients can understand and manipulate it, without necessarily having to know the actual representation of said state.

2.2.6 JSON
JSON, or JavaScript Object Notation, is a text-based format for representing (serializing) data as documents in a human readable form [5]. Its primary use is for transferring application data between servers and clients, often as the payload data on HTTP communication. Due to its versatility, it has recently been adopted for persistent data representation with the BSON standard with NoSQL databases.

Since JSON is a human-readable plain-text format its' use on the internet has been immense, and was the go-to protocol for the REST service, since it can be interpreted by JS effortlessly

2.3 Security

2.3.1 TLS
TLS is a protocol for secure and efficient encrypted communication [6]. The protocol is the successor of the SSL protocol, which is quite similar in use, with the protocols often being used interchangeably.

The way TLS works is that servers have TLS (or SSL) certificates, which contain information about the domain it is representing, but most importantly, the public key. This public key is used by clients to encrypt communication to be sent to the server.

The way a TLS connection is established, starts with the client requesting the server's certificate. The client can then check with a certificate authority, which is a service that issue these certificates, for its validity. Once verified, the client creates and sends a new key to the server, encrypted using the public key, which means only the server will be able to decrypt the message. Thus, the server and client have securely established a connection in which they are the only ones that know the key used to decrypt and encrypt messages.

The benefits of TLS are that the communication is encrypted, and unreadable by anyone not supposed to read them, assuming the keys are kept secret. Since only the client and server knows the secret key used for decryption and encryption, they can both be sure that the sender of any message is the real counterpart, and not an invading third party.

2.3.2 Certificate authority
A certificate authority is a service or organization that provides SSL/TLS certificates issued for different domains [6]. When accessing a website or any sort of web-service which uses SSL/TLS, the given certificate and keys can be looked up by a certificate authority for authentication. The CA can then affirm or deny the validity of the given certificate, thus ensuring that a connection is with the actual resource you are trying to access. CAs themselves are built in a tree with PCs coming with information of some higher-level CA. Since the CAs coming with a pc are known sources of truth, the authenticity of certificate authorities is a given. If this were not the case, CAs could be spoofed, and the validity of any certificate could not be assured. In short, a CAs role in security is to verify that a certificate is legitimate and belongs to the given domain.

2.3.3 Authentication and Authorization

Authentication and authorization are big, and important, topics when it comes to web-apps, but per Seaonics wishes the group has not included any form of either. The group has therefore worked with the assumption that the entirety of the system will run behind some private network and can rely completely on edge-authentication and edge-authorization.

Edge-authentication and edge-authorization means that to avoid any bad actors gaining access, the system must be put behind something like an API gateway with its own authentication and authorization, or a private network which only authorized users can access, either through on-premises access-points or methods like VPN tunnelling. The system therefore does not need features for authentication and authorization, as the figurative wall around it provides the security for the system.

2.4 Architecture and design patterns

Architecture has been an important aspect when designing the system. The entire system solves different issues, some building on others, the delegation of responsibility has been a conscious choice. Since the architecture of software impacts the flow of its information, its' design must be thought through to avoid complicating implementation, or even making implementation of later functionality impossible without major refactoring.

2.4.1 Microservices and monoliths

Microservices is an architectural pattern in which the focus is on maintaining smaller, separate codebases that make up your service [7]. Keeping services smaller make them easier to understand their complexities are limited to the boundaries of the services. This in turn makes changes much easier to implement and development speeds up. One of the major problems of microservices can be the choices around setting boundaries between services, which is necessary, as code will be separated, and what would be simple method calls in a monolith will be requests over the web. This problem is also a blessing, as it forces good cohesion and coupling between the services. In turn, this affects the entire system positively, as one does not need a complete understanding of everything that is going on in a system, which might consist of hundreds of services, instead, the implementation of a single service can be changed and worked on completely independently from other services, assuming its outgoing services, or API remains the same.

Due to the clearly defined lines between the services, the choice to go with a microservice-architecture has been an obvious one. It has allowed the group to create services which are a lot more versatile than what could be made as a single program, or monolith. It has also enabled the creation of a cache-service which can provide only the exact functionality of caching, allowing for a lightweight service for use cases differing from the use cases of Seaonics.

In contrast to microservices, there are monolithic services, which is the more traditional approach to software architecture. Monolithic services represent the entire system using only a single service, which typically leads to increased coupling and reduced cohesion. This in turn makes further development harder since there are no longer any lines drawn between individual components, meaning that minor changes could have massive unexpected consequences.

In this project, the use of Monolithic services would lead to a single application providing both the cache functionality, the web-facing interface, and likely a static website.

2.4.2 Sidecar pattern
The sidecar pattern involves secondary processes running alongside the main process [8]. A typical use case is service meshes, where a sidecar hides the complexities of networking to the main service. Sidecar patterns can also work within a single service too. When some of the code should run independently of other code, this pattern can be helpful.

In Node, this pattern is quite easy to implement with the child_process module which creates a new Node process with its own resources and a message-passing system.

2.4.3 Model-View-Controller pattern
The Model-View-Controller design pattern is a for developing user interface where the layers of logic are separated into the model, view, and controller. Each layer tries to stay within its domain, and our-source tasks to the other layers when needed.

- Model

  The model in this patter consists of the data-related layer. Here the various states and data is handled. Both the view and the controller listen to this layer. As the underlaying data is handled here.

- View

  The view handles the underlying structure of the visual representation of the user interface. All boxes and container, buttons and text fields are defined here, and they should know nothing other more than where they are positioned and how they should look.

- Controller

  The controller is a business layer in between the model and the view. It takes the request from the view and sends it to the model, and then handles the update in the view after the requested task is complete.

2.4.4 Proxy pattern
The word proxy has multiple meanings both in computer science and other contexts. In this context, the proxy pattern is the focus. The word proxy's meaning first came from a person being the spokesperson of someone else, which is what the proxy pattern is on a higher level, some software acting on the behalf of some other software [9]. There is, however, more to it. Using proxies one can re-define parts of how the underlying software works, for example with TLS-terminating proxies, where a webserver lies between a server and its clients, providing encrypted connections whilst being «invisible». This is typically what the pattern is used for, implementing something around an existing system.

In the case of the Time-Series Cache, there already exists vessel clients, but they lack any cloud-caching ability, even though their interface works as intended. The cache service will therefore act as a caching-proxy for the vessels, using a similar interface with the same capabilities, but improving on it. Going back to the origin of the words meaning, the cache provides an improved interface on behalf of the underlying vessels.

2.4.5 Observer pattern
When writing code where multiple «branches» of code are executed concurrently, there often occur situations where some code must wait for another piece of code to complete or reach a set point or state. This is where the observer pattern comes in handy. With the observer pattern, one component can subscribe/observe to another components state to ensure that a state-change in one component gets propagated to other components that use or otherwise rely on the notifiers state [10].

In JS/TS this is a typical occurrence as it has a solid Promise API making asynchronously executed code easy to implement. In the Time-series Cache, such a pattern has been used to notify when files are ready to be read when a request is partially, or fully covered by a request in transmission.

2.4.6 Singleton pattern
The singleton pattern is one of the most common patterns in software engineering. A singleton provides a single-instance implementation of the underlying class/object etc [11]. This can be especially useful for creating global entities that have a high cost of instantiation, or simply for centralizing state in a single object which can be fetched anywhere.

In context of the services developed in the Time-Series Cache project, singletons have been useful in avoiding mass-instantiation of certain expensive resources such as MQTT clients. Another use of singletons has been the metadata aggregator where having multiples would lead to a lot of collisions when aggregating.

2.4.7 Website Routing
The website follows familiar architecture that can be found in most websites serving users on the web today. It consists of creating logical routes that build on each step the user takes further into the website.

When setting up routes it is important to figure out which routes should be static, and which should be dynamic. A static route can be viewed as a folder on our computer. It will always have the same name. This looking it up will yield the contents of that folder, and what lies within this folder should be related to the name of the folder. A

A dynamic route can be viewed as a readable variable that decides what you are going to view on the website, it is therefore mutable in the sense that it can be changed while running to yield different outcome. The easiest example if this could be a product route; it is given the prefix "/product" and is followed by the ID of the product look up to display to the screen. So, the content of the website will be different in the case of "/product/1" and "/product/2". [12]

A good example of the static and dynamic routes are as follows: If a user were to store some images and files relating to cars on their computer, it would be reasonable to keep a folder named "cars" in folders with certain car names can be found as well as an image folder. Each folder with a car name might hold files and plans relating to the car name. While the image folder might contain an image of each car with filenames containing the car names. With this, a specific file of the car can be looked up given the lookup "cars/{carname}/{filename}" etc. But for the image of the file, the route "/cars/image/{carname}" would be used. This is a simple example of how routing can be used to create a logical lookup pattern for accessing various parts of data via routing.

2.4.8 Website Components

Web development has come a long way since the 1990's. Back then, the web consisted of mostly static websites that connected various documents together and web applications were the stuff of dreams for developers. However, a lot has changed after the creation of JavaScript in 1995 by Brendan Eich, which later became the ECMA-262 standard in 1997 [13]. With the power to run more logic in the browser, developers have created large libraries for improving user experience as well as developer experience.

With ReactJS which is the framework the group used for the project; the user interface is written in a component-based pattern. Structure and logic is defined here. It can be used repeatedly, making the development of a we [14]

2.5 Databases

A database is a way to persistently store structured data [15]. They are typically divided into SQL databases, which are relational databases using the SQL language, and NoSQL databases which are any other type of database, such as document databases. In this project, an SQL database has been used, where a row in a table represents a single entity. Such an entity can be a person, or a relationship between different people.

2.5.1 Indexes

Databases typically contain a lot of information, which might take quite a lot of time to look up if you need to go through every row. This is where indexes come in. Indexes are data structures used to improve performance by ordering the data [16]. If, for example you had a row of random numbers, the only way to find what you are looking for would be to go through one-by-one until you find it. However, if the numbers were ordered, you could use an algorithm such as binary search, where you split the entire database in half for every search, giving you a search time of log(n) where n is the row count and log is the logarithm of base 2. Indexes do not need to be set up a way, but it serves as an example on how to improve the time to look up a record.

In the previous example, it might not seem that important to improve lookup times, but imagine a database with millions, if not billions of records of time-series data, where you would go from a million attempts to locate your record to 20.

$$\log_2 1000000 = 19.9315$$

Or if there are billions of records, it would go from a billion attempts to

$$\log_2 1000000000 = 29.8973$$

2.5.2 Foreign keys

In relational databases, the limitation of only being able to represent data by rows in tables is lifted by having relation between the tables and rows in them. These relations are represented by foreign keys, which are columns of one table used to reference a row in another [17]. An illustration of the person-relationship foreign key relation is shown in Figure 2-1.

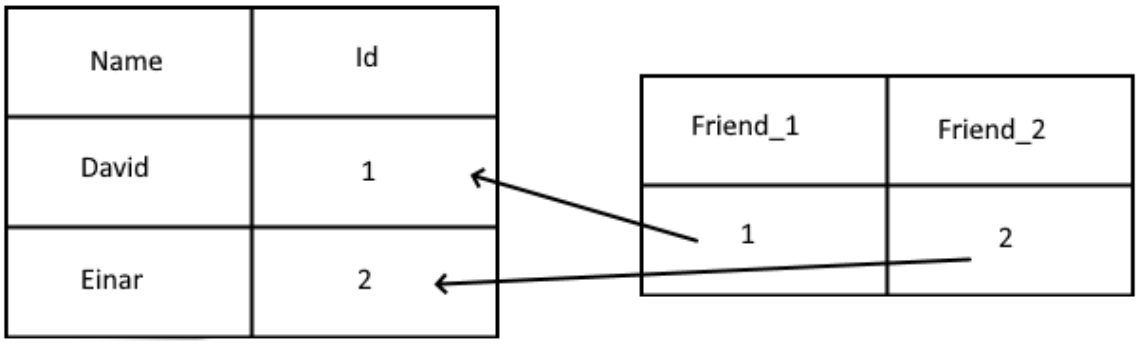


Figure 2-1: Foreign key relations

Here, the right table consists of two foreign keys which both point to a user, representing a friendship between the users.

2.6 Virtual Machines and Containers

When deploying software at any scale, having controlled environments is important to ensure that deployed software will behave as intended and the same way regardless of the specific instance. One way this has been done is using Virtual machines which are built upon the virtualization technology. Virtualization allows a host machine to provide its resources to a, or multiple VMs, in a sense tricking the VM to believe that it is a physical machine. VMs provide independent operating systems which act like regular computers without requiring unique hardware for each system [18]. This improves the scalability of the hardware and drastically reduces the need for high quantities of machines, since one more powerful machine can host multiple systems. Since VMs are purely virtual, they are easy to duplicate with pre-configuration through virtualization management software.

Containers are a technology where code can be bundled with only the resources needed to run it. What makes containers so useful, compared to regular VMs is its extra layer of abstraction. Software often has a lot of different dependencies, some of which may clash with other software running on any given VM, but by using containers, code can be run separated from the environment on which it runs, and therefore isolated from other processes. This way, one can deploy code without the need to worry about dependencies, as it is all baked into the container, and all it needs is the resources of the underlying machine. This means that you can build your container image, which is the recipe of a container on one machine, and deploy it anywhere else assuming virtualization is possible, albeit an Ubuntu or a Windows Server is the underlying operating system. While this could, with some more work, also be done with regular VMs, the operating systems running on VMs are typically a lot heavier, severely impacting the number of separate environments you can run on a single machine [18]. Additionally, when deploying to virtual machines without containers, there is often a lengthy process required to install

dependencies, update to specific versions of software etc. Even though this process can be automated, the simplicity of deploying a container is incomparable.

Since the group has used containers for deployment, anyone can run up the services without any hassle, assuming they have docker, massively simplifying the process of deploying code.

2.7 Languages

2.7.1 JavaScript

JavaScript is a scripting language that was invented to help developers create more complex behaviour when developing websites by enabling clients to run code. It was originally created by Brendan Eich in 1995 when he was working at Netscape. The language was originally intended to be used for Netscape's own web browser [19]

JavaScript is the third building block to how websites work today, where HTML is the literal structure, CSS acts as styling for the structure, and JavaScript acts as the logic. JavaScript is widely used for web development as it can refer to DOM elements found in the website and is supported by all browsers. It is a dynamically typed language running on a single thread where an event loop enables it to run similar to multi-threaded programs seemingly running code in parallel, even though it does not.

As of 2009, JavaScript has also been for server-side code with the Node.js runtime [20]. There were other solutions before Node, such as the NetScape LiveWire runtime, but Node is the currently dominating runtime by far.

2.7.2 TypeScript

Typescript is a superset of JavaScript, meaning that it is strictly built on top of JavaScript. As of today, most of the web is running JavaScript, with w3techs reporting that 97.9% of sites do [21]. As with other dynamically typed languages such as Python, plain JavaScript becomes hard to use with bigger code bases. This is where TypeScript comes in. TypeScript is as previously mentioned, a superset of JavaScript, adding static typing, avoiding the most common issues with JS wherever the codebase has grown past simple applications, websites etc, and the ability to understand what something is, is lost. The introduction of proper typing and interfaces allows for more complex and larger codebases. If one has a little experience with statically typed languages like java, the move from JS to TS should be quite easy. Another major benefit of TS is the fact it is compiled to JS, in practice, this means you can write your code any way you want, and still transpile to your preferred version of JS. If the code you write needs to be integrated into an existing JS codebase, it therefore should not be a problem. However, the reason the group went for TS over JS in this project is the avoidance of issues arising from dynamically typed JS, semantic errors from "typeless” code.

TS/JS make single-threaded programs, meaning that all code is executed synchronously even though the Promise API can make it seem and act like it runs in parallel. This helps in eliminating typical cases of race conditions, but they can still occur. One downside to this is the fact that everything runs on one thread, meaning that if some code holds up a large amount of CPU time, it will block any other code that is queued in the event loop [22]. For this, the language introduces some concepts such as “getImmediate”, which will put the currently executing code back in the event loop, and the workers module, which can execute synchronous code asynchronously as background tasks.

2.7.3 SQL

SQL (Structured Query Language) Is the most widely used language for database querying, at least it was in 2019 [23]. It provides a simple, yet powerful syntax allowing users to query both simple data, and more complex structures and relations. Since the group has used TimeScaleDB, an extension to the popular PostgreSQL DBMS, this was the language used in database communication.

2.8 Time series data

As more devices are connected to the internet, the possibilities around monitoring have increased. Sensor data is one of these things that have become increasingly relevant in the modern cloud and internet. This data is a major part of the IoT, where devices become connected to the internet [24]. These devices often stream their sensor data to centralized clouds-services where the data is used for monitoring, troubleshooting and more. This type of data is called Time Series Data and is characterized by repeated measurements of some entity over time [25]. While the individual records of time series data are quite small, they typically have a massive volume, with sensors often having measurement intervals in milliseconds, meaning potentially thousands of datapoints per second, per sensor, per device. This property is the main contributor to time series data's tendency to turn into massive data sets.

For this reason, there are some additional considerations to storing this time series data. While most types of data are quite easy to handle assuming reasonably efficient algorithms can be used, time series data brings with it a massive scale with its own issues. For example, have several types of databases been developed specifically for the purpose of dealing with massive datasets of time-series data. These databases typically have the primary goal of optimizing high-throughput usage with massive read/write capabilities. Another relevant issue when dealing with time-series data is memory usage. When loading a few rows from a database there is not much consideration required, but when loading a, or multiple millions of rows of data you will have to thoroughly consider memory usage

2.8.1 Time

When most people think of time, they think of time in terms of years, months, dates etc, and can quickly understand the meaning of a timestamp given as an ISO-8601 string. An ISO-8601 string has the following format «YYYY-MM-DDTHH:mm:ss.SSSSz», which starts with the year, all the way down to milliseconds, followed by z, which is the timezone offset [26]. One thing that makes this representation of time sub-optimal for computers optimal is the fact that it requires interpretation by the machine.

Instead, computers typically use Unix-time, which is the amount of milliseconds elapsed since january 1st 1970 GMT/UTC. Since unix timestamps are represented as raw numbers, they are much faster to process, which is an important aspect when handling millions of datapoints.

Two equal timestamps reference:

ISO-8601: 2022-05-18T18:16:52
Unix: 1652897812000

2.9 Universal design

2.9.1 Principles

The Idea behind Universal Design is that any form of environment that is going to be accessed by the public should be designed and formed into something that can be used by anyone. Anyone implies accessibility regardless of age, health, or disability.

There exist 7 principles of Universal Design, these were developed in 1997 [27].

1. Equitable Use

   The design is useful and marketable to people with diverse abilities: this can be interpreted as a guideline for the development process. The solution should provide equal usability for all users where it can be achieved. Segregation should be avoided at all costs as this can lead to stigmatization. All users should feel like the solution is safe for them regardless of ability. And the solution should be inviting and appeal to all users.

2. Flexibility in Use

   The design accommodates a wide range of individual preferences and abilities: a wide range of individual preference can be interpreted as giving the users multiple options as to complete of access parts of the solution, to build on this, the tasks in question should allow for left and right-handed usage, help users with accuracy if there should be tasks containing precision.

3. Simple and Intuitive Use

   Use of the design is easy to understand, regardless of the user's experience, knowledge, language skills, or current concentration levels: complexity just for the sake of complexity should be avoided. Consistency is necessary as to meet the user's expectations when learning to use the solution. Information should be arranged so that there is no doubt surrounding it.

4. Perceptible Information

   The design communicates necessary information effectivity to the users, regardless of ambient conditions or then users' sensory abilities: If the solutions allow it there should be tactile feedback that can help the visually impaired to navigate. Contrast should be kept various enough that it does not confuse the colour impaired. The solution should be compatible with devices found in the hands of the visually impaired.

5. Tolerance for Error

   The design minimizes hazards and the adverse consequences of accidental or unintended actions: There should be guidelines and forethought put into the design of a solution that is to make it next to impossible to get into situations that cannot be easily undone.

6. Low Physical Effort

   The design can be used efficiently and comfortably and with minimum of fatigue: As previously mentioned, where a designer can help the user they should help them, so the overall interface is designed in a way where the effort required by the user to navigate, or complete tasks, does not exceed the amount the typical user is comfortable with.

7. Size and Space for Approach and Use

   Appropriate size and space are provided for approach, reach, manipulation and is regardless of user body size, posture, or mobility.

2.9.2 Human-Computer-Interface

HCI is often talked about in the context of HMI, whereas HMI is the interaction between a human and a machine, HCI is the interactions between humans and computers, more closely the user interface that lets humans control the computer. A common type of HCI is a touch screen. It is what allows people to interact with our phones and do daily tasks at incredible speeds [28].

2.9.3 Gestalt principles

The Gestalt principles is a list of principles that emerged from a theory Published by Max Wertheimer in 1923 called "Gestalt laws of perceptual organization" [29]. These principles are important to any design process in that they guid the designer to make the right choices in form of aesthetics and functionality.

The idea behind the principles is that the human brain actively tries to organize and simplify the complex nature of our visual surroundings. This is done by arranging parts of our reality into smaller parts and groups aspects of it together. The Gestalt principles try to manipulate this part of our brains behaviour by designing solutions that our brain will have no issue in learning without requiring too much effort.

1. Similarity

   In the Gestalt principles, similarity is suggested to be used to make humas draw connections between shapes and colours based on their similarity, as if the user knows the function of a button with a said colour and shape, The users are likely to know what the button does if it presents itself in a later moment.

2. Continuation

   The principle of continuation builds on the law of continuity that states: The human eye will follow the smoothest path when viewing lines, regardless of how the lines where draw. The designer should try and create lines in their design to guide the eyes of the user so that they will follow the path that was intended when viewing the solution.

3. Closure

   With closure, the brain tends to finish what the eyes see, if there are unconnected lines or gaps in a design, it makes the viewer see the finished product like the designer intended. This principle can be quite effective and is widely used in logo design.

4. Proximity

   Organizing elements that are shown in bulk at the same time can be tricky and end up clunky and over managed. Straight lines can hinder the flow of the design and create splits the designer wants to avoid. Therefore, simply grouping elements that are connected in some aspect a little closer together that the rest is a great tool for clean organization. This principle takes in the effect that our brain associates items that are closer together with each other rather than the rest. In general, the brain will see the biggest collective mass of the image as the background and the smaller masses as the foreground.

5. Figure / Ground

   The Figure/Ground principle is closely connected to the closure principle in the way that your brain in some ways finishes the design of the designer. With Figure/Ground this effect is regarding the foreground and background and how the brain distinguishes what is closest to the viewer in an image. With this effect layers can be created from visual media that lets the user know what is closest and what needs attention first.

6. Symmetry and Order

   Symmetry and Order plays on the brains perception of what is a unified group. And in general, the brain will group items more often if they are of similar shapes, and even more if they are symmetrical to each other.

2.10 User Interface Design

2.10.1 Consistency

When designing a UI, it is important to be consistent with design choices, this lets the user learn the underlying pattern that is in place for them to get better and to become more familiar with the UI. [30]

Consistency is found at multiple levels of UI design choices. Most people today have become familiar with the base structure of a website or program, its kept simple, typically with a navigational bar at the top of the screen, a side bar at either side of the centre where the main content can be found. If one deviates too far from the typical website layout, one can often be met with negative feedback, and disgruntled users.

This principle is well formulated by Jakob's Law, this law states that; "Users spend most of their time on other sites. This means that users prefer your site to work the same was as all the other sites they already know"

2.10.2 Ease of Use

Ease of use usually comes down to one measure; "how many clicks does it take the user before they are at their desired destination? ". This question is central to keep in mind when setting up the base architecture of a website. If the design process is negligent and this is forgotten, then the result can become a hard to understand or user for anyone other than the developer that created it. [31]

2.10.3 Forgiveness

The consequences of an action when browsing a site should not be significant enough that the user easily can undo them or go back to a previous state. This will let the user feel the freedom to be curious without any significant repercussions. Building further on the principals that the UI should teach the user over time. [32]

2.11 Data Visualization

2.11.1 Datasets

In data visualization the key to generating informative visuals and tools to understand what is being displayed is knowing what kind of dataset is being visualized. Datasets come in many different forms and should be taken care off according to what is the best fit. The most generic form of data is time-based data. Time-based data has data points scattered over time, it could be once every few days, or multiple data points every second. Almost everything anyone does on the internet is logged with timestamps, what the user clicks, what sites they visit. All this can build informative and accurate charts displaying what they do in a day and when they do it.

2.11.2 Chart for data visualization

A chart or a graph is usually a visual media that represents a statistic from a dataset in a more understandable way than the raw data. There exist a lot of unique types of charts, and they all have their own traits and features that might match a given dataset better than other charts. Some common types of charts are bar charts, stacked bar charts, line charts, and pie charts. [33]

Bar charts are commonly known, with most people having seen some form of chart displaying various data depicting changes of some statistic between years where the statistic is represented by the height of a column. Bar charts are especially well suited for categorical data, these are datasets containing set data entries, data that can either be a "yes" or a "no" value. When data categories can be divided into sub-categories, stacked charts are frequently used. Stacked charts are essentially bar charts, where each column is split into separate sections, thus giving an overview of the whole category through the columns height, but also the relative, and absolute magnitudes of the subcategories

Line charts are possibly the most familiar types of charts, where data is visualized as a line. Line-chart data consists of an x value, which is typically a discrete representation of time, and a y value, which is the value at the given time. An important characteristic of the x-axis in most line charts is that it is ordinal, meaning its order matters. Raw time-series data is typically represented with line charts, as time-series data has values (y) distributed over discrete-time (x).

Pie charts can help visualize the divide of a dataset; it can display the percentage in form of the site it takes up of the complete circle. A common use case can be displaying the results of a poll, where the percentage of users that choose the different options are displayed as slices, this gives the observers a better idea of the divide.

2.11.3 Scales

When displaying data in charts with axis a frequent problem is to decide what type of scale to base the drawing on. A scale can have a tremendous impact on the overall visuals and can help the observer see trends that might be hidden in other types of scales.

2.11.3.1 Linear Scale

The linear scale is one of the more common types of scale. It is usually found on the y-axis of most charts. In a linear scale the progression is as one can guess linear. Meaning that the value of increases along the axis with equal amount for each tick. [34]

2.11.3.2 Logarithmic Scale

The Logarithmic scale is widely different from the linear scale. It has a logarithmic increase along the axis and is therefore better soothed in use for displaying dataset with tendencies for large spikes in data. These spikes would create large towers that takes away practicality of view the other data points found in the dataset. All other points of a dataset with spiking values would be seen as straight lines in the bottom of a line chart and would serve no use for the observer. A logarithmic scale would fix this by making the spike fit in with the rest of the values. However, a logarithmic scale could also hide data spikes by making it seem like the increase is linear. [34]

2.12 Version control

Version control is a central concept in modern software development. When systems grow, and the number of developers working on them increases, having control of code would be an impossible task if the developers had each their version locally. This is where version control comes in. Version control works by applying changes to a codebase where the work done by multiple developers is (mostly) seamlessly merged [35]. Some central concepts with source code are commits, branches and pull requests. A commit is the application of a set of changes to the codebase. A branch is an individual series of changes, branches are often created for a purpose, such as being able to work on the implementation of one part of the program without having to worry about changes to the main program as work goes forward. Once a branch has served its purpose and the planned implementation is complete, it is often merged into the branch it was branched off from by a pull request, thus also applying the changes to the main part of the codebase. There are typically two central branches, a main/master branch which is updated with stable code, and a development branch which is more actively worked on. In addition to the two central branches there are often more branches branched off from them for specific features.

While version control might be looked at as a collaborative tool it is much more than that, and a critical piece of software to keep track of changes regardless of team size.

2.13 Code quality

When writing code engineers often get tied up in getting things done and working, but to make a codebase sustainable the quality of the code must be considered. The quality of a piece of code can be judged by multiple factors, but one of the most important one is the readability of it. Developers come and go, and it is therefore important to write understandable and concise code. Even though the goal of the group is to solve requirements of Seaonics, the group hopes that the resulting MVP will be developed further, and it is therefore important to write code that is easy to read.

2.13.1 Code style

When writing code, there is a multitude of different choices you can make about how you want to write class names, place brackets etc. For the sake of readability, there are certain conformities developers should abide to, which are different guidelines around how code should be written. A typical example of code styles is how to name variables consisting of several words.

Camel-case: aWord

Pascal-case: AWord

Snake-case: a_word

The reason abiding to these conformities is so important is to make reading the code as easy as possible by only having to relate to one style independent of the developer who wrote it. In some cases, the style can give information about the variable. An example of this is the convention of prepending underscores on internal/private variables in Dart (doing so also makes the variable private, so it is not strictly just a code-style convention).

2.13.2 Linting

Since there are many conventions and rules when it comes to how code should be written, programs called linters have been created. The name linter comes from the name of the program Lint, which was a program made for C, which performed what's now known as linting [36]. Linters analyse the code, looking for any code that does not conform to the rules set by or for the linter. This way, even when there are multiple people coding at the same time, one can be sure that the code is written in the same style. While linters are often able to catch syntax errors, their main purpose is typically code-style, with the responsibility of syntax checking falling on the compiler.

2.13.3 Cohesion

Cohesion in code is the principle of how well code does its task and how concise the task is. Writing cohesive code thus means writing code where each piece solves a specific purpose [37]. The goal is to write cohesive code so that understanding what a component does, will be obvious, as well as decreasing complexity since the complexity will be divided into smaller, more concise code snippets. If code is written with high cohesion, it should be easy to understand its function and to make use of it.

2.13.4 Coupling

In coding, the term "coupling" refers to the interdependence between components. The goal is to write loosely coupled code, meaning code that consists of pieces where each piece acts independently, and the implementation of one piece of code, should not interfere with the semantics of another piece [37]. Coupling is important for the readability of the code since tight coupling often involves low cohesion and reading a web of code becomes a daunting task quickly when compared to a loosely coupled codebase that reads more like a book. Tight coupling also introduces the issue of unwanted side effects that may not show themselves and therefore create bugs, or in the worst case, completely break important functionality somewhere unexpected.

2.13.5 Abstraction

Another aspect of code quality is abstraction. When you write code, you should have a goal to make the code as versatile as possible, allowing you to reuse the same code later for different uses, for example by using concepts such as polymorphism [37]. A great example of abstraction are lists. A list includes the typical insert, remove, length etc functions. This does not tell you anything about how it is implemented, as it can be a linked-list implementation using nodes, or an array which lies beneath. An even more useful (arguably) example is the use of generic types, where the entities a list can hold is defined at each use. Imagine having to create a list class for every single datatype used in a program.

2.14 Used Materials

2.14.1 Cache Service Libraries

2.14.1.1 Pg

The pg package is the database driver for Postgres on JS, and a dependency for the Sequelize package to allow connectivity to Postgres databases. Since Sequelize makes some assumptions about the database, mainly that the database it is trying to connect to already exists, pg is used when the cache service is started to ensure that the wirelogger database exists.

2.14.1.2 Sequelize

Sequelize is an ORM package supporting the most popular relational databases, such as Postgres, MySQL, and SQLite. The reason Sequelize was chosen is primarily popularity, as it is by far the most popular package. Sequelize also provides an uncomplicated way to create queries through a JSON like query building object, whilst still allowing raw SQL statements. Sequelize can also be used to manipulate the database, for example for creating tables. While Sequelize does not have TimeScaleDB support, its raw SQL support allows the use of its functions.

2.14.1.3 Protobuf

The cache service uses multiple Protobuf libraries, protobufjs and ts-proto. Protobufjs is a pure JS implementation for serializing data to Protobuf. However, due to a lacking support for typescript, the group has used the ts-proto library. Ts-proto is used to generate native TS code (compared to protobufjs, which only creates declaration files), but other than that it has no use. Protobuf-ts is another package for generating typescript code, but the generated code is harder to use, which is why it was ditched in favour of ts-proto

2.14.1.4 Mqtt

MQTT.js is an NPM package providing capabilities for MQTT connections as well as an MQTT client. The client provides a simple-to-use callback-based interface where you subscribe to topics and provide a function which is called whenever a subscribed topic is published to.

2.14.2 Libraries – REST Service

2.14.2.1 Protobuf

For communicating with the cache service

2.14.2.2 Mqtt

For communicating with the cache service

2.14.2.3 Expressjs

Expressjs is a library meant for server-side applications. With it, you can create an Express app, which is an http server. Out of the box it does not include too much functionality other than providing the client, but, through middleware, an express app can be built into a large system. The main reason the group went with Expressjs was for simplicity and the ability to expand later if requirements would require. It can also be noted that it is the most used library for backend JS/TS. Compared to other frameworks for server-side code, Expressjs is quite fast, only getting beaten by the Spring framework [38].

2.14.2.4 WS (WebSocket)

While Expressjs provides a synchronous API through HTTP, the group has explored how to stream data through the REST Service, even though it will break some of the principles of REST. For this, the group used the ws library. The ws library can easily be used with Express by using the socket created by the Express app. For data streaming, a WebSocket server was attached to the express apps socket on the "/live" path

2.14.3 Libraries – Frontend

2.14.3.1 ReactJS

ReactJS is a frontend library that uses the power of component-based UI and generates JavaScript bundles to be sent to the browser for rendering. ReactJS on its own Is used to create single-page-applications. ReactJS is written in a JSX syntax which merges the C-style syntax of JavaScript with the structural layout of languages like XML and HTML. React is open source but receives support and maintenance from Meta, formerly known as Facebook. React can be paired with other frameworks to build more than just single page applications, with NextJS it can produce SSR webpages, with React-Native it can build mobile applications for both android and IOS, and with Electronjs it can build desktop applications for Windows, Linux, and MacOS.

2.14.3.2 NextJS

NextJS is a wrapper library for the React library that gives all the benefits of React whilst also having the power of SSR. NextJS uses the same JSX / TSX syntax as React with added components such as custom Head, Link, and Image components to speed up load times when fetching files from the webserver. The main benefit of NextJS is as mentioned above, the power of SSR and static page generation. With SSR and static page generation the developer can manipulate SEO principles for making a website easily available for users through search engines.

2.14.3.3 Chakra UI

Chakra UI is a component library for React. A component library is a library of prefabricated components that can be used to build a user interface. They have their own styling and props which can be accessed by the developer to tweak into the desired functionality. Chakra UI comes with loading animations for buttons and inputs out of the box, as well as templates for page and card layouts that can be used for inspiration when building the page layout. [39]

2.14.3.4 ChartJS

ChartJS is a popular open-source data visualization library that lets the developer create functional and aesthetic charts. The library harnesses the power of the modern-day HTML canvas which is a built in HTML element for drawing graphics in the browser. Since it is well integrated performance when using ChartJS is excellent on most types of devices and browsers. ChartJS is the second most used data visualization library according to GitHub, only beaten by the extremely popular D3.js library. Presenting data with ChartJS can create pleasing results, since most changes to the dataset is fully animated as well as the possibility of plugins to make charts interactable, the user experience is there for unparalleled. [40]

2.14.4 Software

2.14.4.1 Docker

Docker was used for deployment of code as well as for easy setup and teardown of processes. The reason Docker was chosen is the fact that Docker is the de facto standard

when it comes to containerization software. There are other services such as PodMan and BuildKit but the likelihood learning to use those will yield any advantage for the group after education is slim. Another approach considered was using Kubernetes, but it has a much steeper learning curve and Docker alone fit the requirements, thus Kubernetes would just be adding unnecessary complexity. The steps the group has taken for deployment is also just for the sake of demonstrating the system, with Seaonics taking over the responsibility of deploying it in their internal systems

2.14.4.2 Postgre/TimeScaleDB

Due to the nature of time series data which the cache service will handle, having a database with high insert rates is especially important. The database will also need to be able to hold considerable amounts of entries. While more traditional SQL databases such as MySQL and Postgre do a fine job, the group chose to go with TimeScaleDB.

TimeScaleDB builds on Postgre with a focus on optimizing the database for time-series data and its high insertion rate requirements. Since it is built on Postgre as a database plugin, it also has the benefits of Postgres' tried and tested DBMS, including full support for SQL, which often is not the case for time series databases.

In terms of performance, TimeScaleDB is fairly similar to Postgres, with improvements for time series data. It also comes with data life cycle management and data compression features which can help in keeping database sizes down over time. As mentioned in the referred papers, the queries used for benchmarking were quite simple and straight forward, not allowing TimeScale to shine, since its performance benefits come on more advanced queries.

Some alternatives to TimeScale are InfluxDB and Prometheus. These alternatives do, however not support SQL, and require the use of their own proprietary querying languages. Additionally, the market looks to trend towards TimeScaleDB with InfluxDB and Prometheus being older DBMS with some growth stagnation [41].

2.14.4.3 Eclipse Mosquitto

Eclipse Mosquitto was the chosen broker used in the test environment. It is a lightweight MQTT-only broker and client. The reason Mosquitto was chosen over other brokers such as HiveMQ is its best-in-class performance [42].

2.14.4.4 Google Protobuf

Protobuf, developed and maintained by Google, is an open-source tool for data serialization which supports all the commonly used languages like Java, JavaScript, Ruby etc. While JSON is an arguably easier format to use, since cleartext is easier to debug, it is not necessarily well supported on all languages and each language might require some digging to find a library fulfilling the needs of a service. This is where Protobuf comes ahead. Protobuf builds its own custom code from a .proto file which contains the definition of the data you want to serialize/deserialize. This way, any service using the cache service already has a full library ready to use after generating the language-specific code. This is especially important for the cache service since it aims to create an API which is easy to use for other services with as few limitations and problems as possible. One thing to note is that its support for typescript is lacking, but there are multiple libraries created to solve this problem, and typescript-native code can be generated using them. For the services in this project, the ts-proto library was used for this purpose.

2.14.4.5 AWS (Amazon Web Services) / Cloud

While deployment is not within the scope of the project, there are some practical aspects to using cloud services. The service provided chosen by the group was AWS due to the group members having some experience with the platform. The demands the group places on cloud services is quite basic, so any other provider such as Azure, Google, Heroku etc could be used. The purpose of using the cloud has been simplification of development and the setup of a test environment which is set up to mimic an environment fulfilling all the dependencies to run the time-series cache.

While the use of cloud services is not a requirement for a testing environment, there are in total 5 separate simultaneous services running. In turn, this creates a lot of overhead when developing, since it might take quite a few minutes just to get everything up and running, especially if doing so on older hardware

2.14.4.6 Domain registrars

Since security is an important aspect in services handling confidential data, the group acquired the wirelogger.com domain through a domain registrar. A domain registrar is an entity where one can go to register a domain, leasing it from the registrar [43]. With the acquirement of a domain, the ability to acquiring an SSL certificate from a certificate authority opened. This way, the testing environment can be set up to use encrypted connections, which will be the primary use case for the services. The same can be achieved with self-signed certificates, however, creating an environment as close to a proper production-environment is preferred, and the group thought it would be cool to host the system on a relevant domain.

2.14.4.7 Webserver

Nginx is a widely used webserver for both small-scale and enterprise-scale web-applications. Apache was another webserver the group considered, but Nginx gives off an easier-to-use impression and is, according to hackr.io, 2.5 times faster than Apache (At least with static content) [44]. The Nginx webserver has been used as a TLS terminating reverse proxy for the REST service, website and MQTT broker. This way, the data transmitted between clients and services is encrypted and cannot be read or tinkered with by any unwanted third parties. The approach of using a TLS terminating reverse proxy makes development a lot easier, since each service can handle connections as not-encrypted, completely unaware of the complexities of TLS encryption. This also displays good cohesion on an infrastructure-level, since the responsibility of encryption is both clearly defined and implemented without requiring any adaptation by any other service

2.14.4.7.1 Let’s Encrypt and Certbot

To acquire a TLS/SSL certificate, the Certbot tool was used. Certbot is a tool developed and maintained by EFF (the Electronic Frontier Foundation) to simplify transitioning sites from HTTP to HTTPS. Here, Certbot has been used to acquire the correct certificates from the Let’s Encrypt certificate authority, but also to perform some auto-configuration of the Nginx proxy. These certificates are in addition to securing HTTP used to secure MQTT communication through the Nginx proxy. The Certbot certificates were also used to enable TLS on the Postgres database, however, since Postgres uses a proprietary application layer handshake, this responsibility was put on the server, and not the NGinx reverse-proxy [45].

2.14.5 Development tools

2.14.5.1 GitHub

GitHub was the groups choice for version control mostly due to the group being familiar with it, already using it with most of their development projects. Since the repositories

have mostly been worked on by one person each, the use of branches hasn't been particularly high. However, when exploring options, or performing reworks that are likely breaking, the code has been split into new branches, to ensure that the main branches are stable. Once such a refactor or implementation has been fully implemented, the branch would be merged into the branch it was branched from.

2.14.5.2 IntelliJ IDEA and Visual Studio Code

IDEA and VSC (Visual Studio Code) are two different IDEs. IDEA is made by JetBrains and comes with a lot of functionality out of the box. In contrast to IDEA is VSC, which is developed by Microsoft, which has a lot of the primary features, but is more plugin dependent (Keep in mind that VSC can do the same things as IDEA with a tiny bit more tinkering). The IDEs do, in practice, the same things, and the choice between them was simply from preference. The primary features used throughout the development process are the typical code features like syntax checking, intellisense and code completion, with some more advanced features such as database clients and debuggers also being used throughout development.

2.14.5.3 MQTT Explorer

MQTT Explorer is an MQTT client where you can publish and subscribe to MQTT topics with a simple interface. During the development of both the cache and REST service, MQTT Explorer has been used as an intermediate for testing responses from all services and for observing the activity on the MQTT broker. Since the data traveling across the MQTT broker is serialized by Google Protobuf its utility has been decreased when it comes to seeing the actual data and verifying that the data is as expected, since the data traveling over MQTT is not in a human-readable format. That responsibility has instead been placed on unit testing. MQTT Explorer subscribes to all topics on the broker and thus gives a full overview of which point in the process a request fails. Each request is typically called on the cache in the data_request topic, which is either fully hit in the cache, or partially or fully forwarded to a vessel in a request/vessel/cdplogreader/changes topic. The vessel service would then respond in the response/vessel/cdplogreader/changes topic that the cache would now subscribe to. The cache service then handles the request and forwards data to the client issuing the data_request-request over the DATA topic. Due to these semantics, having an overview of where the flow stops has been a massive help in debugging and testing.

2.14.5.4 Postman

Postman Is like MQTT Explorer, with the main difference being which protocol it is built for, which for Postman is HTTP(S) and WS(s). Postman's use has been testing the REST service, which provides both HTTP and WS endpoints for synchronous requests of certain ranges and asynchronous streaming of contiguous data. Postman can be used to make any HTTP or WS requests, opening for quick testing of endpoints with practically no client-side setup. Therefore, the development of the REST services API has been a smooth ride. Another thing that sets Postman apart from for example using a JS HTTP client is the way data is presented. Most clients built into frameworks or languages do some types of handling and can often be hard to use to debug as a "failure" HTTP code would trigger an exception and losing a lot of data. What Postman does is simply return the raw response, regardless of whether the request is successful or not (of course assuming the Postman client receives a response). This applies to both HTTP requests and WebSockets.

3 Method

This chapter describes the methodologies used throughout the project and how the group has used them.

3.1 Development methodology

3.1.1 Agile

Agile development methodology is a more recently developed methodology which has in many regards succeeded the old waterfall method. While the old waterfall model had a more direct lifecycle, like a river with waterfalls, hence the name, agile methods shake things up by using an iterative approach. This means that the development cycle consists of smaller cycles which are reflected upon as they are completed [46]. During the cycles, as either new issues arise, or new demands come from the customer, they are accounted for going forward. This way, the product will respond quicker to changes in demands, and the product will be developed more in line with what the customer wants. This in turn avoids the issue of the developers misinterpreting the requests and create a program that does not fit the needs. Sprints, which are periods of development, are often short when using agile methods, again for the sake of more responsive processes.

3.1.2 SCRUM

Scrum is a framework for agile development, bringing with it some more specific strategies and roles [47]. While the methodology used throughout the project is not strictly SCRUM, some of Scrum's concepts have been utilized, such as sprint reviews and retrospectives, as well as the SCRUM master role.

3.1.2.1 Sprint review

The group has done bi-weekly sprint reviews at the end of every sprint. During the sprint reviews the group has reflected on what the initial goals of a given sprint was, and which of the goals have been fulfilled, writing the review down, and sending it to the Seaonics team before the meetings [48]. This way, the Seaonics team have been up to date with the project when entering meetings.

3.1.2.2 Sprint retrospective

In addition to sprint reviews, the team has performed sprint retrospectives at the end of the sprints. The sprint retrospectives have had focus on the process-aspect of the completed sprint. Here, the goal has been to figure out what the team has done that is good for the process, what has been not that good, and which things the team could start doing to improve the efficiency of development [49].

3.1.2.3 Sprint planning

At the end of each bi-weekly meeting with Seaonics, the team has sat down immediately afterwards to discuss the coming sprint [50]. During this planning, the group has formulated issues to be slightly more digestible and gone through the backlog to select which issues are the most relevant for progression, often ending in a selection steering towards a goal, such as improving the rigidity and error handling, or more specific goals like implementing certain features.

3.1.3 Iterative development

Iterative development is a key component in agile methods and has been key in the process of creating the time-series cache. The process used in the project has been very iterative, with things being implemented in stages, through sprints, where better, more complex implementations were continuously developed [51]. For example, the data model was initially just the data records which captured time-series data, but then metadata was added improving the insight on cached data. As time went by, the group saw storage-saving opportunities in re-shaping the data model into record, vessel, and variable tables.

3.2 People in SCRUM

3.2.1.1 The customer

The customer in this project is the team of engineers at Seaonics, who will use the service for monitoring of relevant data. Among the customers is also the part of the team that has been involved in this project. Aligned with agile methodology, the wishes of the customers have been propagated to the developers biweekly in the end-of-sprint meetings. Since the opinions of users have been voiced frequently, the process has been able to quickly respond to specific wishes or changes of them quickly by putting the most relevant ones on the agenda for the following sprint. Thanks to this flow of information the project has been guided by the customers and the product has been shaped by them to a large degree.

3.2.1.2 The product owner

The product owner is a representative of the customers who works closely with the development of the product. During this project, it has been one of Seaonics' senior control system engineers who has been available for the team throughout the process, answering questions and helping nail down requirements, in turn helping the maintenance of the product backlog [52].

3.2.1.3 The development team

The development team has, of course, been the students. When writing code, the group has used the product backlogs and selected issues for each sprint to guide the development process [53]. This way, when writing code, the developers have clear goals when it comes to what they need to do or implement. The issues have been delegated to developers based on responsibility, since the project has been split into two components, the front- and back-end.

3.2.1.4 The SCRUM master

The SCRUM master's responsibility is to aid in the process. For the team, this means providing guidance and creating an environment in which the team can work optimally. The SCRUM master's responsibility towards the product owner is to have an overview of the process, so that the product owner can be filled in, but also aiding in setting up requirements for the product backlog [54]. Thus, the main responsibility is to facilitate an efficient development process for all parties involved. Since the group only has two members with clearly split responsibilities, the SCRUM master role has been taken by the group members based on their respective responsibility (frontend and backend). The result became a somewhat blurry role in this project, where the developer was its own SCRUM master.

3.3 Pair programming

One of the parts of SCRUM the group did not utilize were the SCRUM meetings, or daily stand-up meetings. A major reason for this is that the group often utilized an adjusted form of pair-programming where each member was at separate locations but using collaboration tools were able to see the other persons monitor [55]. Therefore, each team-member was always aware of the other team members status. Due to this factor, the group saw daily stand-ups as excessive and most likely a waste of time. Pair programming between the components of the system also made continuous integration easy and more natural, often with endpoints on the server being developed at the same time they were implemented on the website.

3.4 Use Cases and User Stories

For functional requirements, the group has developed use cases and user stories for each service.

User stories are informal descriptions of requirements from different users' perspective that give a higher-level overview of the semantics of use [56]

3.5 Testing

3.5.1 Test driven development

TDD is a methodology within software development where tests are developed either before or along the implementation of logic [57]. This way, the interface of a logical unit is already defined as it is written and the goal of the implementation is often easier to see, since the written tests act as a functional requirement. Keeping TDD in mind when developing software also has the added benefit of making regression testing straightforward. When there are interfaces between components, TDD is also quite useful as tests can be written to test that a unit adheres to the interface regardless of internal change. The TDD approach taken by the team in this project has been quite careful, whereas TDD can often go towards more extreme rules, such as only writing code to make tests work. Calling the project strictly test driven would therefore be wrong, but it has been an inspiration to the approach.

3.5.2 Unit tests

Unit tests are tests written to test a smaller portion of code, or a single logical entity. The main purpose of unit tests is to ensure that the tested logic acts as expected [58]. Since unit tests should be limited to testing a single logical entity, there needs to be some mechanism to eliminate outer variables impact on the tests. This is where mocks come in. Mocking modules overwrite their logic, this logic is usually replaced with some simple mocking of data to make the mocked module act as if it were the actual implementation. Thus, through mocking, you can test how you code acts when there are network issues, or how it acts normally, without having to set up the required resources for proper testing (ex. Database or MQ brokers).

Tests have been developed alongside the implementation of components as requirements were solidified, this way, the group has to a large degree avoided manual testing which has saved a lot of time. Whenever breaking changes have been made, the unit tests have also been updated to ensure that the test results are valid.

3.5.3 Regression testing

Regression testing software is a testing approach where code is re-tested when it is changed to quickly determine if a change has had any negative impacts on the larger codebase, or if the change has broken down the business logic. When regression testing you would run all tests that worked before change was implemented [59]. After running the tests, you can look at the results to instantly discover whether some previously working code has broken due to the latest changes. The relevance of regression testing grows with the size of a codebase and as its inter-dependencies grow in size and numbers.

The development of the cache service has included a lot of tinkering due to seemingly random bugs due to the multiple asynchronous subsystems, as well as frequent tweaking and testing changes for performance. Here, regression testing has been important to ensure that previously discovered bugs haven’t come back and that the internal changes of a module don’t change anything outside it.

3.5.4 Integration testing

Integration testing can be seen as a step up from unit testing where instead of testing the individual units, you combine them and test how they work together [60]. Since most systems consist of multiple sub-systems, frequent integration testing is important to ensure that sub-systems do not diverge from their established interfaces, creating bugs or completely breaking at the point of integration.

The integration testing performed by the group has been manual and consisted of frequent deployments of the newest software to a testing environment. The reason the group went with manual integration tests instead of automatic integration tests is that there wasn’t any real need for it, and the time was best thought saved. In terms of the cache, the unit tests also worked somewhat like integration tests, as they all used actual databases and mqtt brokers for unit tests, so the test code relied on the code working with real databases and mqtt brokers.

3.5.5 Usability Testing

User testing is an important aspect of proofing an application. Automated test suits aim to speed up development and improve the developer experience. However, the way a user interacts with the application can never be 100% predicted when writing test cases, therefore, a test user can find holes in the applications coverage that otherwise would go under the developer's radar. Technical tests can’t really test the usability of a system, which is instead covered by having users try out the software and find where improvements can be made to ease use [61].

During the project, the Seaonics team has had access to the newest code throughout, with some clicking around and testing of the system at the bi-weekly meetings where feedback was provided. Additionally, the group had one of Seaonics senior engineers who was not familiar with the project perform a user test with a test case near the end of the project. More on this test in the results chapter.

3.6 Documentation

3.6.1 Confluence

Confluence is a web-based wiki, created by Atlassian, which can both be deployed on your own hardware but is also available as a cloud service. The group has developed ideas and models using Confluence cloud as the place to store them. Other documentation such as sprint reviews and Gantt-diagrams have also been created and stored in the Confluence cloud using their templates where available.

3.6.2 Jira

Jira is a more project management-oriented tool also created by Atlassian, running on the same platform. Jira has been integral for planning the work ahead of and during sprints, both on the fore planning tasks, but also for planning the sprints themselves.

3.6.3 Draw.io and Miro

Draw.io and Miro are two different web-based tools for drawing figures and modelling. Draw.io has an extensive library of figures for things such as UML (Unified Modelling Language), database modelling, cloud modelling and much more. This made it the groups go-to tool for modelling things closer to convention. Atlassian also has a Draw.io plugin, which allows for creating models without even leaving Atlassian, which makes the process even easier. Miro is somewhat simpler, without a lack of much of the figures, such as UML, which Draw.io has. The thing Miro does better than Draw.io, is collaborative editing, with a snappy and easy setup collaborative mode. Therefore, Miro was used when the group was brainstorming alone or with the team at Seaonics where conventions such as UML were less important.

3.6.4 JSDoc

JSDoc is a standard for documenting JavaScript and TypeScript code. When documenting code with the standard, different tools can be used to generate HTML wikis where all functions, classes etc are documented.

3.6.4.1 Typedoc

While the JSDoc format is mainly for JavaScript code, there are libraries out there that provide more TypeScript oriented approaches for generating documentation. The one used in the Time-Series Cache project was Typedoc. Typedoc goes through all the projects code and generates a HTML site with all the projects documentation, which is used as our code-documentation.

4 Results

This chapter discusses how the solution the group has created solves the problem given by the client. The results will be analysed with respect to the problem statement which can be found in the introduction of the document. A more technical insight can be found in the System documentation attachments. Chapter 4.1 provides an overview and short explanation of the system for context. Chapter 4.2 considers how the produced system improves the availability of vessel-side data for other services and the website. Chapter 4.3 explains how the visualization tools created on the website helps in troubleshooting for Seaonics personnel. Chapter 4 details the caches efficiency in terms of storage. Chapter 4.5 discusses the current response times when using the REST API in terms of records per second. Chapter 4.6 discusses the caches' ability to ensure that data is only fetched once. Chapter 4.7 plays further on the caches ability to minimise data transfer by comparing the old and new method of retrieving vessel-side data.

4.1 Overview

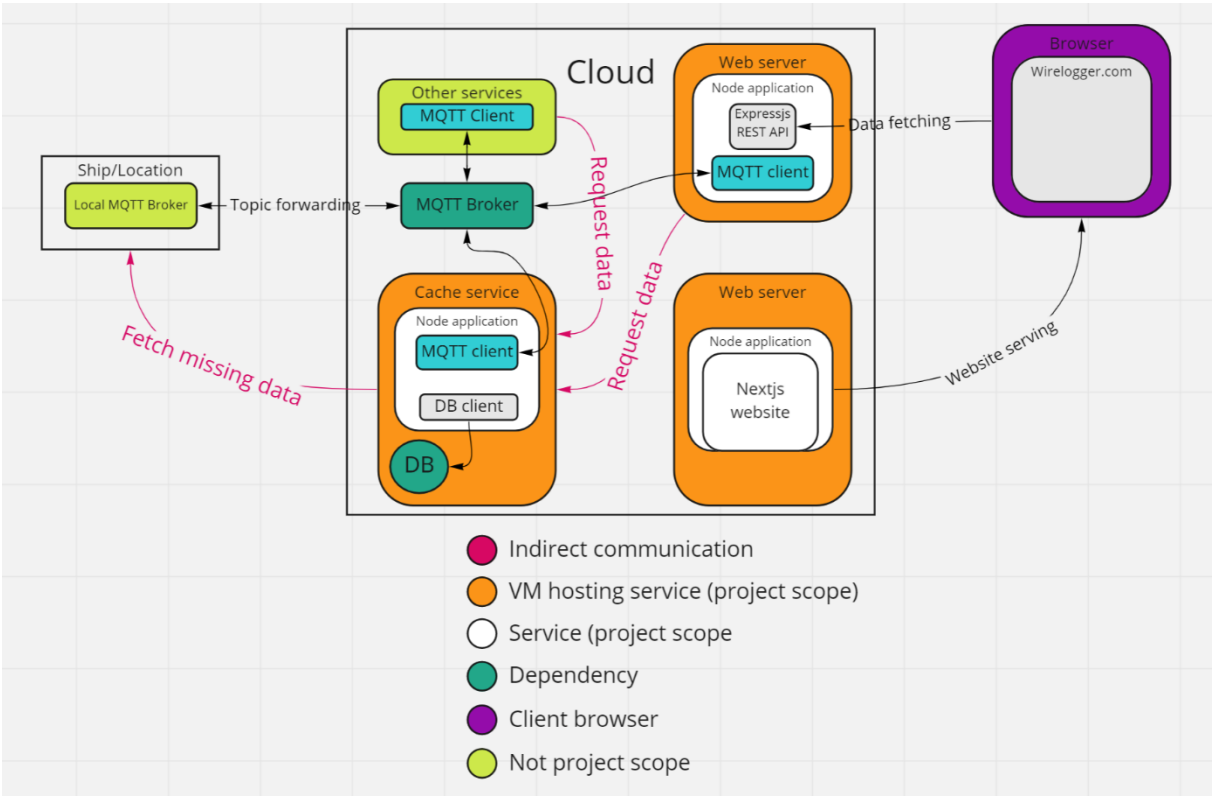


Figure 4-1: System architecture overview

The architecture for the complete system is shown in Figure 4-1. The main component is the Cache service which holds a database and acts as a proxy for all vessels. Communicating with the cache are other services who communicate with it through the MQTT broker. Any service can use its API assuming they have access to the broker. The REST service acts like any other service and communicates with the cache through MQTT. It then exposes an API which acts as an HTTP proxy for the cache and extends its functionality to a webapp. The wirelogger site uses the aforementioned REST service to query time-series data from the vessels and displays the data with different graphs.

This illustration of the architecture is to some extent an abstraction with other components such as TLS terminating proxies most likely being used between these services in a real-world scenario.

Since the project given by Seaonics didn't contain any real technical factors to measure success by, the results are primarily in the context of what one would expect from a generic cache, with some comparisons over the older system Seaonics currently uses where applicable.

4.2 Availability

The time series cache has made the vessels data readily available from anywhere. For users of the system in its current form, this comes down to the quick access with the web interface. Using the interface, a user can find out what data signals are being logged in a matter of seconds, and for debugging, a user can choose the exact time they want to look at by the push of a few buttons.

Another type of client is other services. While the main purpose of the cache has been to create the web interface in this context, it has been developed for further usage too, where other services can be built with the cache as a dependency. Thus, the data is also readily available for a multitude of other use cases through the caches Protobuf-MQTT interface. Optionally, if other services need an HTTP interface, the REST service can also provide the data through the cache.

There is also a somewhat noticeable issue in terms of availability being the time it takes to retrieve data, or response times. The high amount of data is the main reason for this and is arguably not an option to change for a general cache, since it should not change the data for other services wanting to use it. For the sake of keeping the cache as general as possible, this issue has been dealt with by minimizing response times by for example making database insertions run after a request has been dealt with. This should also not be a problem considering the purpose of the website, which is troubleshooting, where loading in hours of data is unlikely. The primary use cases are therefore highly responsive with loading times usually being around a second long, which the group deems satisfactory. This compromise is discussed under "providing data in different resolutions" in chapter 5.1.

The contrast is really shown when comparing this solution with the current solution, where one would have to request large portions of a database with practically no flexibility in what data to request. The old method required a user to log onto a website, create a query and download all the data within certain ranges, relevant or not, which could end up being a whole week of signals, even though a 5 minute interval of one signal is what was initially required. Then, that file had to be opened with a tool for visualization, which would have to have been installed. Now, all a user needs to do is to go onto the website, select one to six signals, then enter a time to view, which after a couple uses takes only a few seconds. The same goes for machine-to-machine interactions, where a machine can now request data down to the millisecond, and single signals, versus massive database files.

4.3 Visualization

When visualizing data there are a lot of features that can be viewed as purely aesthetic, however most of what the group implemented is to help the end-user to identify anomalies and present data in a way that makes it easier to use for displaying results.

For customizing how each dataset should be displayed, the group implemented a visual settings menu for each of the data sets the user has added. These settings can be changed in the per-variable settings shown in Figure 4-2, and consists of the following options:

- Stepped: when toggled on the values of the dataset keep their value all the way horizontally before another value is logged, thus creating a stair looking line where each plateau is a logged value. This option can be more relevant to signals with smaller ranges of values where each value might indicate different types of states.
- Radius: this option indicates how large each point of data should be drawn, when set to 0, data point will not be drawn, this option can be performance heavy based on how many points are available on the screen.
- Line: this option indicates how think the line will be drawn in the chart, this can be used to distinguish datasets from one another.
- Tension: this option will only be available when the stepped option is turned off. When set to 0, the chart will draw straight lines between each point, if greater than 0, a Bezier curve with the selected factor will be drawn between each point.

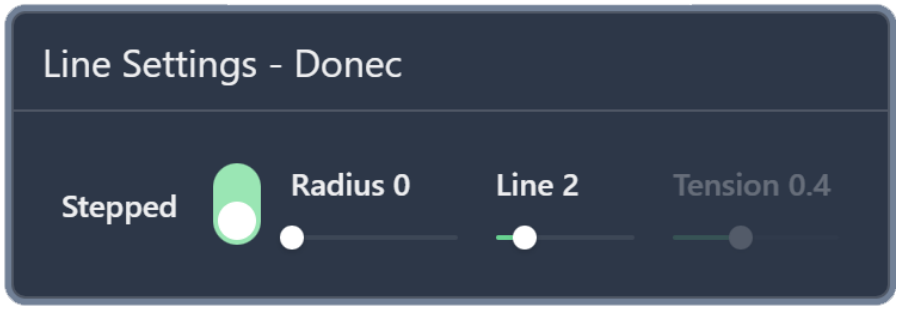


Figure 4-2: Dataset Settings

Additional visualization options are scaling the individual datasets in the y-axis for minimizing the amplitude of the rendering. There also exists a method to offset the datasets in the y-axis, these features were made to make visualization easier when rendering multiple datasets in the same chart.

Colours are defined by the index of the datasets so that the matching y-axis gets same colour based in the index when rendered. These colours where picked to make it easier for colour sensitive people, therefore there does not exist any method for changing these colours yet. The colours and their hexadecimal values are shown in Figure 4-3.

Figure 4-3: Predefined colours for datasets

Visualisation options such as individual scaling of datasets, layering datasets on top of each other even though value ranges vary and offsetting individual datasets in the y-axis where all features added to help the user complete their service tasks in a better way

than the original manual method. As development went on, more features were added per request of our clients.

4.4 Storage

The raw data retrieved from a vessel consists of four parts, the timestamp and value of the record, as well as a reference to the vessel it was read on, and the variable, which will typically be a sensor, or a digital state.

There were two database models considered, one with a straight record including all variables plainly, shown in Figure 4-4.

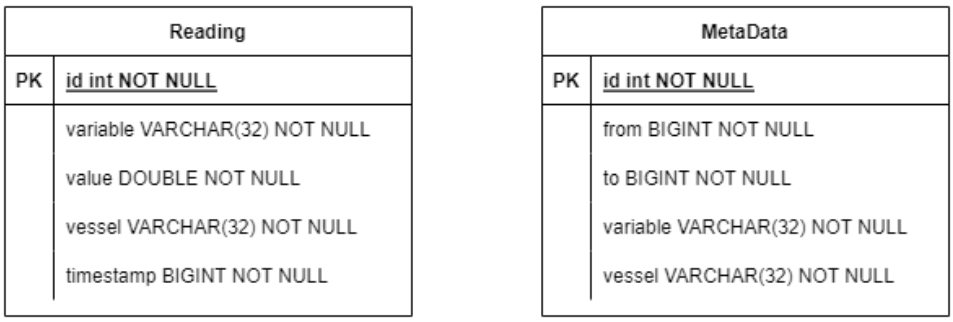


Figure 4-4: Simple data model

The other model used a composite primary key and vessel and variable tables to save some space from repeated 32-byte varchars and is shown in Figure 4-5.

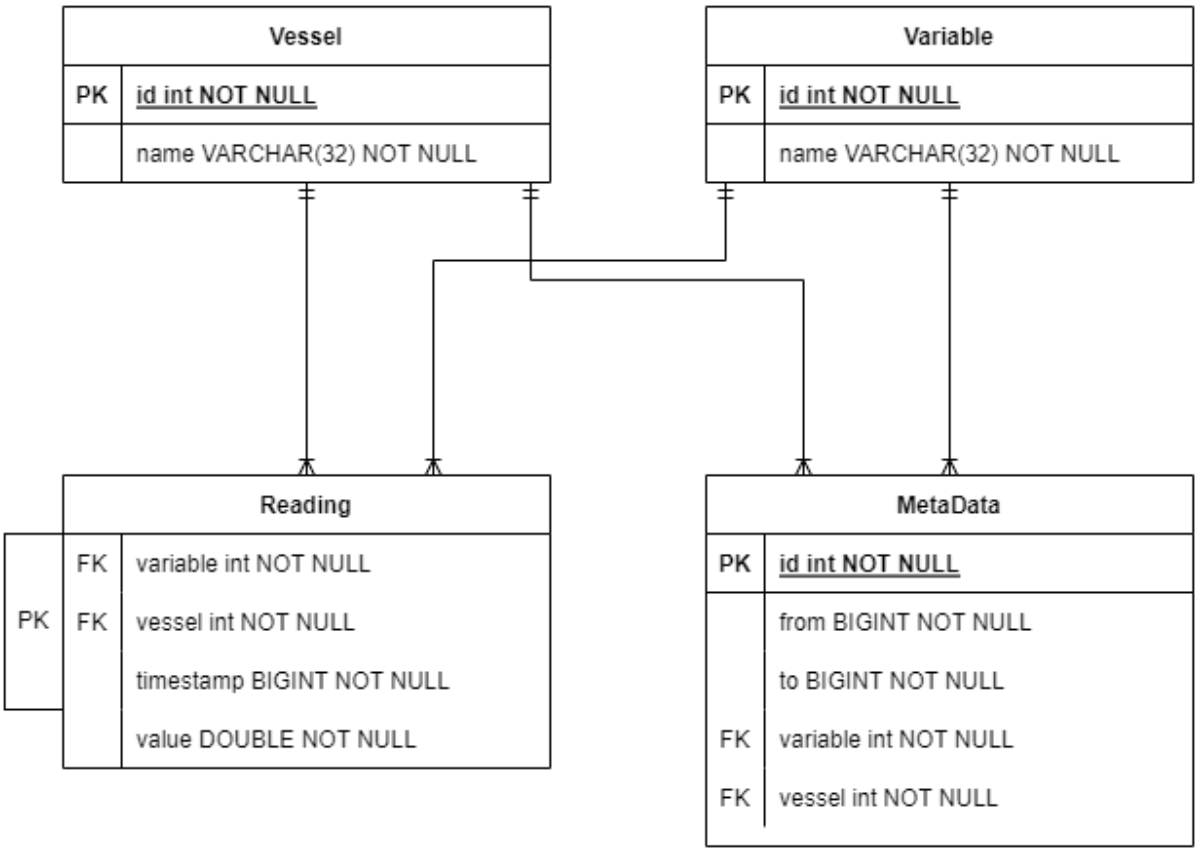


Figure 4-5: Updated data model

The following table shows the space usage of the simpler model, lower bytes per record is better.

Records	Database size (MB)	Bytes per record
0	9.8	
186’000	34	130
461’000	75	141
605’000	94	139

Table 4-1: Storage space test results with simple model

This table shows the space usage of the vessel and variable table model.

Records	Database size (MB)	Bytes per record
0	9.9	
78’500	19	116
161’000	30	125
247’000	39	115
878’000	88	89
6’688’000	580	86

Table 4-2: Storage space test results with updated data model

The data was collected by requesting larger and larger intervals of time from the mocked vessel service, which generates random data as well as random densities of data, which is why the record counts are unique for each table.

From this we see that the new table model, whose results are shown in Table 4-2 has a clear advantage over the simpler model whose results are shown in Table 4-1. Even though there is a lack of datapoints on the original model for larger data sets. Summarizing the amount raw data from the simpler model we see that its raw data alone will be higher per record than the entire database running the new model.

From this data, going with the bytes per record for 6.7 million records, 86B/r, and the information we got from Seaonics, being that there are at most 20 readings per second, we get 1.72kB/s, or 6.2mB/hour. This means that 1GB of storage should be able to hold slightly over 161 hours of raw data (Of a single signal), or 11.63 million readings. Keep in mind that the readings database sizes in the table are of the entire database, which means that this includes the readings, metadata, vessel and variables tables, their indexes, and the overhead from TimeScaleDBs hypertables. All the while taking up less space than the raw data.

One thing to keep in mind when looking at these numbers is the fact that the system is not meant to cache all data, but to cache the data which is relevant to troubleshooting which is likely to be only a few minutes in duration. When keeping this in mind, 161 hours of data should take a lot of time to accumulate.

These tests were performed with 32-byte varchars, that were later changed to 64 bytes, which should not impact the new model, but double the record sizes of the older data model, putting the new model even further ahead of the original one.

With this considered, the group is very satisfied with the resulting storage efficiency, as the data stored in the cache is smaller than the raw data itself, thanks to the use of the Vessel and Variable tables each decreasing the amount of data by 60 bytes.

4.5 Response times / efficiency

One important aspect in handling a lot of data is how efficiently the data can be handled. This was tested by performing requests from Postman, through the REST API and to the cache, which is the primary use-case of the cache out of the box.

This would therefore include the encoding and decoding of all the data with Protobuf, which was a major bottleneck, with just the cache-side encoding taking up about 10-25% of the total time on a request. Including the overhead of Protobuf encoding speeds are shown in Table 4-3.

Fetch type	Records/s
Purely fetch from vessel	42000-43000
Capture fetch in transmission	107000-127000
Purely fetch from database	58000-78000

Table 4-3: Cache + REST performance

This data was found by performing requests of one hour intervals with Postman. This means that the times include the complete loop for an HTTP client, client -> rest -> cache, and back. A record is a single data-tuple (timestamp-value)

This goes back to the compromise mentioned in chapter 4.2, and will be discussed in chapter 5.1. The resulting bandwidth that is achieved by the whole system as a unit is something the group is very happy with. From testing, the performance increase is a full 35% in what is a worst-case for cached data and a best-case for the remote data. Sadly, the group were unable to test the performance increase in realistic scenarios due to time, so even though a 35% increase is solid, in a real-world application of the cache, the difference will be multitudes better. The test environment hosted a mocked vessel-service on the same machine as the cache and MQTT broker, which generated data, thus having both practically no delay from networking, and also having extremely high throughput of data since all data is generated and all time to fetch data is omitted. In addition to this, the server this all ran on was quite slow with a lot of bottleneck issues from the database, which would pin its CPU usage to 100% when requests were performed. These facts show just how efficient the cache is, when its worst-case performance is, in worst case 35% better, but more realistically between 50% and 300% better, again, in the worst-case.

Since the group has been unable to test the current systems throughput it’s hard to draw a comparison between the systems. Then again, from the sheer amount of data the old system would put through, and the fact that the above rates of throughput are quite high, it’s safe to say that the new system is highly efficient in comparison.

For some perspective, in a system where signals are logged at a resolution of 25 milliseconds, a 72000 record/s throughput equals 30 minutes of data per second.

4.6 Fetch once

Through the use of state management and asynchronous events implemented by a publish/subscribe pattern, the cache is able to control data at all three stages of transfer, being cached, and in the database, not cached, as well as data in transfer. Data in the database is quite safe and stable, making it easy to retrieve, and the same goes for data only located in vessels, even though that requires some network usage and time.

Due to the highly dynamic scenarios seen during testing, there's been put in a lot of extra effort to ensure that the cache won't request any interval of data more than once. This has been achieved by centralizing the state of all requests and creating a pub/sub system where the first request of an interval can essentially lock the interval from other requests. The other requests will then place themselves in a queue, waiting for the initial request to complete, leaving the data in a temporary file. Once the temporary file is complete, the cache will stream the data into the database in the background and keep the file available for other requests until the data streaming is complete and no requests are using the file. This mechanism is the reason for the high transfer rate of "Capture fetch in transmission" from the previous sub-chapter, stemming from the fact that local I/O is faster than database querying and vessel-fetching.

4.7 Data-over-wire

While storage size is important, the data-over-wire is also an important aspect when measuring efficiency, as it is one of the main measures of success in the project itself. When comparing the data over wire between the old method of transferring the database files and the time-series cache the difference per data-point is not exactly important, but instead, one can look at the amount of data transfer required to troubleshoot.

4.7.1 Old method

The old method of fetching data was to load in database files from the vessel and put them into a tool to graph the data points. When using this system, there was a lack of flexibility in terms of what data to request. This often led to fairly large files having to be downloaded which both took a lot of time due to the poor network connections at the vessels, but also affected other users of the network since it would be heavily loaded by the streaming of the database.

4.7.2 The Time-Series Cache method

Using the time series cache, data can be loaded a lot more selectively. Instead of having to load hours of data of the entire bank of signals, the cache allows a user to load millisecond intervals of specific variables from specific vessels. When the group performed user tests, the test subject was able to pinpoint the relevant data within only a few minutes getting to know the program, with the end transfer cost (Data over the vessels network) being around 200kB, even after some messing about, playing with the system.

Since the group does not have access to the actual test tools currently in use, the only basis to judge the results on is what has been said about the current process used by Seaonics. The old process would often download a few 100MB of data, which could probably take upwards of minutes just to download, but then also requiring the moving about of files, starting software and so on. With the Time-Series Cache website, the user found the data in a matter of seconds after getting familiar with the site.

Although somewhat anecdotal data, the group sees a definite improvement both in terms of time to troubleshoot, but also network cost, bringing network costs down from hundreds of megabytes to hundreds of kilobytes.

4.8 Testing

4.8.1 Automated Testing

A thorough unit-testing suite has been developed for the cache service with a coverage of around 80%. More information about the test suite and methods used to emulate real use can be found in the system documentation attachment.

There were also set up automated end-to-end tests for the website, however as the functionality of the website became more advanced, writing these tests with Cypress became increasingly difficult.

4.8.2 User Testing

When the group planned to conduct user testing, the idea was to bring in people from Seaonics that had previous experience with the old method of reviewing signal data from vessels. First round of testing was reduced to showing the solution to our main counsellor from Seaonics and displaying the functionality of the website. The reason for this was that the group and the counsellor discovered some new bugs when testing with the test-data provided by the in-house vessel-simulator. A second user test meeting was therefore planned, where a user not familiar with the project would try to go through a test-scenario, using the Time-Series Cache.

For the second round of user testing, the group wanted to give someone working at Seaonics access to the system, who had no knowledge on how to operate it, and let them navigate their way through the website without help. This allowed the group to see any confusing parts of the website and possible changes that would be needed to improve the usage.

The testing was conducted online so that the session could be recorded and could therefore be played back afterwards. The group met with the technical advisor from Seaonics, and a service personnel from Seaonics with previous experience with viewing logging data from remote vessels in related service cases.

After getting familiar with the website and where to go for signal viewing, the test user was given a scenario which the technical advisor had set up, as he had controls to the simulator running on Seaonics servers that were used as the data-source for the user test-scenarios.

The scenario was as follows:

"21-04-2022 10:09 Start of operation",

"Client wants someone to check why gangway (electric crane created by Seaonics) is underperforming"

"21-04-2022 11:08 Client experienced a jump from the telescope while performing AMC"

From these inputs given as a case supplied by a hypothetical client our test user showed how he would monitor this situation by first checking that operation of the crane started at 10:09 by monitoring the signal "CurrSystemState" which is the default signal from Seaonics cranes that indicates where a system is running, with different integer values above 0.

After loading in this signal, the test user was confused to what time they were viewing as for that version of the website, the last 10 logged signals would by default be loaded into the chart after signal selection.

After getting the hang of how to navigate time with the chart and controls the user could find that the system had changed from 3 = Normal Mode to 5 = AMC mode between 10:09 and 10:10.

The next step was to check the jump done by the telescope, and our test user immediately recognized the plus icon for adding additional signals to the chart, after searching for the signal that the user knew would indicate a jump from the telescope, they added it to the chart. After this some minor state management bugs made it so that the site had to be refreshed before the test user could continue.

At this point the test user was starting to get familiar with the scaling and panning functionality of the chart, and there were a lot of positive feedback. The test user then navigated to 11:08 to inspect the telescope position for any anomalies in the data, and they found a clear jump in position.

After completing the scenario that was set up for the testing, Erik started to move the telescope in the simulator, and our test user monitored the signal in real time and got a look into the flexibility of the solution, they could monitor the changes while it was happening, and this looked like an eye-opening moment for our test user. This was due to the old method of reviewing data was constrained to downloading bigger database files over the sat link that would take time. Then reviewing the data in a desktop program one signal at the time.

This was the only scenario related user testing that the group managed to fit into the development cycle, however for each meeting with Seaonics after the user testing, the clients had access to the locally running solution which the group continuously deployed when there where new versions available, and from these meetings there was additional feedback to how the solution worked.

4.8.3 Summary of User Testing

After the second round of user tests, the group noted the following areas of improvement, based on where the user got confused, and where any issues arose:

- *Label formatting on the x-axis*: The test user found it difficult to spot the difference in time when zooming out, as the labels skipped from “mm:ss” - minutes and seconds to “HH:mm” - hours and minutes. Example for this scenario would be when looking at the 33 second of a given time the chart will have 33:10, 33:20 etc. And when zooming out to display more than the 33 second the x-axis might display the 10 minute all the way up to the previous 13 minute, and it would look like this: 10:30, 11:30, 12:30 etc. These where hard to differentiate.
- *What time is being displayed*: The test user found it difficult at times to spot which data the current data was taken from. As on higher zoom levels the x-axis only displayed milliseconds and the date could only be seen at the top of the screen as well as on the label displayed when hovering the line found in the chart.
- *Input field for fetching last “n” datapoints of selected signal*: When the test user was testing out controls found above the chart area, they got confused by how they would benefit from this feature. They questioned how this would yield different ranges of time based on the signals logging frequency as this varies from signal to signal.
- *Chart area size too small*: The test user commented on the chart area size, and that it might be beneficial to expand it to fill all the available screen real estate, especially when viewing multiple signals in one chart. The versions the test user was interacting with had the chart centred to the screen under a set of controls, due to this layout the chart area would only take up a small portion of the screen, leaving a considerable amount of dead-space on the screen.

- *Stability*: During the testing, the cache would at times run into issues making it crash when the datasets grew above 15-minute intervals. While this could simply be fixed by restarting it, it became clear that the fault tolerance had to be improved.

Following this feedback, the group improved upon them by doing the following:

- *Label formatting on the x-axis*: The smallest format of time was changed to always display the current second on higher zoom levels, thus the x-axis will display "ss.SSSS" when zoomed in, then when a certain level is reached the format changes to "HH:mm:ss" as this is more natural to read. When the chart is zoomed out even further, the format reaches its final form; "MM DD HH:mm", as at this level the chart is displaying up to multiple hours, and therefore the date is also relevant to display, example of the last format would be "Apr 3 18:01".
- *What time is being displayed*: This was in some way slightly improved by the fix found above. But the group also put more emphasis on the date selector found at the top of the screen so that it would be clearer that the displayed date was the one they were looking at, and further actions can be taken to display the full date on the ends of the x-axis to make it even more clear what time is being reviewed.
- *Input field for fetching last "n" datapoints of selected signal*: This feature was an old testing input that the group initially utilized for testing the difference between pulling data from a vessel without caching, a request for a range would always be cached but a request for latest points was directly sent to the website. This has no real user-case, so it was stripped and reserved for live-feeding data through the usage of web sockets.
- *Chart area size too small*: This is something the group missed on the first iteration of the development of the chart. Since it was more beneficial for testing to have a lot of surrounding controls and a smaller chart area to see the changes in. However, for actual use a full view just works better in every case. Therefore, the group implemented a full screen mode that renders the controls at the top of the screen in a container with around 5% of the screen height and reserving the rest of the screen for the chart.
- *Improving cache stability:* The cache has since had a considerable overhaul in terms of handling errors, as well as preventative measures, which has turned into the creation of multiple subsystems for tracking and responding to changes in state.

In addition to the more formal user tests, there were frequent, informal tests by the Seaonics team involved in the project. These tests have resulted in a solid visualization tool thanks to lots of feedback both for features, as well as tweaks.

4.9 Downsampling

When rendering timeseries data fetched from the REST API performance is a big issue. Since rendering tens of thousands to millions of data points requires an immense amount of computing power, the data retrieved from the backend was downsampled. The first iteration of downsampling, the group used a plugin for the chart library that tried to do this automatically. It worked for statically viewing the chart, but when complexities such as panning and zooming the chart were introduced, in addition to dynamically hydrating the chart with more data, it became clear this was not sufficient. Thus, the move to manually downsampling the data before plotting it was made.

The group opted to sending all the data retrieved by the cache to the website, and then down sample it in the browser with a down sampling library. More on this in chapter 5.3.2. The library had support for the following downsampling algorithms:

- ASAP: Automatic Smoothing for Attention Prioritizing
- SMA: Simple Moving Average
- LTTB: Larges Triangle Three Buckets
- LTOB: Largest Triangle One Bucket
- LTD: Largest Triangle Dynamic

When first testing the different downsampling algorithms, it seems that LTTB and ASAP were giving of the best performance while still displaying the data in a readable format. For the testing data, ASAP showed off the random numbers in a more readable format that LTTB, however it hid more of the data than what the group deemed necessary. Therefore, the choice became to stick with LTTB as this kept the overall height range of each dataset while still giving the performance needed when displaying up to 5 datasets in the chart.

Further in development it was experimented with what kind of factor was going to be used when decimating the data when there was more than one dataset in the chart. The down sampling library had to parameters, one for the data formatted with x and y values, and then the width of the chart area. However, performance started going down when all datasets where decimated equally since it was decimating them equally. The fix for this problem was to divide the chart width by then count of datasets the user was currently viewing. As to make the downsampling a little bit more intense for each entry. After this, the performance was sustainable, and the hardware needed to view the data was now minimal

4.9.1 Largest Triangle Three Buckets

The LTTB algorithm was the one chosen to use when downsampling data due to its ability to maintain a high-resolution representation of the original data. The way the algorithm works is by taking the first and last points of the given data range, which has to be sorted. It then splits the range up in n-2 intervals, where n is the target resolution. For each interval, it uses the preceding and following intervals to find out which point in the interval will create the largest triangular area. The following Figure 4-6, shows a graph of 500 datapoints downsampled from 8614 points using the LTTB algorithm

Figure 4-6: LTTB downsampling to 500 points [62] [62]

An example of the downsampling that takes place within the rendering of the data can be seen bellow figures, Figure 4-7 and Figure 4-8. From the starting point visible steps can be seen for the sine waves when viewing the graph with the initial current zoom level, if zoom levels are increased further the data will be rendered more precisely and the curvature of the sinewave can be seen in greater detail than before zooming. As the downsampling is based on the width of the chart the sinewave gets more detailed as the data set has more detail to use when it is looked opon closer.

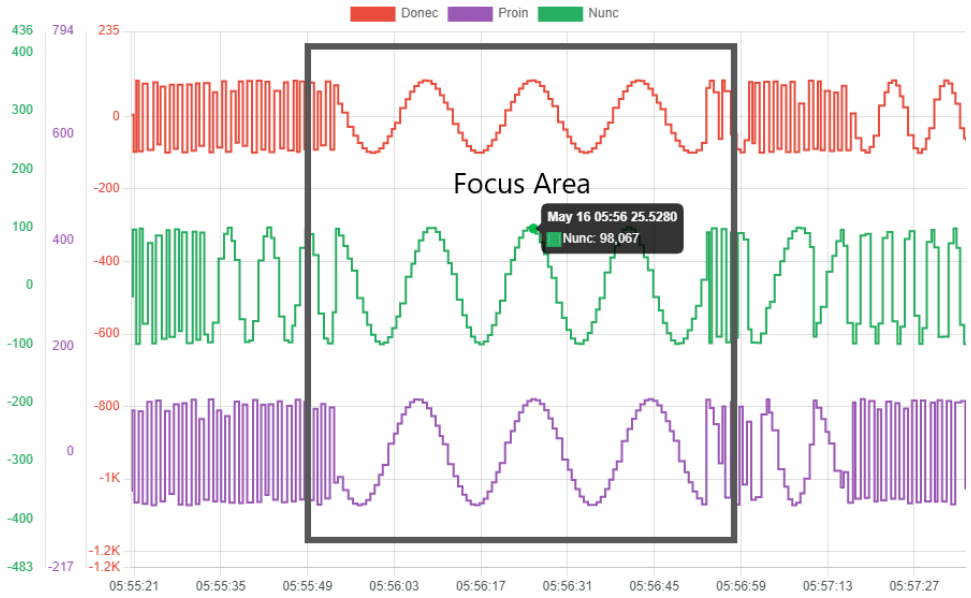


Figure 4-7: Focus Area before increasing zoom levels

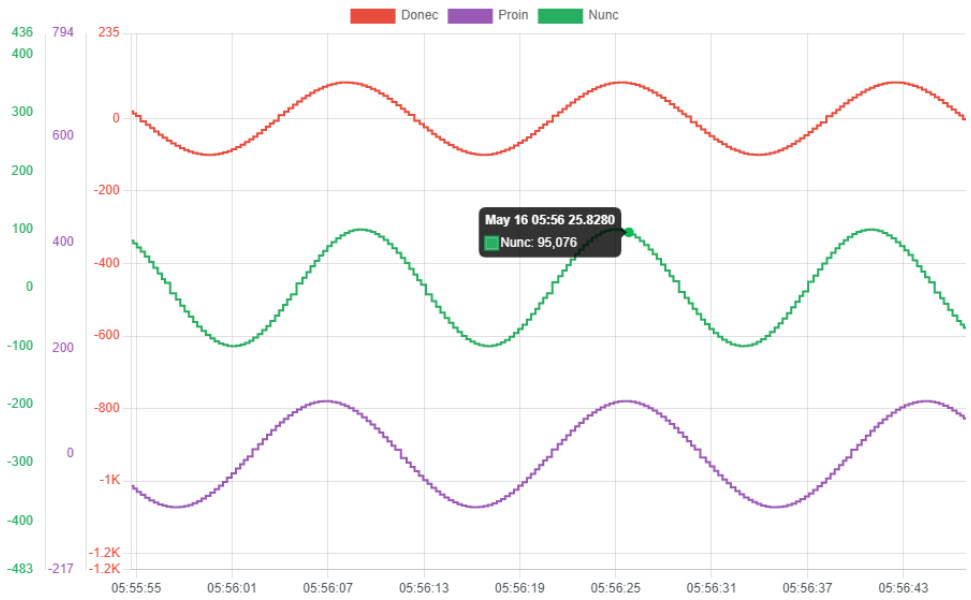


Figure 4-8: Focus area after increasing zoom levels

4.10 Responsive Design

As with most web solutions found today, the Time Series Cache website had to be able to handle all types of screen sizes. Since from our target group could be any one from a service engineer sitting in an office, to someone working offshore trying to figure out what went wrong.

The semi mobile-first approach was used when developing the website. With this comes the usage of media queries. With the power of media queries side bars and other inline components of the website can be hidden in a dropdown menu when viewing at smaller screen sizes.

With Chakra UI there are predefined sizes for the various media query breakpoints, these are based on the width of the screen:

- Small - "sm": 480 pixels
- Medium - "md": 768 pixels
- Large - "lg": 992 pixels
- X Large - "xl": 1280 pixels
- XX Large - "2xl": 1536 pixels

With these breakpoints, a responsive design is created by adding they keywords "md", "lg" etc. To the styling of a component. For example, wanting a Flex component to change from displaying items in a row, to displaying them as a column. In the components given by Chakra UI this could easily be done by creating objects with these breakpoints with them when styling:

FlexDirection={{ base: "column", md: "row" }}

The example above has the base direction set to column, and all sizes above medium will display a row. This is a highly effective way of keeping the UI responsive when developing and creates minimal boilerplate code.

4.10.1 Example of Responsive Design

The following figures show an example of fullscreen mode (Figure 4-9) followed by mobile mode (Figure 4-10), here various flex directions are changed based on screen width as well as font sizes to make reading easeier.

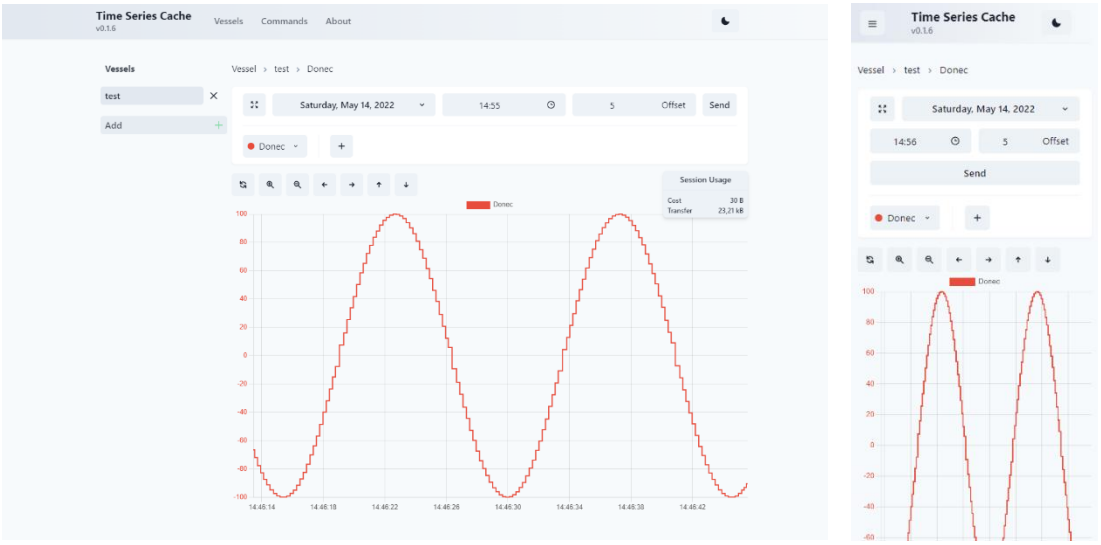


Figure 4-9: Signal Viewer full screen mode vs. Mobile mode

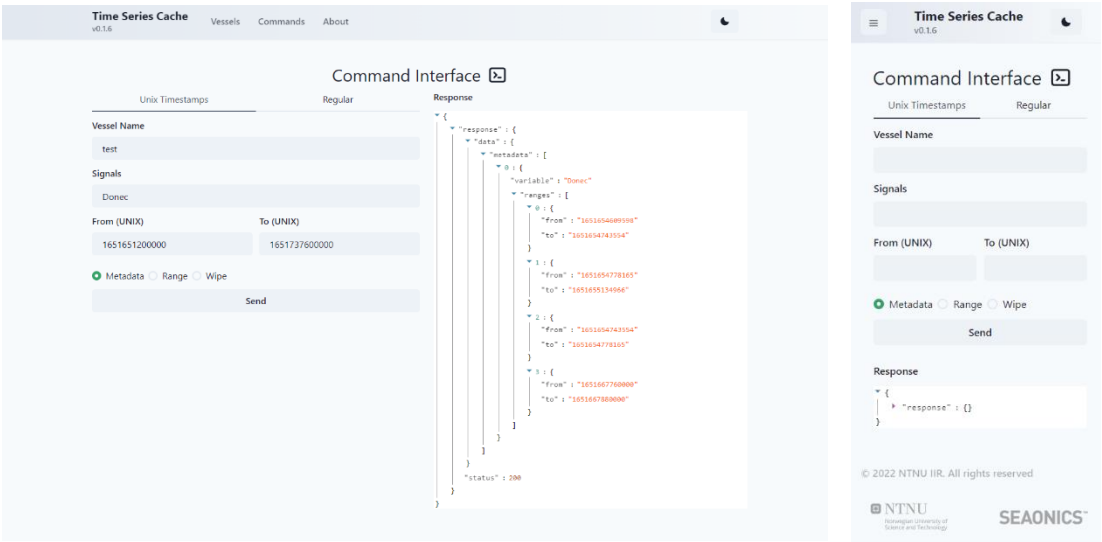


Figure 4-10: Command Page full screen mode vs. Mobile mode

This use of responsive design enables a user to use the Time-Series Cache independently of a workstation and makes the system readily accessible from practically anywhere without having to compromise the user-experience.

4.11 Sharing

4.11.1 URL

An important part of the core functionality of the website was to be able to find anomalies or events in the data, and then be able to share it with others that also had access to the website. This was solved by keeping key information as query parameters in the URLs.

When viewing one or more signals in the signal viewer the path resembles something like this:

```
/vessels/{vesselid}/signal/{signalid}?compare=[signalid]&from={from}&to={to}
```

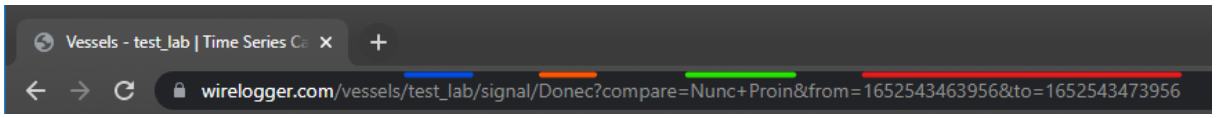


Figure 4-11: Example – Vessel (Blue), Focus Signal (Orange), Comparison Signals (Green), Range (Red)

In the path, shown in Figure 4-11, there is the current vessel that is being reviewed "vesselid" the current signal "signalid" all signals that are being compared in the "compare" query, then lastly the from and to timestamps for what data is being viewed in the chart.

Whenever a new signal is added to the comparison it is added to the query, same goes for navigating the chart. This makes the state of the signal viewer semi persistent and if someone where to send the current URL of what they were viewing, someone else could view would get the same data on their screen.

4.11.2 Export chart as Image

Another feature that comes with rendering data in a canvas element in the browser, is the ability to export the chart as a PNG file by right clicking the chart, then clicking "Save as Image". This feature can for example be used to share findings with others that do not have access to the website.

4.12 Handling REST API calls

When building a web application, the problem of seamlessly calling a REST API becomes a central problem for user interactions. There should be clear indications when something is being loaded, and if the result deviates from normal behaviour actions should be taken to let the user know what is happening.

This starts with communication between the backend and frontend developer, to define how an error should be handled, and what is deemed normal behaviour. Most of this communication is brought down to what HTTP response codes should be used for different deviant behaviours.

4.12.1 Handling side-effects of responses

There are a lot of ways to implement fetching of data when creating web applications that run in the browser. However, the group had previous experience working with a library called Axios that helps streamline the process and makes it more predictable for when a request is timed-out or loose connection to the API. When fetching from, there exist a set of expectations from a user, that the website should indicate when a request is being processed. Then the website should indicate the result of the previously processed request.

The group developed a generalized request handler, that takes in a predefined string key for what request is wanted, together with the needed parameter data as payload.

Another important aspect of fetching data from a remote endpoint from the browser is when a request is called, but the user navigates somewhere else or does some action that makes the result of the request obsolete, it should be cancelled, so that the website does not re-render based on the result of the request as the user is no longer in the place from where the request was called. With this the group implemented an abortion system, this connects an Abortion Controller to all outgoing requests. With this the group could abort calls that were done from a component when it was unmounted in the browser, an example of this would be when a user requests a new range of data, but mid-request they decide to go back to browsing signals. When this happens the wanted behaviour is to abort the request for range, as the website cannot re-render the chart when it is no longer being rendered on the page.

4.12.2 Preventing unsessesary fetching

Building on the previous segment, another important aspect of fetching data, specifically fetching data because of user event as mouse drag, scrolling or navigating with the arrow keys can be quite heavy for the backend. As if the user fires another event while a request is still pending. This will have a confusing result for the frontend as the page will re-render when both requests get their response, as well as the backend must handle both requests.

To prevent this a debounce or more commonly known term rate-limit was implemented. A predefined buffer period is set up on events that might happen close to one another, and for each new event fired in this buffer period will start a new period. This makes it so that only the last event will trigger a request to the backend. This feature has been an important one for battling impatient users, and the urge to move the graph if the backend was too slow to respond. This was especially noticeable when requesting larger ranges of data, where a lot of dragging around and request-making would hog down the cache for longer periods.

5 Discussion

This chapter will discuss some aspects of the project such as further development and how the methodological choices have affected the process.

5.1 Further development

Throughout the development process, the group has designed the system in a modular way to allow for further development where other modules can be added with minimal coupling. With this in mind, the following sub-chapters discuss some features that could be added through modules, as well as some improvements to the current modules.

5.1.1 Users

As mentioned throughout the report, has access control and the concept of users not been implemented, since the system would be brought into another one. If this were to act as an independent system, having some sort of user service would be important. Users would not necessarily have to be implemented by the system itself, but rather a third party access management service, for example firebase. This approach would not require much code and should be quite easy to implement with something as simple as authentication middleware on the REST service.

5.1.2 Providing data in different resolutions

The main drawback of the system in its current state is its performance and response times, from a website perspective, which is a result of the caches API returning raw data. While this is a feature, and not a bug, its impact on the website is very noticeable. One way to fix this is to put more time into the cache itself and have it provide an API of pre-handled data where downsampling is performed either by the database or in the cache, eliminating the long CPU-intensive bouts of encoding and decoding data with Protobuf, and also massively reducing the amount of data flowing through the MQTT broker, saving a lot of time and bandwidth. The TimescaleDB was chosen for this purpose, but the group has not had time to investigate these features but will do so down the line. When the system has been tested, it has been shown that encoding of data can stand for up to half the time a request takes. Transferring all the data also strains the MQTT broker, and while the group hasn't measured how much time goes to just transferring data, it is likely significant.

5.1.3 Distributed

One direction the cache service can take is toward a distributed service, where multiple instances of the cache can work together. As seen throughout the development, data quickly adds up and becomes a major bottleneck, so having multiple separate instances could massively improve the systems scalability, which is currently limited with no horizontal scalability and limited vertical scalability. If the system in its current state is deployed in multiples, it would simply lead to a lot of duplicated responses since each cache would try to handle a request and do so at the same time.

5.1.4 Statistics and analysis

The group sees the potential benefit of having a module which performs different types of statistical analysis for finding outlier values to not only enable for troubleshooting and manual monitoring, but also proactively looking for problems. Statistics can also be used

to create an overview of how systems change over time which could be beneficial during development.

5.1.5 Notifications

While notifications are quite far out of scope for the current project, a lot of the groundwork is already set up, with vessels always being connected to the cache. Expanding the systems API where vessels themselves have the ability to distinguish between normal statuses or trends and potential errors could in turn introduce notification systems where errors on vessels would propagate to emails or SMS messages.

5.1.6 Website

Some additional features that the client wanted for the website, was an innovative way of viewing different timelines in a horizontal format, where the available cached intervals of time would be displayed so that a user could more effectively view what parts of time a signal had been previously cached. This feature could be extended into letting a user select intervals on this horizontal timeline and request that the data to be cached in the background, this making it the data more accessible in the future.

Another feature that was mentioned in one of the last meetings was the feature to be able to add comments for others and connect them to a timestamp withing a signal. So that others viewing the data in the same frame of time could see previous activity.

This could be seen as some sort of messaging board for users with access. And a user could then theoretically navigate all comments of a signal to view the data in from where the comment originated.

5.1.7 HCI

The result that the group ended up with strictly focused on keeping HCI in the same pattern as websites has done for some time now. There were still aspects of the website that was questioned when the first user tests were conducted. As the easiest part of following good HCI practises related to page layout and button positioning. Aspects such as formatting of date time of the x-axis of the chart fell under our radar when first developing the user interface. However, the group kept good UI standard that made the confusing parts small and easy to fix when discovered.

An interesting part of developing user interfaces is always how natural an interface can feel when using it for testing every day for an extended period, then watching a test user try to complete tasks, and needing help on every other step of the process. However, it is important not to lose faith in oneself, as this is all a part of the learning process.

5.1.8 Splitting state-management

As a wanted feature of the website was sharing, a problem that arose was what was the minimal information needed to share the content from one user page to another with the use of link-sharing.

As the first versions of the signal viewer did not take this problem into account. All the variables needed to display the current time and signals where all existing purely in the memory of the browser.

However, the group found a way to keep track of variables such as the current comparison signals as well as the min and max time stamp displayed in the chart. This was all that was needed for the browser to regain the state that one user was viewing if another clicked the link with these same variables in the URL.

The last version of the website had a result that was deemed satisfactory to the wanted functionality. However, for further development, more of the state that currently resides in the memory of the browser could also be brought into the URL. As this makes sharing an even more personal experience. Though this can come at the cost of creating a cluttered URL for the website, something that can be seen as a negative by other developers.

5.2 Problems

When working with any type of project there will be problems along the way. In this segment there will be a summary of problems the group faced while developing our solution for the Time Series Cache.

5.2.1 Problems with ChartJS

For the frontend development, a lot of problems arose when trying to bend the functionality of ChartJS to the wanted outcome. Out of the box, ChartJS will animate a dataset to a defined chart type. And then provides the tools to define how it should look, and what information the hovering tooltip should display. This was not even close to what the group needed as for functionality for a graphing tool.

A big part of the group's selling point was to let the user pan left or right in in the chart area, and then request additional data to fill the new gap created when panning. With this the chart needed to be interactable, so that the user could in a sense navigate time with pan and zoom actions. The idea of offsetting a dataset in the chart based on mouse drag events seemed easy at first. However, this took quite a bit of learning and searching the depths of GitHub issues raised on the official repository of the ChartJS Library before figuring out how this could be achieved.

The same goes for a lot of the wanted functionality the group wanted to get out of the chart, and the solution to most of these problems where third-party plugins, created by others seeking a similar solution as the group needed. However, as with most fixes there were drawbacks, The plugins the group deemed as features did not work well with together. It seemed that on their own, a ChartJS instance and 1 or 2 plugins usually worked but trying to do everything with plugins would simply break everything.

Because of what the group learned with the plugins, all new features that were requested regarding the functionality of the chart were always developed to outside ChartJS if possible, and if the result were underwhelming, the corresponding plugin was testing to see if it would work with the project.

5.2.2 Problems with Docker image build when using NextJS

When working with NextJS as a frontend routing library, all seemed well on the local development and testing front. However, later in the process when it was time to deploy the website with docker, it was discovered that the NextJS framework loads environment variables when the code is built. For this reason, the website has to be built from the bottom whenever the docker container is run, adding a large overhead on start-up. This was a problem the group researched for a while with seemingly no way to get around. The current site is planned to be implemented into an existing system as a module. Once this step is taken, this issue could be solved by changing to another framework such as Vite [63] which allows environment variables to be set when starting up the program.

5.3 Process

This chapter discusses the process of the project.

5.3.1 Communication with client

At the start of development, the group aimed to pinpoint the scale and scope of the assignment given by the client. However, as it is with a lot of development projects, the scope was hard to put into writing as the client did not have any prior experience working with students studying Computer Science. When development started, and the client got a better understanding of what the group was capable of, the requests for features became more specific.

5.3.2 Agile

As previously mentioned, the groups approach to methodology was agile and loosely based on SCRUM. This approach has worked well, with a flexible path where features and bugs were fixed at the same time, allowing the group to continue progressing steadily and solving issues as they arose and proved problematic.

Over the course of the project, thanks to the agile approach, the Seaonics team has had access to the more recent, working versions of the code, which has enabled a fluid process of discovering what functionality is wanted. Since the projects definition was so open, this has been essential for the group to home in towards a system solving Seaonics' problem the best. It has also given the group creative freedom to implement features they have seen fit, and quick responses and opinions at the bi-weekly meetings, or through other dialogue in between.

5.3.3 Distribution of work

The approach of distributing the work by front-end and back-end has paid off immensely, with no issues in terms of work colliding. The frequent communication between group members has also contributed to quick integrations whenever new features were implemented in the backend as well as a clear target to work towards in terms of back-end functionality.

6 Conclusion

The goal of the project was to develop a cache to interface with remotely located equipment, typically being installed on vessels where internet connections could be both slow and expensive. Complimenting the cache, an interface visualizing the data for troubleshooting was to be developed. Since the task was open, the group were allowed to design and implement the system with a high degree of freedom.

The result was a modular, micro-service, system which can be implemented into another system, or as its own system. This approach has also set the system up for future development both in terms of additional features, but also scalability through distribution of services.

The cache itself is a solid base for further improvements. The cache contains multiple state managers to improve both the speed of the cache and ensuring that the cache only retrieves data once, which is highly important when limiting the amount of data transfer. Additionally, the cache uses hashing to protect the data located in the caches database, so that if the database is ever compromised, the compromised data lacks any context, therefore also any value. The caches implementation is defined by high cohesion through an approach where requests are implemented through handlers, and other utility functions performing actions that are repeated. Due to the complexity of the state, some coupling exists, but most logic relating to state is implemented through different state-managers.

The REST service provides a simple to use HTTP/REST API for web clients wanting to use the cache. The approach taken in the REST service is similar to the cache, with a handler approach to endpoints. Since the REST service itself only acts as a proxy for the cache, it has no need for state managers, it has maintained loose coupling.

The website builds on the simplicity of the REST service, to provide an easy-to-use interface for accessing the data from the remote vessels, while giving the user a range of visualization options for investigative work. It puts focus on the visualization component of the website to make it easy to implement as a component in Seaonics´ own web solution.

In summary, the system provides a simple to use interface where users can instantly retrieve and visualize data from vessels, which is what the group set out to do. In addition to all core features, which were successfully implemented, additional features such as the ability to perform command(s) were also implemented and will be built upon further.

Social impact

The social impact from the Time-Series Cache comes from the improvements in uptime of services where time-series data is used for troubleshooting in the case of Seaonics. An example in the context of Seaonics is the marine fleets efficacy and avoidance of pollution caused by malfunctioning cranes and winches requiring replacements.

Due to the general nature of the implementation, the system can also be used for other systems, such as weather stations, cars, medical equipment etc, or anything accumulating time-series data with an internet connection. Thus, the primary benefit of the Time-Series Cache system is its configuration-free setup, not requiring loads of work to get up and running, other than client-side coding since that was omitted from the scope.

References

[1] e. a. Fielding, "ietf.org," June 1999. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2616. [Accessed 28 January 2022].

[2] R. J. Coppen and R. J. Cohn, "oasis-open.org," 29 October 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. [Accessed 28 January 2022].

[3] F. &. Melnikov, "ietf.org," December 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6455. [Accessed 28 January 2022].

[4] R. T. Fielding, "gbiv.com," 2000. [Online]. Available: https://roy.gbiv.com/pubs/dissertation/top.htm. [Accessed 28 January 2022].

[5] E. T. Bray, "ietf.org," March 2014. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7159. [Accessed 28 January 2022].

[6] D. &. Rescorla, "ietf.org," August 2008. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5246. [Accessed 28 January 2022].

[7] C. Richardson, "microservices.io," [Online]. Available: https://microservices.io/.

[8] "microsoft.com," [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar. [Accessed 16 May 2022].

[9] Refactoring.Guru, "refactoring.guru," [Online]. Available: https://refactoring.guru/design-patterns/proxy. [Accessed 16 May 2022].

[10] Refactoring.Guru, "refactoring.guru," [Online]. Available: https://refactoring.guru/design-patterns/observer. [Accessed 28 March 2022].

[11] Refactoring.Guru, "refactoring.guru," [Online]. Available: https://refactoring.guru/design-patterns/singleton. [Accessed 28 March 2022].

[12] Next.js, "Routing: Introduction | Next.js," Vercel, [Online]. Available: https://nextjs.org/docs/routing/introduction. [Accessed 06 03 2022].

[13] "w3schools.com," [Online]. Available: https://www.w3schools.com/js/js_history.asp#:~:text=JavaScript%20was%20invented%20by%20Brendan,JavaScript%20for%20the%20Firefox%20browser. [Accessed 09 May 2022].

[14] "Components and Props - React," [Online]. Available: https://reactjs.org/docs/components-and-props.html. [Accessed 06 03 2022].

[15] Oracle, "oracle.com," [Online]. Available: https://www.oracle.com/database/what-is-database/. [Accessed 28 March 2022].

[16] "postgresql.org," [Online]. Available: https://www.postgresql.org/docs/current/indexes.html. [Accessed 28 March 2022].

[17] "postgresql," [Online]. Available: https://www.postgresql.org/docs/current/ddl-constraints.html#DDL-CONSTRAINTS-FK. [Accessed 28 March 2022].

[18] vmware, "vmware.com," [Online]. Available: https://www.vmware.com/topics/glossary/content/virtual-machine.html. [Accessed 28 March 2022].

[19] L. School, "Introduction to Programming with JavaScript," https://launchschool.com/books/javascript/read/introduction, 2022.

[20] R. Dahl, "github.com/nodejs," 27 May 2009. [Online]. Available: https://github.com/nodejs/node-v0.x-archive/releases/tag/v0.0.1. [Accessed 14 May 2022].

[21] "W3Techs.com," 08 May 2022. [Online]. Available: https://w3techs.com/technologies/details/cp-javascript. [Accessed 08 May 2022].

[22] "mozilla.org," 18 February 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop. [Accessed 08 May 2022].

[23] "scalegrid.io," 4 March 2019. [Online]. Available: https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/. [Accessed 08 May 2022].

[24] The Internet Society (ISOC), "internetsociety.org," October 2015. [Online]. Available: https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf. [Accessed 16 May 2022].

[25] InfluxData, "influxdata.com," [Online]. Available: https://www.influxdata.com/what-is-time-series-data/. [Accessed 03 February 2022].

[26] M. Kuhn, "cl.cam.ac.uk," 16 June 2020. [Online]. Available: https://www.cl.cam.ac.uk/~mgk25/iso-time.html. [Accessed 18 May 2022].

[27] Universal Design, "Universaldesign," [Online]. Available: https://universaldesign.ie/what-is-universal-design/the-7-principles/. [Accessed 08 May 2022].

[28] "What is Human-Computer-Interface," [Online]. Available: https://www.interaction-design.org/literature/topics/human-computer-interaction. [Accessed 17 05 2022].

[29] D. Todorovic, "Scholarpedia," 11 December 2008. [Online]. Available: http://www.scholarpedia.org/article/Gestalt_principles. [Accessed 08 May 2022].

[30] "Consistency Article," [Online]. Available: https://blog.prototypr.io/consistency-a-key-design-principle-5d125469da8e. [Accessed 18 05 2022].

[31] "Design Choices," [Online]. Available: https://uxdesign.cc/improve-user-response-selection-and-choices-simplicity-our-hand-and-eyes-coordination-follows-cf75ca52191d.

[32] "Design principles: Forgivness," [Online]. Available: https://uxplanet.org/design-principle-error-forgiveness-1495f7471113. [Accessed 18 05 2022].

[33] T. Munzner, "Visualization Analysis and Design," https://www.amazon.com/Visualization-Analysis-Design-AK-Peters/dp/1466508914, 2014.

[34] "Linear Vs. Logarithmic Scales," [Online]. Available: https://study.com/academy/lesson/linear-vs-logarithmic-scales.html. [Accessed 17 05 2022].

[35] Atlassian, "atlassian.com," [Online]. Available: https://www.atlassian.com/git/tutorials/what-is-version-control. [Accessed 28 January 2022].

[36] S. C. Johnson, "wolfram.schneider.org," 26 July 1978. [Online]. Available: https://wolfram.schneider.org/bsd/7thEdManVol2/lint/lint.pdf. [Accessed 16 May 2022].

[37] D. J. Barnes and M. Kölling, Objects First with Java, Pearson, 2016.

[38] J. Muittari, "Theseus," 2022. [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/336520/Joel_Muittari.pdf?sequence=2. [Accessed 20 January 2022].

[39] "Chakra UI," [Online]. Available: https://chakra-ui.com/. [Accessed 17 05 2022].

[40] "Chart.js," [Online]. Available: https://www.chartjs.org/docs/master/.

[41] "db-engines.com," May 2022. [Online]. Available: https://db-engines.com/en/ranking_trend/system/InfluxDB%3BPrometheus%3BTimescaleDB. [Accessed 08 May 2022].

[42] B. Mishra, B. Mishra and A. Keretsz, "bevywise.com," 14 September 2021. [Online]. Available: https://www.bevywise.com/wp-content/uploads/2022/02/energies-14-05817-v2-1.pdf. [Accessed 21 February 2022].

[43] Cloudflare, "cloudflare.com," [Online]. Available: https://www.cloudflare.com/learning/dns/glossary/what-is-a-domain-name-registrar/. [Accessed 16 May 2022].

[44] S. Hari, "hackr.io," 01 March 2022. [Online]. Available: https://hackr.io/blog/nginx-vs-apache. [Accessed 05 March 2022].

[45] A. Karlsson, "postgresql.org," 13 December 2018. [Online]. Available: https://www.postgresql.org/message-id/d05341b9-033f-d5fe-966e-889f5f9218e5%40proxel.se. [Accessed 06 March 2022].

[46] Atlassian, "atlassian.com," [Online]. Available: https://www.atlassian.com/agile. [Accessed 17 May 2022].

[47] scrum.org, "scrum.org," [Online]. Available: https://www.scrum.org/resources/what-is-scrum. [Accessed 17 May 2022].

[48] scrum.org, "scrum.org," [Online]. Available: https://www.scrum.org/resources/what-is-a-sprint-review. [Accessed 17 May 2022].

[49] scrum.org, "scrum.org," [Online]. Available: https://www.scrum.org/resources/what-is-a-sprint-retrospective. [Accessed 17 May 2022].

[50] scrum.org, "scrum.org," [Online]. Available: https://www.scrum.org/resources/what-is-sprint-planning. [Accessed 17 May 2022].

[51] A. Kochar, "distantjob.com," 28 October 2021. [Online]. Available: https://distantjob.com/blog/iterative-development/. [Accessed 17 May 2022].

[52] K. Schwaber and J. Sutherland, "scrumguides.org," 2020. [Online]. Available: https://scrumguides.org/scrum-guide.html#product-owner. [Accessed 17 May 2022].

[53] K. Schwaber and J. Sutherland, "scrumguides.org," 2022. [Online]. Available: https://scrumguides.org/scrum-guide.html#developers. [Accessed 17 May 2022].

[54] K. Schwaber and J. Sutherland, "scrumguides.org," 2020. [Online]. Available: https://scrumguides.org/scrum-guide.html#scrum-master. [Accessed 17 May 2022].

[55] B. Böckeler and N. Siessegger, "agilealliance.org," 15 January 2020. [Online]. Available: https://martinfowler.com/articles/on-pair-programming.html. [Accessed 23 April 2022].

[56] M. Rehkopf, "atlassian.com," [Online]. Available: https://www.atlassian.com/agile/project-management/user-stories. [Accessed 12 February 2022].

[57] J. Acosta and K. Gajda, "ibm.com," [Online]. Available: https://www.ibm.com/garage/method/practices/code/practice_test_driven_development/. [Accessed 01 February 2022].

[58] T. Hamilton, "guru99.com," 16 April 2022. [Online]. Available: https://www.guru99.com/unit-testing-guide.html. [Accessed 17 April 2022].

[59] T. Hamilton, "guru99.com," 17 March 2022. [Online]. Available: https://www.guru99.com/regression-testing.html. [Accessed 17 April 2022].

[60] T. Hamilton, "guru99.com," 16 April 2022. [Online]. Available: https://www.guru99.com/integration-testing.html. [Accessed 17 April 2022].

[61] T. Hamilton, "guru99.com," 30 April 2022. [Online]. Available: https://www.guru99.com/usability-testing-tutorial.html#1. [Accessed 17 May 2022].

[62] M. Drolc, "github.com/pingec," 01 September 2018. [Online]. Available: https://github.com/pingec/downsample-lttb. [Accessed 14 May 2022].

[63] "Vite," 2019. [Online]. Available: https://vitejs.dev/. [Accessed 04 05 2022].

[64] Mozilla, "mozilla.org," [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_programming. [Accessed 28 March 2022].

Appendix

Appendix 1	Pre-project report
Appendix 2	Requirement documentation
Appendix 3	System documentation

Appendix 1

Time-series cache

Forprosjektplan

Versjon 1.1

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
14/01/2022	1.0	Førsteutkast av prosjektplan	David, Einar
25/01/2022	1.1	Endring av hovudaktiviteter grunnet endring i omfang	David

Innholdsfortegnelse

Begreper og forkortelser

Backend - Programvaren som er kjørt sentralt hos Seaonics

Frontend - Programvaren som kjøres lokalt me klientene, en nettside

SCRUM - Ett rammeverk for styring av utviklingsprosjekt

MQTT - Message Queueing Telemetry Transport, en data-orientert nettverksprotokoll

API - Application Programming Interface, ett grensesnitt som to parter kan kommunisere mellom

SSR - Server Side Rendering, egenskapen til å bygge opp nettsiden hos serveren, fremfor klienten

MVP - Minimum Viable Prototype, ett produkt som ikke nødvendigvis er klar til bruk, men som i det minste kan fungere som en demonstrasjon av konseptet

1. Mål og rammer

1.1 Orientering

Time-series cache oppgaven kommer fra Seaonics, hvor Einar tidligere har jobbet. Etter å ha hørt om oppgaven og hva hvilke ulike problemstillinger gruppen kan få, har de komt frem til at oppgaven er relevant til hva vi kan komme til å gjøre etter fullført utdanning. Ved å ha utført en slik oppgave og vist frem vår kompetanse mener gruppen at destiller i en bedre posisjon etter utdanning, ettersom de vil ha erfaring fra ett høyt relevant utviklingsprosjekt. I tillegg vil oppgaven kreve at gruppen lærer hvordan de kan utnytte teknologier som ikke er nevnt gjennom studieløpet, og dermed vise til god læringsevne.

1.2 Problemstilling / prosjektbeskrivelse og resultatmål

Seaonics' utstyr som er utplassert hos kunder loggfører ulike system-data i sanntid. Denne dataen er viktig å ha snarlig tilgjengeli når utstyret har feil eller mangler som må diagnostiseres og løsest. Dagens metoder krever overføring av relativt store mengder data og filer, hvor mye er urelevant, som gjør prosessen tregere samtidig som det plasserer mye stress på nettverket hos fartøyene.

Problemstillingen gruppen står ovenfor er dermed at Seaonics trenger ett mer effektivt system som forenkler og effektiviserer prosessen av å innhente og analysere diagnostiseringsdata. Dette systemet må både gjøre dataen enklere tilgjengelig og forbedre prosessen av å diagnosere problem gjennom illustrasjoner av den. I tillegg vil deler av loggdataen være data Seaonics' kunder ønsker å holde internt og avlukket for utenforstående. Grunnet dette aspektet vil sikkerhet og autentisering bli en viktig problemstilling gruppen må ta for seg. Siden dataen Seaonics vil uthente er plassert hos fartøy som ofte har dårlige nettverksforbindelser vil netverksbegrensinger være av spesielt stort fokus. Gruppen vil måtte løse probleme om å ha dataen mest mulig tilgjengelig til en hver tid, uten å overlaste sat-link tilkoblingene hos fartøyene.

Resultatmålet er å opprette en fullstendig tjeneste bestående av en backend og en frontend. Backendens ansvar er å holde relevant data lett og raskt tilgjengelig samtidig som den minimerer nettverksbruk på fartøyene. I tillegg må backenden håndtere tilgangsstyring slik at kun tillatte personer har tilgang til de forskjellige ressursene. Frontendens ansvar er å

representere dataen fra backenden på best mulig vis for å forenkle prosessen av å se gjennom datapunktene for å finne feil

1.3 Effektmål

Hovudsaklig vil effekten av prosjektet dreie seg om “quality of life” gjennom en modernisering av det gamle systemet som har blitt brukt. Denne effekten vil gruppen oppnå gjennom å etablere ett mer moderne sanntidsverktøy Seaonics kan bruke under feilsøking og feilløsing. Dette fører til mindre nedetid av fartøyene som bruker systemet, som igjen bidrar med å effektivisere flåten. Effektivisering av flåten minker kostnadene og bidrar til mindre utslipp ettersom utslipp forbundet med reparasjon kan i noen grad bli unngått.

1.4 Rammer

Ved utfylling av forprosjektplanen har vi ingen spesialbehov mhp penger eller materialer.

Vi har heller ingen spesielle behov for rom i form av fysisk arbeids areale, Seaonics har uansett tilbudt seg å sette opp et arbeidsområde hvor vi kan møte fysisk på deres lokale og jobbe med relativ kort vei til hjelp.

Programvare

Jira / Confluence
Tilbudt gratis gjennom atlassian cloud

GitHub
Gratis

Intellij idea
Lisens fra NTNU

Visual Studio
Gratis

Maskinvare

Gruppen stiller med egne datamaskiner for utvikling

2. Organisering

Gruppen

Oppgaven er hovudsaklig server-side orientert der fokuset er å gjøre data tilgjengelig. Vi ser derfor at ansvaret mest sannsynlig må fordeles litt på kryss av hovedansvar. Hovedsaklig vil David Rise Knotten fungere som hovedansvarlig for server-side utvikling, medan Einar Andreas Aglen vil være hovedansvarlig for UI design og utvikling. Ettersom vi kun er to medlem er det vanskelig å sette opp roller slik scrum metoden vanligvis vil ha, slik som for eksempel scrum master. Isteden vil ansvaret bli hovedsaklig allokert basert på hovedansvarene til gruppemedlemmene. Metodikken gruppen vil bruke blir altså en tilpasset scrum metode hvor de begge vil ha ett gjensidig ansvar for å opprettholde backlogen, dokumentasjonen etc, samt organisering av arbeidet.

Veileder

Veileder og Programansvarlig for BIDATA, Girts Strazdins

Oppgavegiver

Seaonics

3. Gjennomføring

3.1. Hovedaktiviteter

I starten vil hovedfokuset ligge på planlegging av systemer og valg av teknologi for å hjennomføre oppgaven. Dette vil gruppen gjøre hovudsaklig selv, men med innspill fra veiledere og oppgavegiver.

Oppgaven kan deles i to kategorier, frontend og backend. Hvor David vil jobbe primært med backend, og Einar primært med frontend.

Backend

Målet for backend-utviklingen vil være å opprette to tjenester som sammen skal fungere som en enkelt tjeneste for å holde data tilgjengelig. Den ene tjenesten vil kjøre lokalt hos fartøyene og har dermed direkte tilgang til loggført data lokalt. Den andre tjenesten vil kjøre i skyen, og vil da fungere som en data-cache for logg-data.

Grunnet tidsbegrensinger vil gruppen fokusere hovudsaklig på sky-tjeneste siden av utviklingen, siden denne er mest relevant både for gruppen og Seaonics. Neste avsnitt forklarer den aktiviteten som blir utsatt til alt annet er ferdig, dersom tiden strekker til. Gruppen vil heller bruke en simpel prototypet versjon av onboard tjenesten som Seaonics har fra før. Denne prototypen gjør lite mer enn å sende data som om det var en ekte tjeneste, men mangler den logikken som krevest.

Backenden vil utvikles trinnvis. I starten vil målet være å opprette en kommunikasjonsflyt mellom tjenestene gjennom Seaonics MQTT-megler. Denne flyten må optimaliseres mest mulig ettersom tjenerene som kjører hos fartøyene vil ha en begrenset nettverkbåndbredde gjennom sat-link.

Etter kommunikasjonsflyten er opprettet flyttes fokuset til å opprette ett caching-system hos skytjenestene som gjør at antall kall gjort til onboard tjenerene minimeres. Dette steget vil da involvere oppretting av databaser og ett førsteutkast av ett API som tilgjengeliggjør dataene.

Etter backenden lagrer data vedvarende vil tjenesten bli oppkoblet mot Seaonics adgangssystem slik at Seaonics har kontroll på hvem/hva har tilgang til dataen lagret på tjenesten. Dette steget kan også skje før tjenesten blir satt opp for å lagre data, dette må seest ann.

Når tjenestene som oppgjør backenden i prosjektet nærmer seg fullført vil fokuset legges om til å tilgjengeliggjøre datane til web-grensesnittet gjennom ett API. Sannsynlig blir også da ett eget autentisering-system nødvendig for tjenesten.

Etter alle tjenestene er sammenkoblet vil gruppen hovedsaklig bruke tid på å utforske diverse tema som datakompresjon for å videre forbedre nettverksbruken. Målet blir og å utbedre diverse aspekt rundt tjenestene dersom gruppen, veileder eller oppgavegiver finner feil eller mangler.

Frontend

Hovedoppgaven for frontend, vil være å presentere logg dataen som har blitt fanget i backend. Her vil det derfor bli viktig å har et grensesnitt hvor oversikt over tid er sentralt. Det skal være lett å navigere seg gjennom forskjellige dager, måneder og tidspunkt for den gitte logg dataen.

Det som er viktig når man utvikler brukergrensesnitt, er å ta seg god til i forkant til utforming og design valg. det å endre på fundasjonen til en nettside lengre ut i utviklingsløpet kan stjele mye verdifull tid til videreutvikling av ny funksjonalitet. Men man skal samtidig ikke se bort ifra at det blir mer enn nokk gjenopptakelse av gamle valg i en sprint prosess.

Designprosessen til nettsiden vil holdes i korte sprinter ettersom man får tilbakemelding fra kunden / oppgavegiver. Viktige spørsmål som; Hvem har tilgang til tjenesten? eller; Hvordan får man tilgang til tjeneste? er begge sentrale spørsmål til hvordan funksjons flyten til nettsiden må være. videre kan man spørre; Trenger man innlogging før man får tilgang til noe som helst? eller kan man rett å slett få se alt som er tilgjengelig rett etter ankomst til nettsiden?

Etter å fått nærmere innføringen i hvordan flyten av nettstedet vil være, kan man begynne sett å tegne opp sammenhengen mellom de forskjellige rutene en bruker kan ta inne på nettstedet. disse vil være essensielle for hvordan man bygger opp nettsiden.

Håndtering av nettverks kall opp mot backend er også sentralt i utviklingen av et robust nettsted. frontend og backend må presisere en sterk protocol for hvordan forespørsel og svar skal se ut. ut i fra dette kan man begge sider på best evne sikre at man ikke ender opp med uønsket utfall.

Ettersom at løsningen kan bli brukt av service personell som befinner seg på båt, vil det å lage en nettside som er lett å laste være prioritet. Dette er noe som kan oppnås ved hjelp av en flere siders SSR (Server Side Rendering) applikasjon. Primært på studie, har vi egentlig kun lært om enkelt side applikasjoner når det kommer til web-utvikling. en slik løsning vil måtte kunne sende store mengder JavaScript, CSS og HTML i hvert enkelt nettverks kall. Men er SSR løsning vil kun det som trengs til hver side sendes med til klienten.

ED2: Time-series cache

Utviklingen vil starte med utgangspunkt i statisk data, som vil etterligne hva et eventuelt kall til backend ville sett ut, dette vil hjelpe utviklingen slik at nettsiden kan utvikles i forkant av funksjonaliteten til backend, samt være lett å sy sammen når funksjonaliteten er klar.

Videre i utviklingsløpet må vi se ann om det skal være noe form for data manipulasjon som skal gjøres gjennom nettsiden. Dette vil føre til videre utfordringer i form av hvordan å opprettholde nettside staten, om data som vises på skjermen har blitt endret av bruker, hvordan skal vi da innføre den nye dataen slik at nettside er på samme stadiet som databasen.

Det vil også være relevant å få formulert user-stories fra service personell. Dette er hoved bruks gruppen vår, så å forstå hvordan de feilsøker og evnt bruker dataen som de har tilgjengelig den dag i dag vil være hjelpsomt til å finne ut hva som trengs å vise fram. Om det lar seg, så er et av de senere målene å få løsningen testet av noen med service bakgrunn, for å se om et evt problem kan bli oppdaget gjennom den nye løsningen.

3.2. Milepæler

Opplisting av kritiske datoer.

10. Januar: Planlegging av prosjektet begynner

28. Januar, fullføring av kontrakter og forprosjektplan

11. Februar, få satt opp kommunikasjonsflyt mellom komponenter

25. Februar, datahåndtering i skyen, og klargjøring for å lagre data vedvarende

11. Mars, ferdiggjøring av databaseoppsett, slik at data kan lagres vedvarende

25. Mars, fungerende tjeneste som fungerer som en sky-cache av data

20. Mai: innlevering av bacheloroppgave / rapport

Grunnet uvisshet om kompleksitet har gruppen satt av en periode etter 25. mars for å hente inn igjen det omfange som ble fjernet (onboard tjeneste), dersom målene frem til da er nådde. Milepælene vil derimot trolig bli forskyvd.

4. Oppfølging og kvalitetssikring

4.1 Kvalitetssikring

For å sikre kvaliteten på arbeidet vil gruppen først og fremst kjøre kortere sprinter med daglige møter hvor gruppemedlemmene vil legge fram gårsdagens fremgang og valg slik at alle ligger på samme plan. De daglige møtene vil også benyttes for å se på hverandres kode og eventuelt stille spørsmål om den. I tillegg til dette skal koden dokumenteres fortløpende for å gi best mulig innsikt. Dersom tiden kan strekkes til, og behovet er der, skal og koden testes.

4.2 Rapportering

Rapportering vil bestå av sammendrag av prosjektets status for hver sprint. Rapportene vil være utsendt til veileder og oppdragsgiver innen de bi-ukentlige møtene. Desse rapportene vil stort sett være i form av SCRUM retrospektiver for sprintene.

Gantt-skjemaet på gruppens wiki vil og bli holdt oppdatert, slik at veileder(e) og prosjektgivere skal kunne få innsyn i prosjektets status

5. Risikovurdering

Ettersom systemet gruppen skal utvikle vil være sterkt knytt til Seaonics systemer er der en viss risiko rundt integrasjonen. Den risikoen går hovudsaklig på uforventet tidsbruk, og har en ganske høy sannsynlighet. Tiltaket gruppen har tatt for å eventuelt mitigere dette er å legge en buffer i planen, slik at de fortsatt vil ha tiden de trenger.

Slik som alle kode-prosjekt er der en risiko rundt tap av data og tap av arbeid, som kan oppstå ved uhell, ved korrupterte filer etc. For kode bruker gruppen GitHub for versjonsstyring, slik at en enkelt maskins feil ikke vil føre til store tap, i tillegg til dette utøver gruppen god “git-hygiene”, med hyppige commits. Alt av kode vil derfor alltid være lagret både på gruppens maskiner samt i skyen. For andre data og filer vil gruppen bruke Confluence som gir mye av den samme sikkerheten som GitHub gjør for koden.

6. Vedlegg

Følgende dokumenter leveres som separate filer ved innlevering i Blackboard i januar (obligatorisk arbeidskrav), men ikke i endelige leveransen av hovedrapporten den 20. mai!

6.1 Tidsplan

Startfasen i uke 3-4 vil gå til å innhente krav og ønsker fra arbeidsgiver, Seaonics. Deretter vil gruppen danne en vel-definert problemstilling for oppgaven. Programvare som trengst for dokumentasjon, hovudsaklig confluence, jira og github vil bli satt opp slik det er klart til utviklings-start i de kommende ukene.

Videre i Uke 5-6 vil gruppen senærmere på hva som trengs (teknisk) for problemstillingen de lander på, det vil da også være relevant og begynne med Wireframing og sketching av grensesnitt for nettstedet, samt datamodellering og service modellering for database og API. Valg av grunnleggende teknologier vil og bli tatt innen slutten av denne perioden. Dersom tiden kan strekkes til begynder gruppen å utvikle de ulike komponentene allerede her.

Uke 7-8: Gruppens første utviklings-mål er å sette opp kommunikasjon fra fartøyene til skytjenesten, gjennom Seaonics MQTT system. Fokuset på å minimere nettverksbruk, og komprimering av data vil stå sentralt under denne sprinten. Samt oppsett av grunnstrukturen for nettsiden, slik at videreutviklig kan ta plass når gruppen finner ut hvordan nettsiden vil brukes. Grunnet endringer i omfanget vil denne sprinten bli brukt på følgende sprints gjøremål.

Uke 9-10: Gruppen vil fokusere på caching av data på skytjenesten og håndtering av date på nettsiden, her vil dummy data brukes om relevante endepunkt ikke blir satt opp i tide.

Uke 11-12: Fullføring av integrasjon mellom skytjenesten og nettsiden, samt skytjenesten og on-board tjenesten.

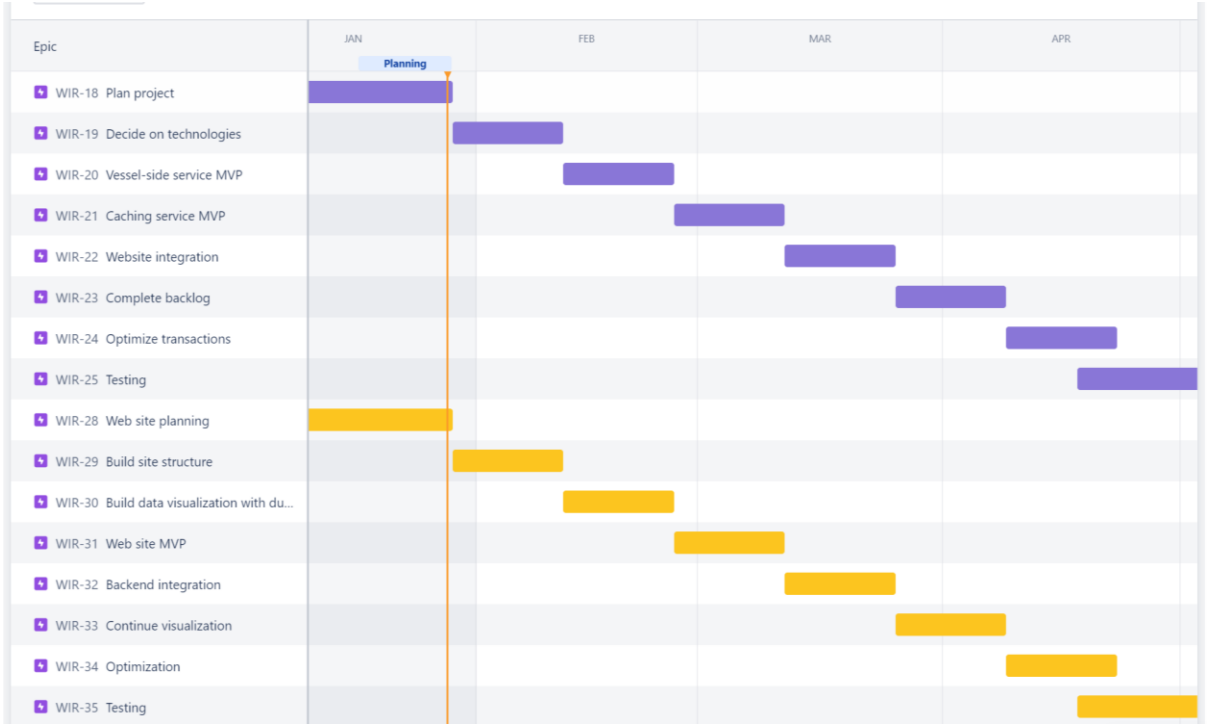
Uke 13-14: Mindre kritiske oppgaver som ikke har blitt gjort som planlagt vil bli fullført ukene.

Uke 15-16: Buffer-tid dersom noen av oppgavene tar lengre tid enn anntatt, eller om utbedringer må gjøres. Dersom gruppen ikke trenger desse ukene vil planen gå videre til neste ukers gjøremål. Ett annet fokus desse ukene vil omhandle optimisering av systemet ved hjelp av nye teknikker eller forbedring av tidligere nevnte teknikker, slik som komprimering.

ED2: Time-series cache

Uke 17-18: Testing av systemene i “den virkelige verden”. Dette vil gjøres så fort gruppen klarer å opprette en MVP, slik at gruppen får hentet inn tilbakemeldinger fra brukere av tjenesten. Disse tilbakemeldingene vil da hovedsakelig bli brukt for å reflektere over prosjektet og resultatet, men, dersom der er tid vil gruppen agere på de mest relevante tilbakemeldingene

Uke 19-20: Vil gå til fullføring av dokumentasjon



6.2 Adresseliste

Navn, firma, tlf., epost, adresse

David Rise Knotten
tlf: 94801211
epost: davidrk@ntnu.no
addresse: Nørvegjerdet 2B, 6009 Ålesund

Einar Andreas Aglen
tlf: 97829858
epost: einaragl@stud.ntnu.no
addresse: Hatlevika 21, 6016, Ålesund

ED2: Time-series cache

Tom Jørann Giske
Email: tom.giske@seaonics.com
Tlf: 71391669

Erik Espenakk
Email: erik.espenakk@seaonics.com
Tlf: 71391656

6.3 Avtaledokumenter

6.3.1 Arbeidskontrakt for bachelor-gruppen

6.3.2 3-partsavtale

Appendix 2

Time-series Cache Requirement Documentation

Table of contents

Table of Figures

Introduction

This document describes the requirements of each of the systems components, the cache, the REST service and the frontend. Since the project started out very open, these functional requirements have been created as the project has progressed. In that sense, the process has been very agile, with functional requirements being created as their needs arise.

Use Case diagrams

The use case diagram for the services are shown in Figure 1, Figure 2 and Figure 3

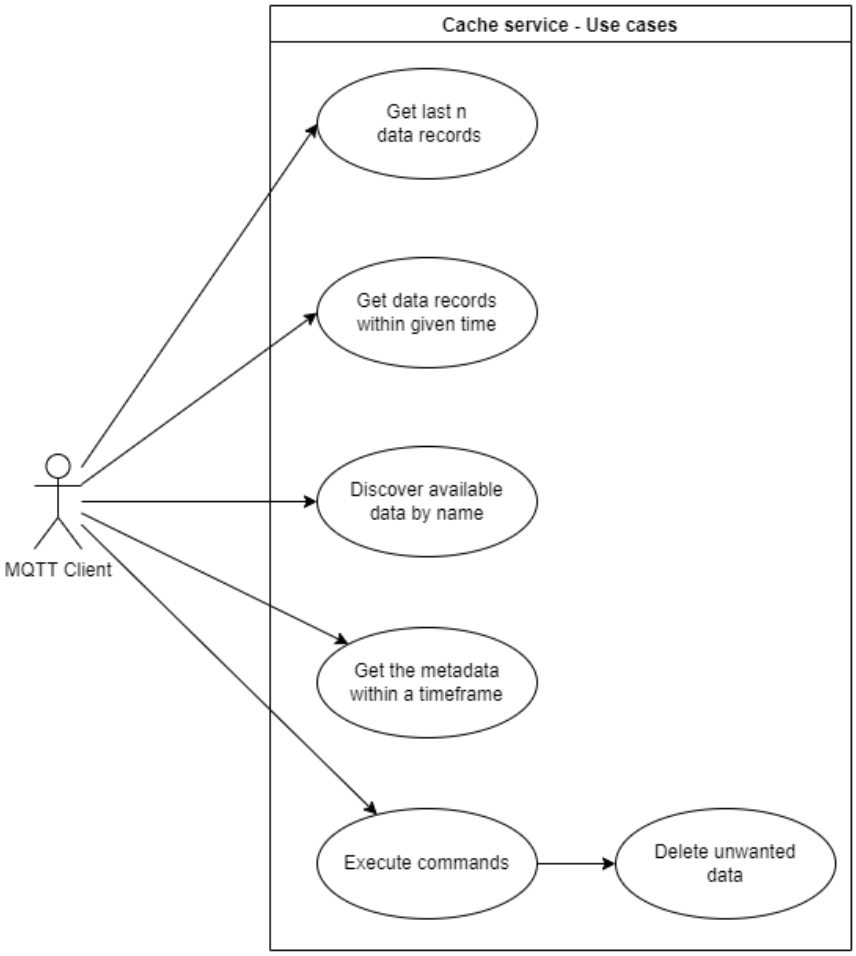


Figure 1: Cache Service Use Case Diagram

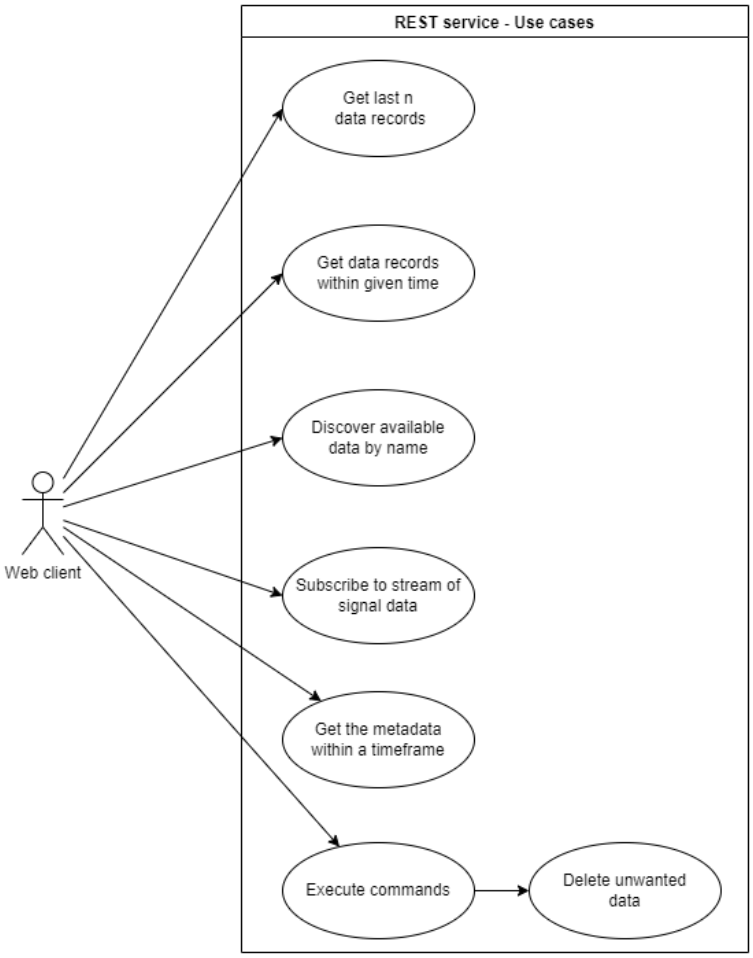


Figure 2: REST Service Use Case Diagram

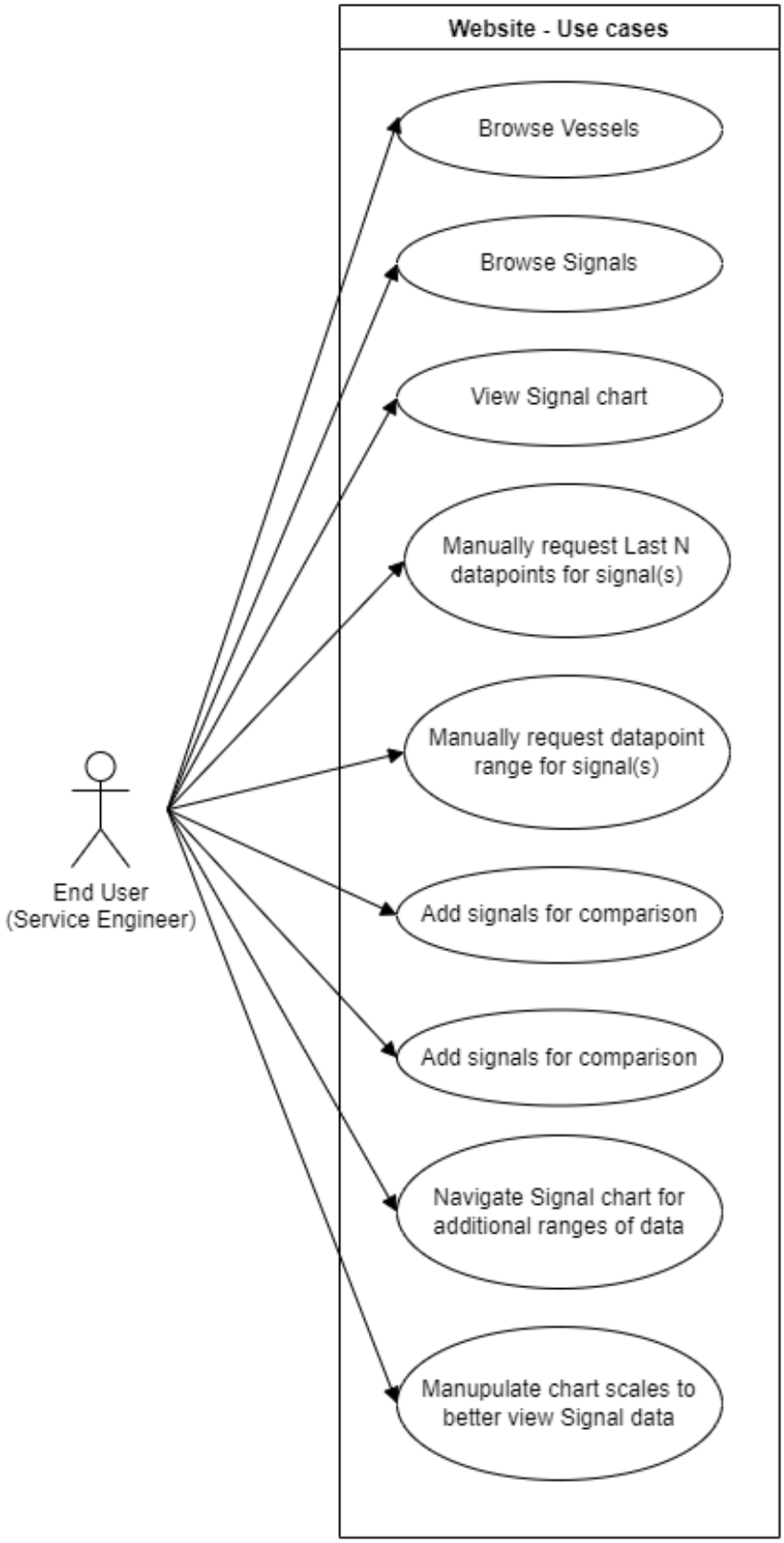


Figure 3: Website Use Case Diagram

User Stories

Backend

The user stories for the backend services are quite similar since the REST service is just a web-facing proxy for the actual cache. They are also few, due to its technical complexity being hidden from clients.

For both services:
As a user I want to discover the signals on a specific client so that I can find relevant signals for debugging
As a user I want to fetch the last changes of specific signals so that I can monitor the current status, or value.
As a user I want to fetch the data points from specific intervals so I can monitor earlier states of the system to find potential issues.
As a user I want to see how much data a request uses so that I can be aware of the load I place on a vessel
As a user I want to delete data from the cache so that I can avoid running out of space

For only the cache service
As a user I want to know whether or not the cache is active so that I can avoid waiting for or requesting data that won't be sent
As a user I only want to request the same data from a vessel once so that I don't strain vessel-networks

For only the REST service
As a user I want to continuously stream changes so I can monitor a signal in real-time

Frontend

User stories for the group's frontend web-application aim to represent the needs of the end-user, typically this would be a service engineer working for Seaonics.

As an End User I want to be able to browse for available vessels.
As an End User I want to be able to browse available signals for a given vessel.
As an End User I want to be able to search for available signals for a given vessel.
As an End User I want to be able to read About the solution.
As an End User I want to be able to view the last N datapoints of selected signals.
As an End User I want to be able to view datapoint in each time-space
As an End User I want to be able to view selected signals datapoints in chart format
As an End User I want to be able to add additional signals to chart for comparison
As an End User I want to be able to discover new datapoints when interacting with the chart
As an End User I want to be able to view individual scales for each dataset
As an End User I want to be able to view individual colors for each scale
As an End User I want to be able to share my findings via direct link
As an End User I want to be able to share my findings via quick image capture

Diagrams

In Figure 4: Website Activity Diagram the general flow of actions can be seen. There are multiple ways of achieving the same range fetch, as well as interacting with the chart. The activity diagram displays available options for different parts of the website, then what the effect of such actions would cause.

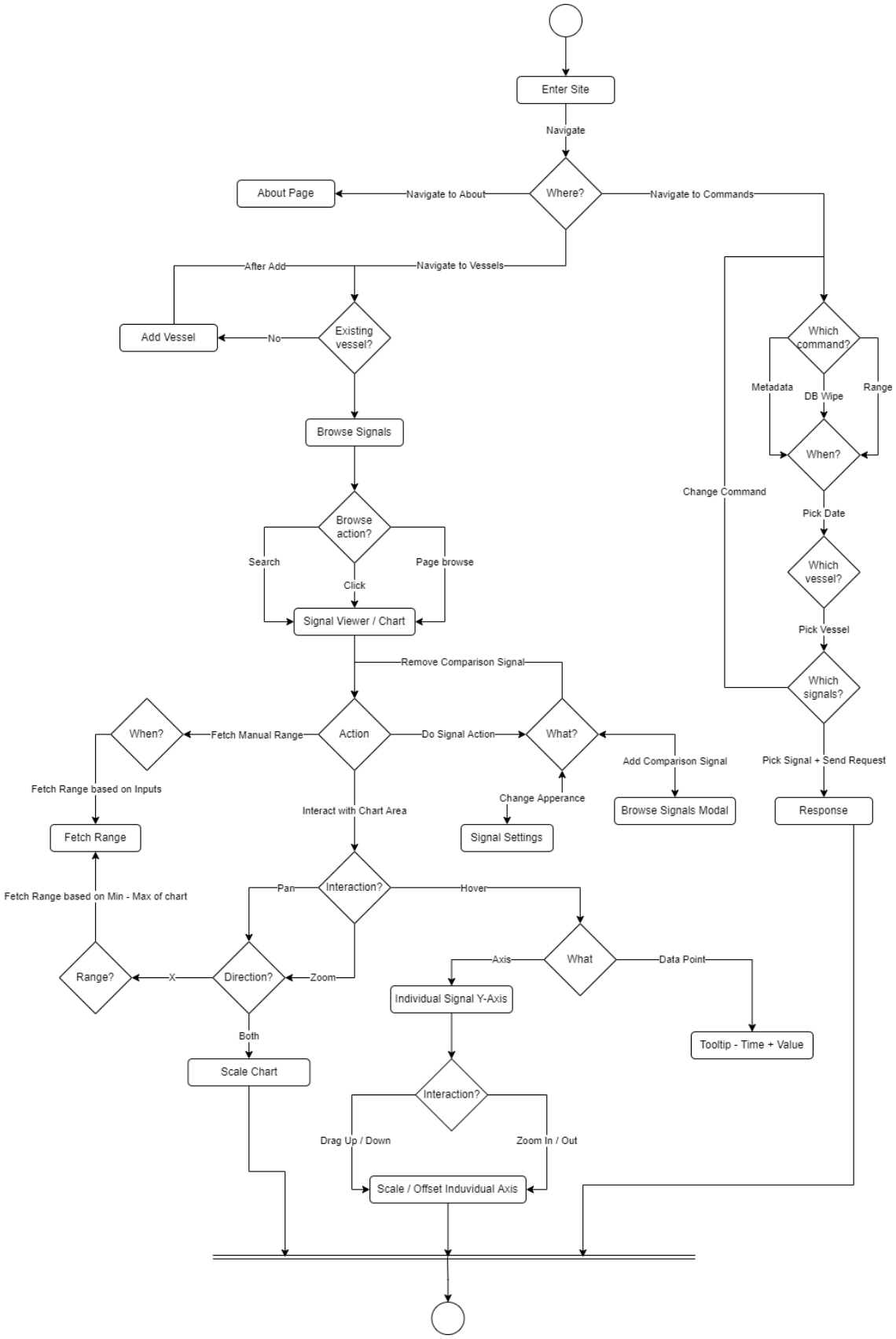


Figure 4: Website Activity Diagram

Prototypes

At the start of website development, it was clear that the group had do build some type of multiple dataset interactive chart. Some prototypes were developed to see if it would be realistic to place multiple datasets into one chart, then overlay them and let the user style them in different way. In Figure 5: Early version of adding multiple datasets in same chart a prototype of when the first implemented multiple datasets can be seen. Each data set got their own randomly generated color and y-axis

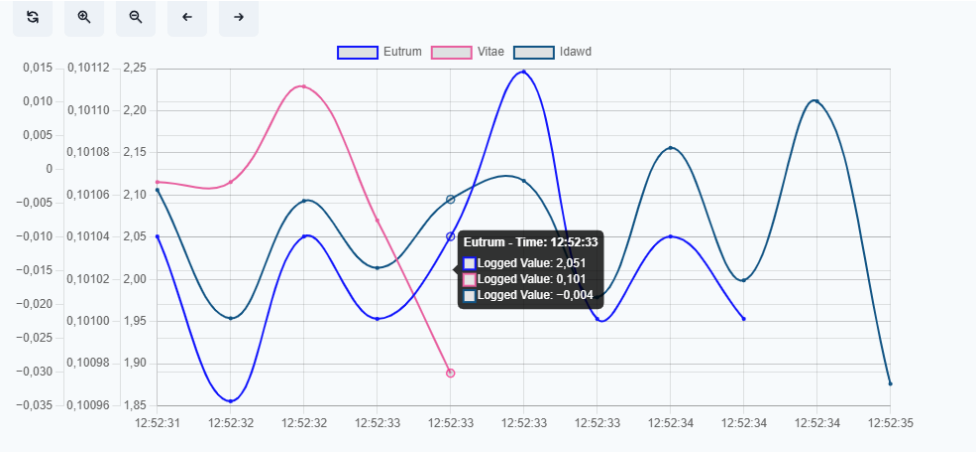


Figure 5: Early version of adding multiple datasets in same chart

The prototype grew into something closer to the final product after some feed back from clients. In Figure 6: Prototype Screenshot in Dark Mode, the y-axis has been updated to match the color of the dataset that it controls. The implementation of stepped lies can also be seen. Experimenting with displaying all datapoints that log the same timestamp can also be seen.

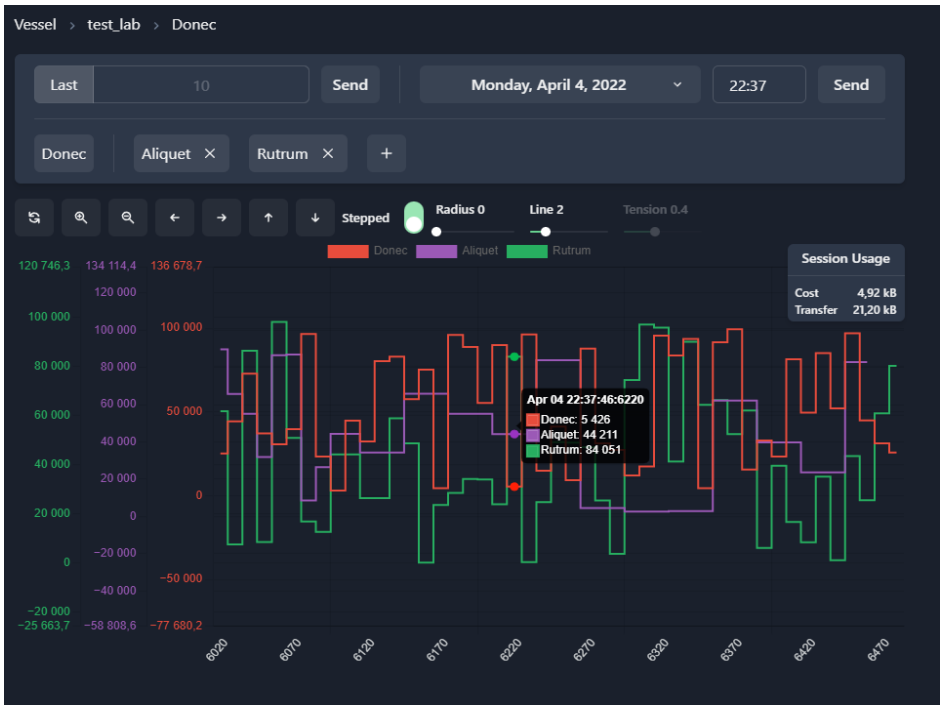


Figure 6: Prototype Screenshot in Dark Mode

The option to toggle between dark and light mode was implemented from the start with Chakra UI, in Figure 7: Prototype Screenshot in Light Mode, the light mode version can be seen working with the chart area as well.

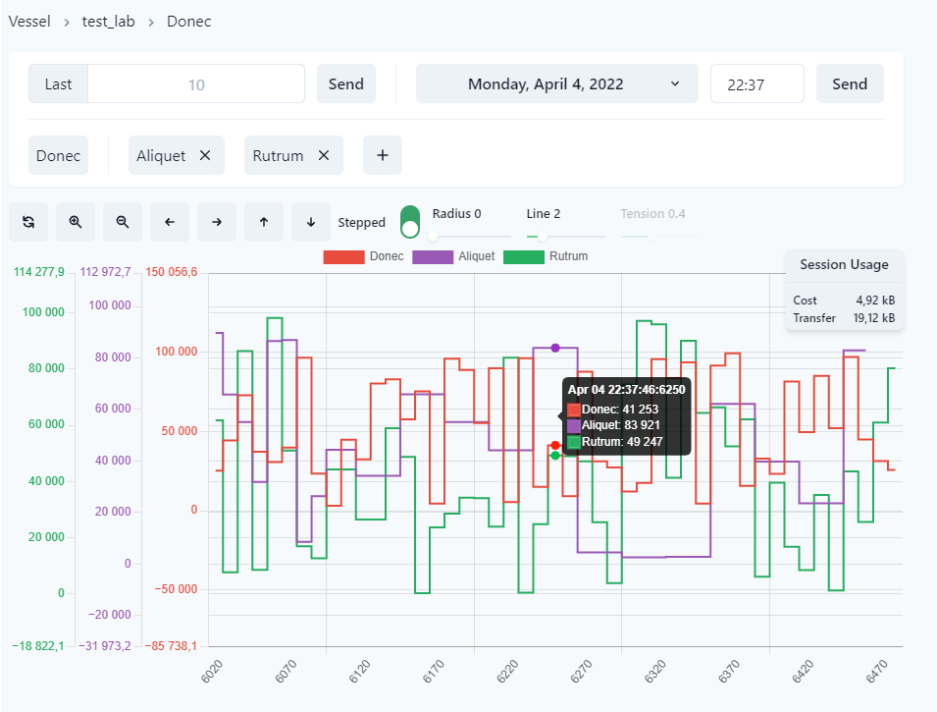


Figure 7: Prototype Screenshot in Light Mode

Wireframes

Wireframes were used to get a feel for the placement of important aspects of the workflow that would be presented to an end-user of the product. Here the group has tried to get a feel for what is urgent for the user to always access, and what can be hidden for more specific tasks. The following figures, Figure 8: Signal Viewer Wireframe and Figure 9: Signal Picker Wireframe show the two main components of the website. The first component shown is the signal viewer, which displays a graph with data collected from the cache, as well as some options regarding the visualization of the graph. The second component shows the signal picker, which displays a list fetched from the cache containing all available signals at the selected vessel.
On the left side of both wireframes, a list containing all vessels added by the users spans the height of the page. Additionally, a navigation bar at the top includes any links to pages added later on.

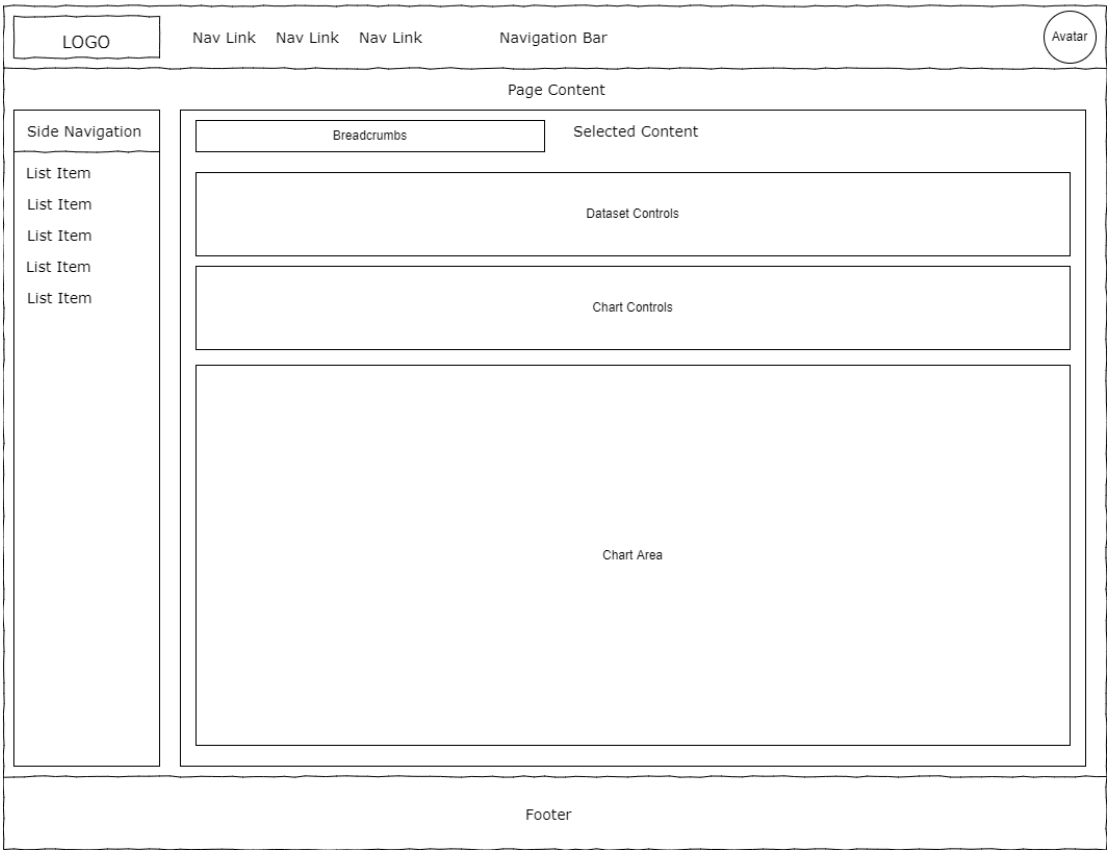


Figure 8: Signal Viewer Wireframe

Figure 9: Signal Picker Wireframe

Storyboards

As seen in Figure 10: Storyboard, there are a lot of hidden functionality in the chart area. Since the group wanted the user to have full control of the lines being drawn, the group found that the best way of implementing this was through mouse and scroll events placed on the charts general rendering area as well as the axis connected to the datasets being drawn

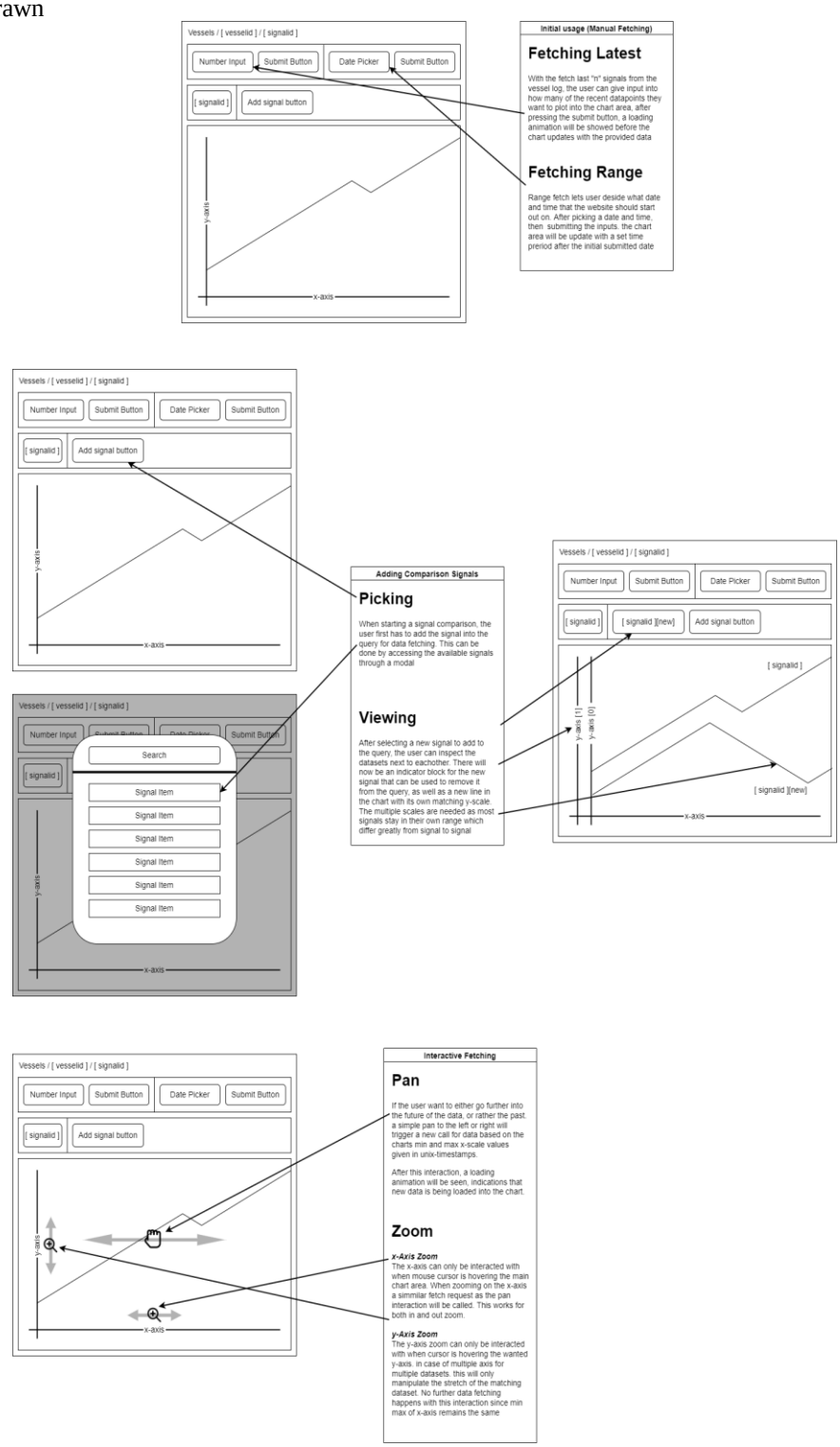


Figure 10: Storyboard

Appendix 3

Time-Series Cache System documentation v1.0

Revision history

Dato	Version	Description	Author
17/05/2022	1.0	Original submission of Time-Series Cache system documentation	David Rise Knotten, Einar Andreas Aglen

Table of contents

Table of Figures

Introduction

The purpose of this document is to provide a technical overview of the system developed in the Time-Series Cache project.

Architecture

Overview

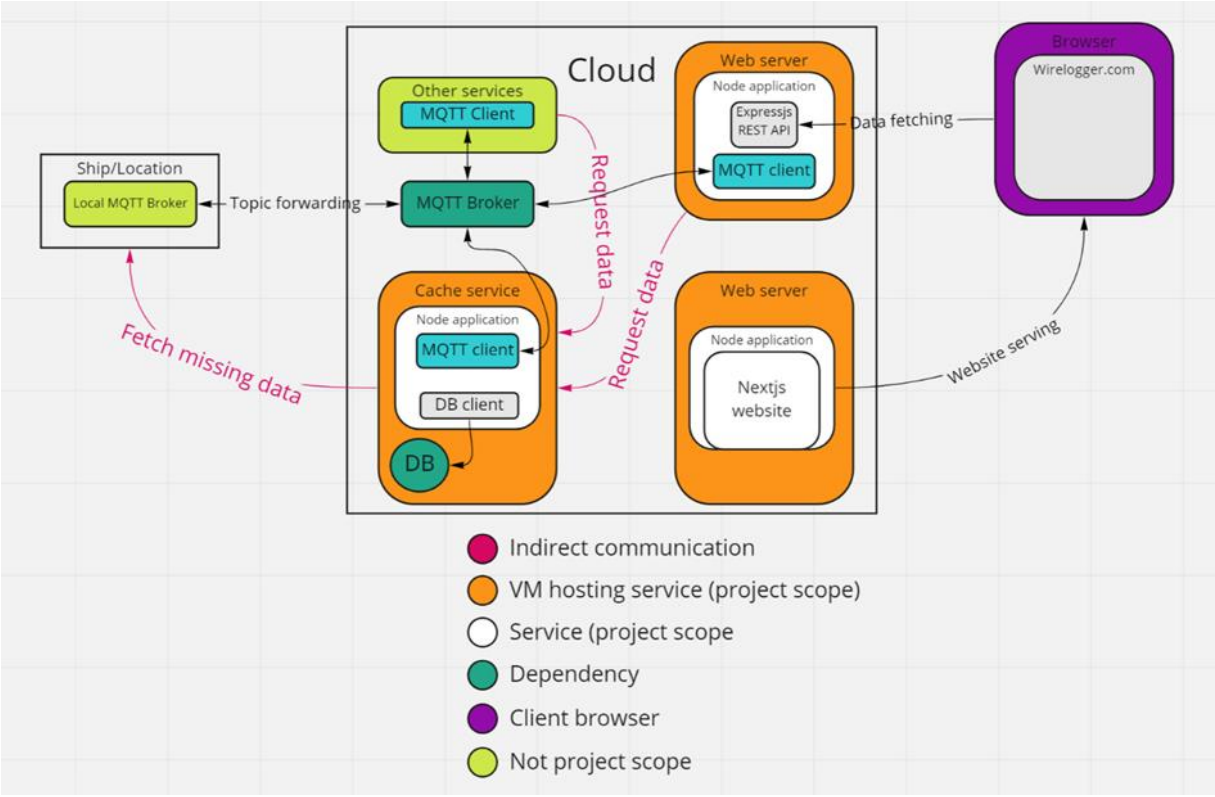


Figure 1: Overview of system architecture

The architecture for the whole system is shown in Figure 1. The main component is the Cache service which holds a database and acts as a proxy for all vessels. Communicating with the cache are other services who communicate with it through the MQTT broker. Any service can use its API assuming they have access to the broker. The REST service acts like any other service and communicates with the cache through MQTT. It then exposes an API which acts as an HTTP proxy for the cache and extends its functionality to a webapp. The wirelogger site uses the aforementioned REST service to query time-series data from the vessels and displays the data with different graphs.

This illustration of the architecture is somewhat simplified with other components such as Nginx proxies most likely being used between these services in a real-world scenario.

The system consists of six key components:

- The cache service
- The REST service
- The website
- A TimescaleDB database
- An MQTT broker
- A vessel-service

The way the components are depicted in the figure is not necessarily how things need be set up. For example, the database could be replaced by a hosted database from timescale [1].

Communication

All of the communication between services, excluding the REST-website communication, goes through the central MQTT broker. All services also use a protocol buffer definition to ensure that all communication can be understood by other components, as well as being easily implemented in additional services independently of programming language.

```
message Signal {
    string name = 1;
    map<int64, double> double = 2;
}

message Error {
    int32 code = 1;
    string message = 2;
}

message Changes {
    message FromTo {
        int64 fromTime = 1;
        int64 toTime = 2;
    }
    message LastN {
        int64 nChanges = 1;
    }
    oneof request {
        FromTo fromTo = 1;
        LastN lastN = 2;
    }

    repeated Signal signal = 3;

    optional Error error = 4;
    optional int32 bytecost = 5;
}
```

Figure 2: Changes interface

Since the Changes interface is defined by Protobuf, code can be automatically generated for most languages, both for interfaces, but also for encoding/decoding, thus eliminating any issues of repetitiveness for implementing definitions on services of other languages. The changes interface shown in Figure 2 is used for all signal-data requests. For more details on how to use it, check out the cache service readme.

To give an example in a more recognizable format, the following is a JSON object with a Changes interface:

```
{
  fromTo: {
    fromTime: 100,
    toTime: 1000
  },
  Signal: [
    {
    name: Lorem,
    double: {
      102: 35.72,
      104: 35.83
    }
    }
  ]
}
```

Figure 3: JSON Changes

This JSON changes format (Figure 3) is what's used between the REST service and website, simply to avoid unnecessary encoding/decoding at all steps, as well as to keep things simple for the frontend, both in terms of code handling, but also readability and unnecessary dependencies.

While the changes interface is the most used, the group has also created a "command" and "healthcheck" interface, shown below

```
message HealthCheck {
  string message = 1;
}
```

Figure 4: Healthcheck interface

Since requests of big data can take a long time, the group saw fit to implement a healthchecking system, this way, if the cache goes down or cannot handle requests, the REST service will be aware and respond accordingly, instead of simply never responding and forever wait for a response. This interface is shown in Figure 4, a message is published every three seconds, go to the cache service readme for more information.

```
message Ranges {
  int64 from = 1;
  int64 to = 2;
}

message Command {
  string action = 1;

  optional string vessel = 2;
  repeated string data = 3;
  optional Ranges ranges = 4;
}

message CommandResponse {
  bool success = 1;
  optional string error = 2;
}
```

Figure 5: Command interface

The command interface, shown in Figure 5, is quite simple and currently only supports deleting things from the database through the "wipe" action. For more information, check out the cache service readme.

```
message MetaDataRequest {
  repeated string variable = 1;
  optional int64 from = 2;
  optional int64 to = 3;
}

message MetaDataRange {
  int64 from = 1;
  int64 to = 2;
}

message MetaDataList {
  string variable = 1;
  repeated MetaDataRange ranges = 2;
}

message MetaDataResponse {
  repeated MetaDataList metadata = 1;
  optional string error = 2;
}
```

Figure 6: Metadata request and response interfaces

The Metadata interfaces, shown in Figure 6, are used to provide the caches metadata if any service requires the context. Currently it's not used other than displaying the raw JSON data on the website, but Seaonics has asked for a feature of displaying what data is saved, which will be implemented in the front-end down the line. More details are provided in the caching service readme.

Project structure

This chapter will give an overview of the services file-structures as well as an overview of how the folders relate.

Cache service

The cache service is the core component in the system and handles all data flowing through. The architecture of the cache service was developed with a modular, functional approach.

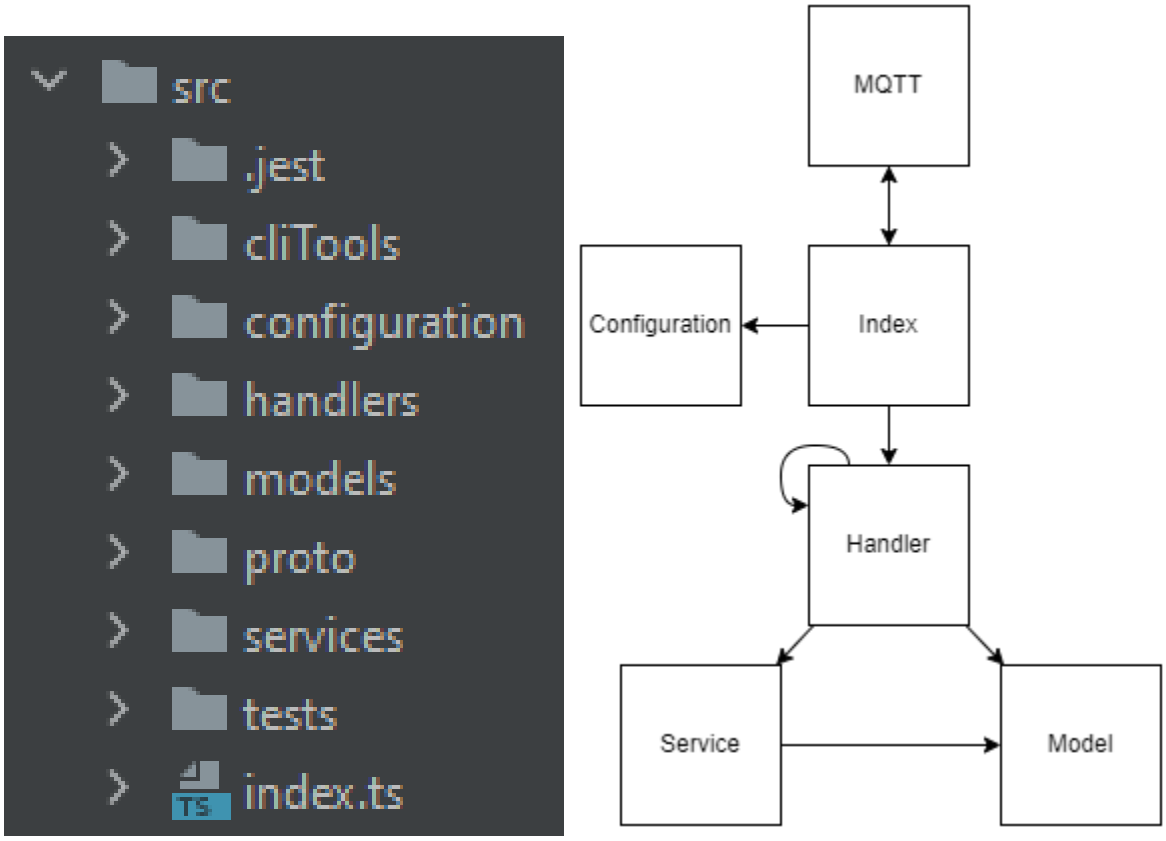


Figure 7: Cache code structure

Figure 7 shows the code structure of the cache service. On the left side there is the folder structure of the code, and the right figure shows how the main components rely on each other, MQTT being the remainder of the system through the MQTT broker

handlers
The main business logic of the service, such as fetching a range of data and combining it to return in a response. These functions control the flow of execution and are called by incoming requests or other handlers (There are some internal “layers”). The handler structure is set up according to the flow of a requests. One exception is the get range request which is simply too complex to place in a single file, and is therefore split into its own subdirectory of requests

models
Mainly database models and some interfaces.

services
More general code which is more repetitively used or of more utilitarian nature. Used by handlers either to do repetitive actions or more complex actions that aren’t directly tied to the fulfilment of the request.

proto
The proto directory contains the typescript code generated from the proto definitions.

configuration
Code that relates to configuration is placed in the configuration folder. This includes the code to load environment variables, some constants used throughout and definitions.

.jest
All code used for testing that isn't strictly unit tests is in here, such as the in-memory database and broker setup files

tests
All tests are, for the sake of tidiness combined in their own folder

cliTools
The clitools folder contains some utility code accessed by package.json scripts, such as a script for wiping the database, starting up a mock of the vessel service etc.

REST service

The REST service extends the caches interface to be used on the web with HTTP and WebSockets.

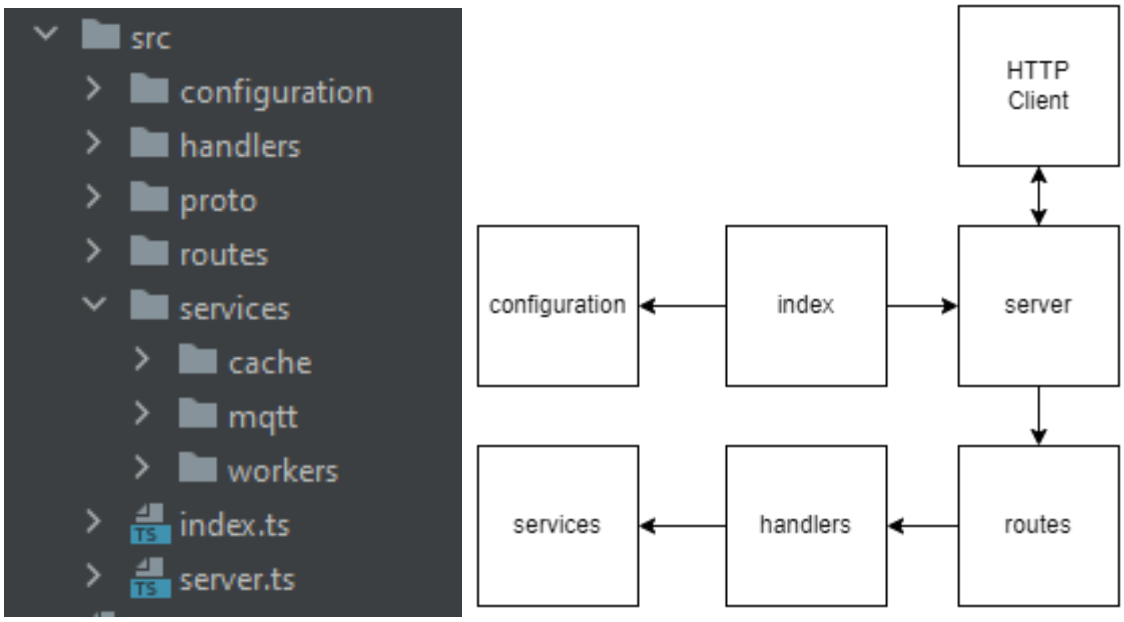


Figure 8: REST code structure

Figure 8 shows the code structure of the REST service. On the left side there is the folder structure of the code, and the right figure shows how the main components rely on each other.

handlers
All the logic required to act as a proxy to the cache

proto
Same as the cache service, the proto files include the code generated from proto definitions

routes
The route directory is a tree representing the routes of the REST API. Each endpoint is defined within the tree based on its path. For example, is the endpoint vessel/data/:data added in routes/vessel/data/datum

services
There isn't a lot of services in the REST service since it doesn't have a lot of logic, but anything used by multiple endpoints and that can be generalized into a utility function is placed in this directory

configuration
Same as cache, code that relates to configuration is placed in the configuration folder. This includes the code to load environment variables, some constants used throughout and definitions.

Vessel service

The vessel service is not developed in this project and therefore a simple dummy-service providing randomized data has been created. It is, in fact, not even its own service, but a testing component in the cache-service.

Website

The website acts as the HMI layer for using the overall system. Here a user can navigate cached and un-cached data, and furthermore, pinpoint events of significance.

When writing a website with a JavaScript library such as Next.js and React.js keeping a clean and well managed folder structure is of high importance.

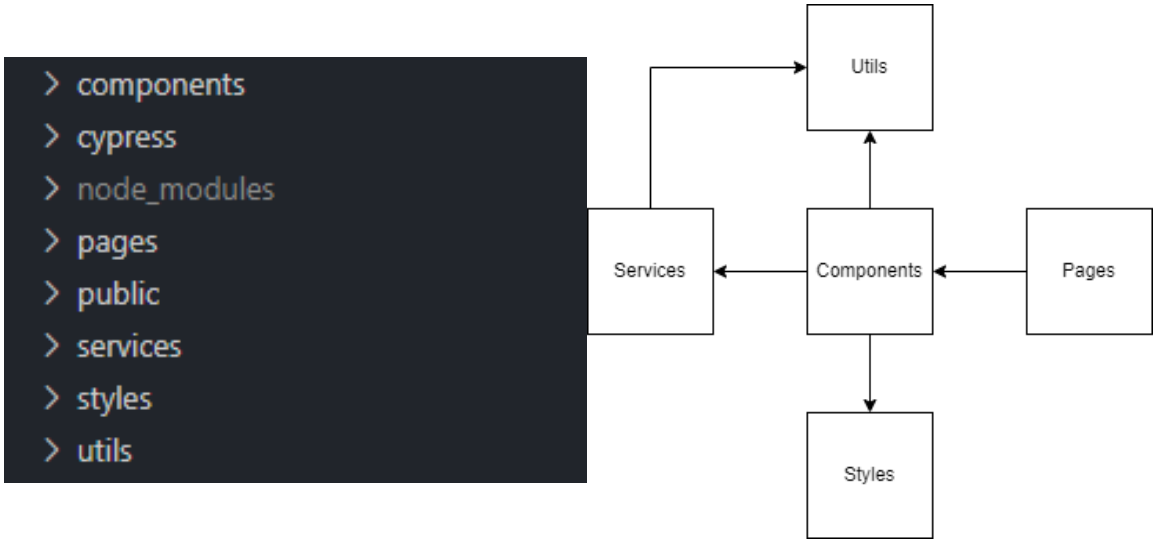


Figure 9: Website Project Structure

Figure 9 shows how the frontend project is built up, using various folders for different utilities.

components
All smaller bits of reusable bits of UI code should be stored here. All files should export a JavaScript function that returns data of type ReactElement which is then built into HTML, CSS, and plain JavaScript for the browser to render.

Cypress
All UI related tests are found here. They are written with the Cypress library to describe and predict UI behavior after user input.

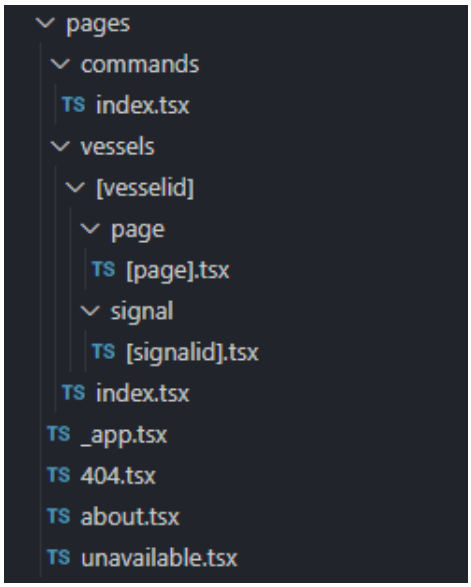


Figure 10: Pages structure

pages

Figure 10 defines how the websites routes are nested together and contain similar files exporting JavaScript functions. The main difference to the content of the Components files is that these files act as containers for the components, and they act as a template for the HTML the Next.js library is going to generate. Because of this, most logic should be kept out of these files, best practice is to only focus on the overlying website structure. And let the components handle the logic. Creating a layer between the view and the controller.

A special file called "_app.tsx" can be seen in the root of the folder. This file is the main point of entry for all the files. Here most state providers and various page layout wrappers are applied to the website.

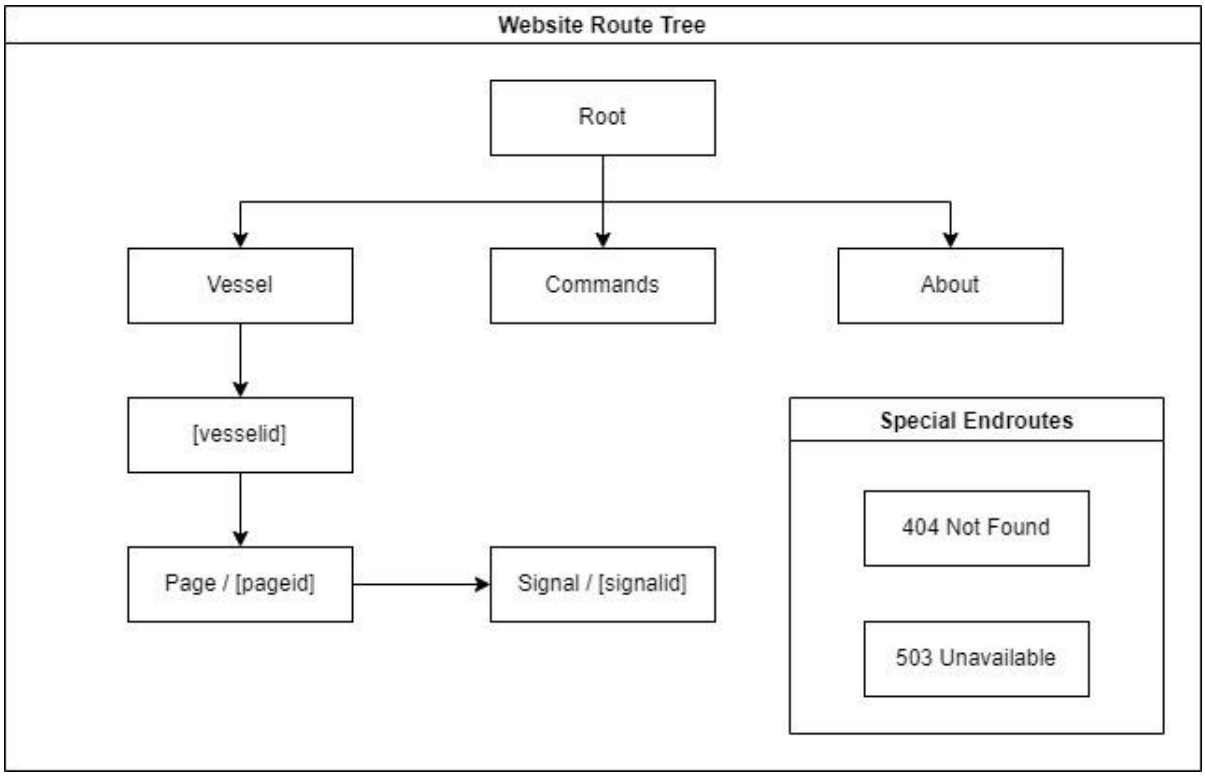


Figure 11: Tree Structure of website

Structure
In Figure 11 the navigable routes can be seen connected with pathway arrows. While the special routes such as 404 and 503 are only accessible at specific cases when user tries something that is either not there, or when the backend services are not available.

services
In the Service folder files that handle most of the connection to the REST API are found. Functions for calling on endpoints can all be found here, most of these are made to take in vessel, signals, and range.

styles
Because of the modularity of our chosen component library, there was the option to define and override a lot of the base styling that the library is shipped with. As the website is in a way going for a feel / look much resembling Seaonics previous solutions and services, the group wanted to change the colors that do not fit.

Inside we find the global CSS file where we can define smaller style classes that we can use as our own custom styling if needed. There can also be seen a components folder that directly relates to the various Chakra UI components that we want to override.

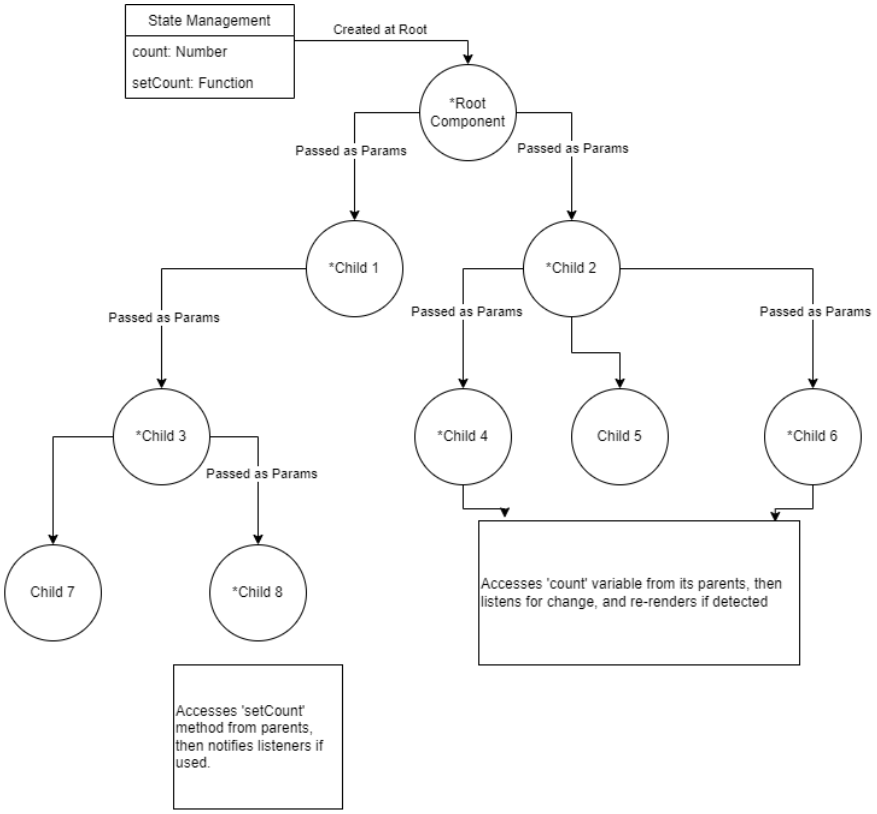
utils
The Utils folder is where most of the “dirty work” is found. A simple Tools file contain a lot of the reusable functions found throughout the website, such as string capitalization. There are also various handlers found in a Handlers folder. Inside this folder handlers for network requests that acts as a central for all requests, and where all potential side effects are handled.

State Management

Although the website should not keep track of too much data, there is a minimum limit to the responsibility that must be covered in the browser. When conduction state management for a component-based UI, the developer must consider what approach fits the project the best. In smaller parts of the component tree, simple prop drilling could be maintained, however, as the tree grows and more components enter it, the drilling can become un-manageable.

Consider the component trees seen in Figure 12 the one to the top utilizes prop drilling, where the first common parent initializes the shared state variables, then passes them via parameters to reach the components that need them. This makes it so a lot of components hold these variables without the need for them, these can be seen with a “*” as a prefix to their name. And if more components were to utilize them as well, we would have to traverse the tree yet again to find out how to reach the new component.

On the bottom we see the context approach. We use a Provider to wrap components and initialize shared variables in a common context. All components can tap into this context and mutate / notify other or listen for changes.



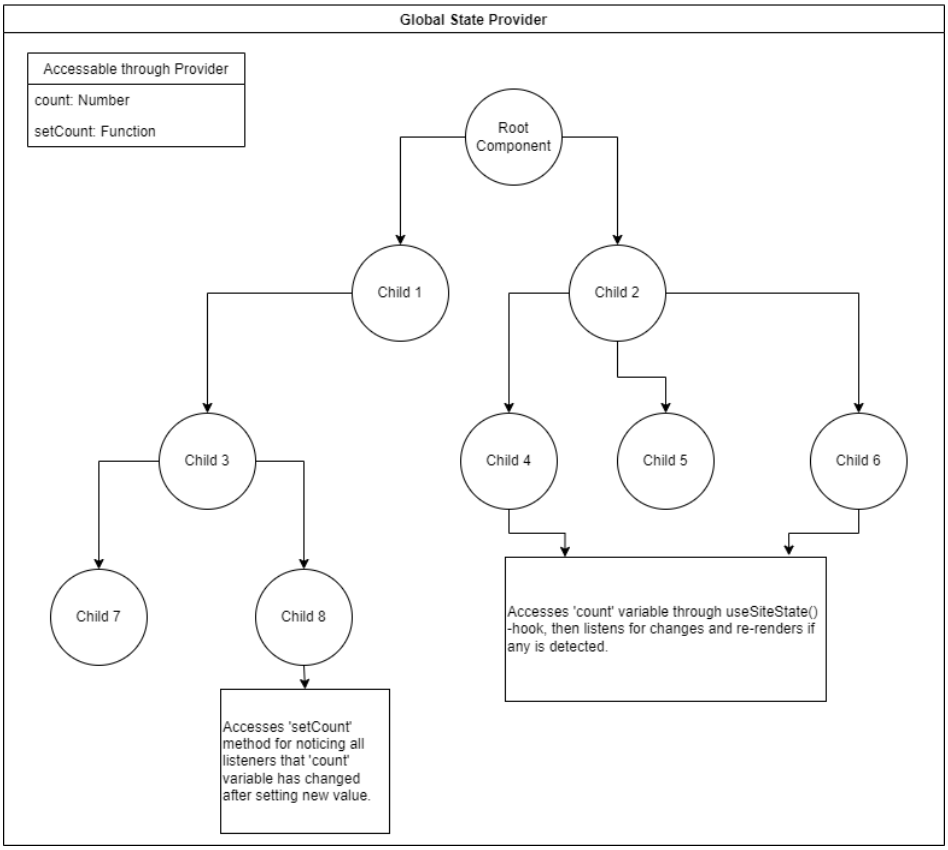


Figure 12: Prop drilling vs. Context Provider

Now for the projects a context-based state management system is used. Variables needed from all parts of the website are defined in the Global Site Context, whist shared variables for the Vessel / Signal part of the site are found on the Site Context. As seen in Figure 12, the group has tried have tried to differentiate between what is needed generally for the site experience, and what is needed for when viewing a signal and working with the data.

Since network requests can be sent from anywhere in the site, the “bytecost” and “bytetransfer” are defined in the Global Context. This is so that all requests can be added to the total count for each of these usage variables. And in the end the user can see how much data is being used.

Other context variables can be seen in their respected domain in Figure 13.

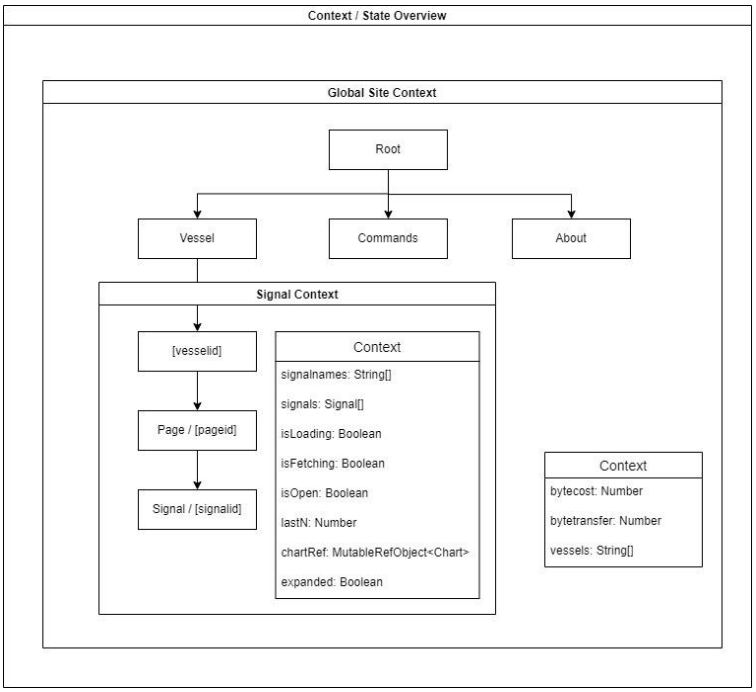


Figure 13: Project Context

Activity diagram – Website

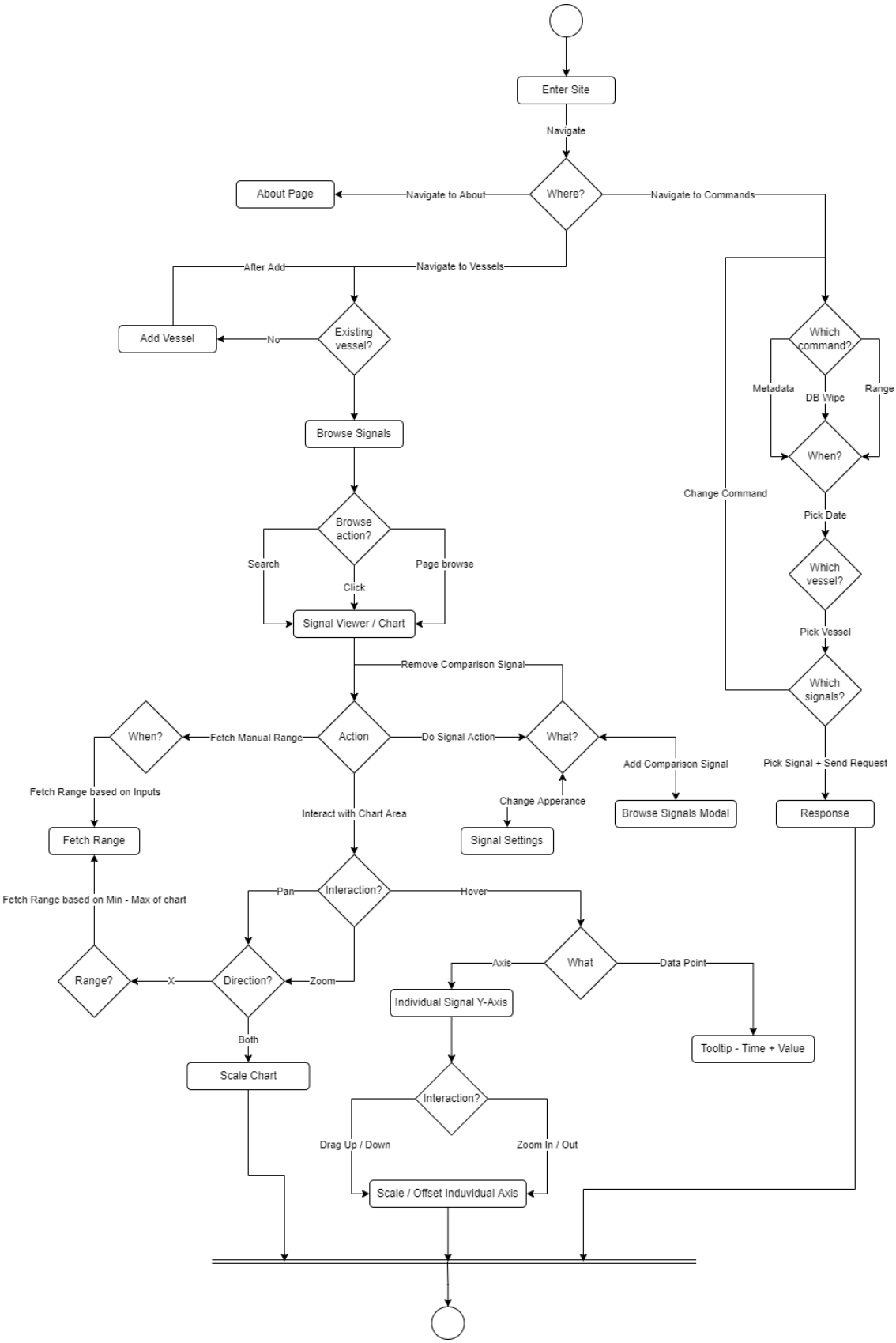


Figure 14: Website activity diagram

The activity diagram seen in Figure 14 depicts the actions and choices a user has when accessing the website. Entry point is found at the top of the diagram, and exit point is at the bottom of the diagram. Between these points all available paths and options can be found.

Sequence diagrams

Cache

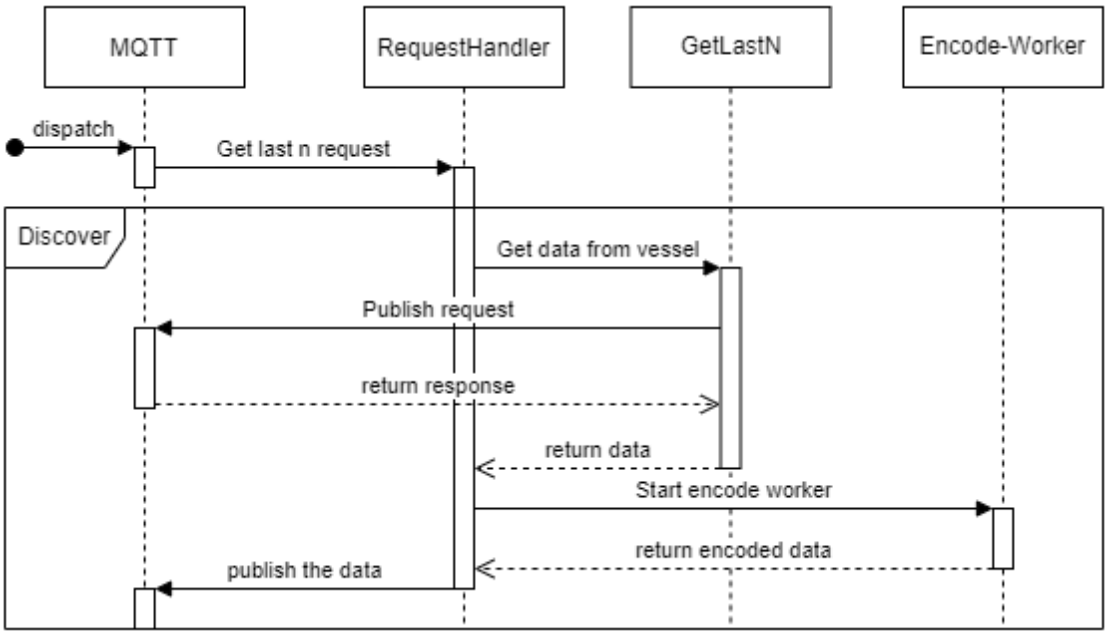


Figure 15: Cache get last n sequence

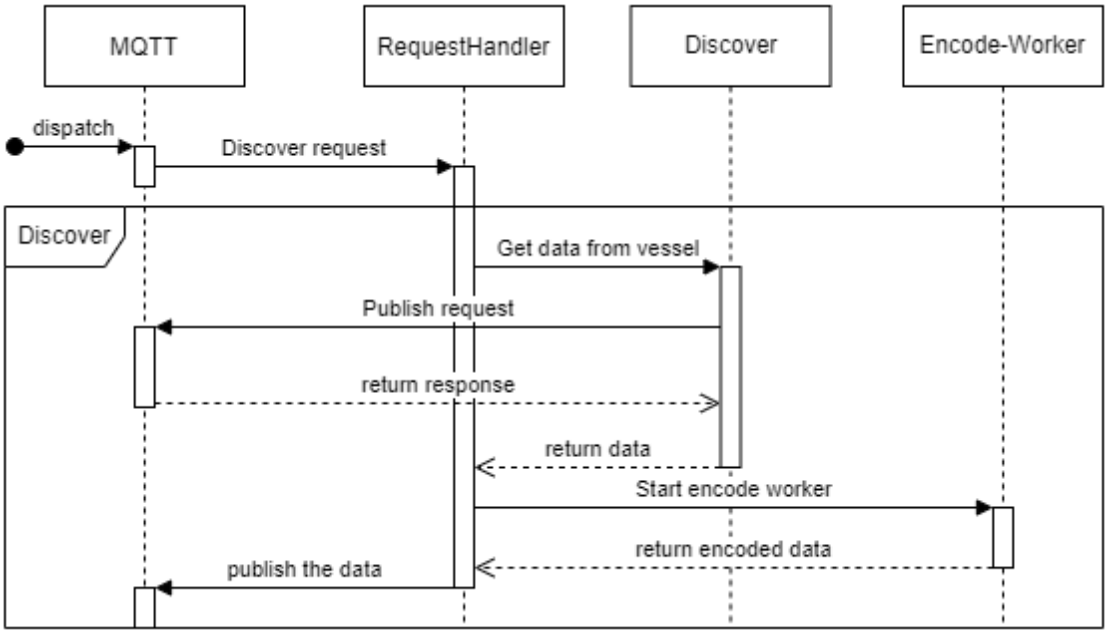


Figure 16: Cache discover sequence

The two simplest request handlers in the cache, GetLastN, shown in Figure 15 and Discover, shown in Figure 16, act as proxies to the requested vessel. The first thing that happens is that the request handler decides what kind of request it is by looking at the given parameters:

- LastN if lastn is set to a number other than 1 and at least one signal is given
- Discover if lastn is set to zero and zero variables are given

After the request handler has delegated the request to a handler, the handler constructs a changes object to retrieve the necessary data,

then, adds a message handler to the MQTT client before publishing the request, followed by a 10-second timeout which will throw an error if not cancelled.

Once the request comes in, the timeout is cancelled, and the data is returned to the request handler as JSON. The request handler then starts a worker which asynchronously encodes the data, as not to block any other current requests from continuing. Finally, the encoded changes object is published to the response topic,

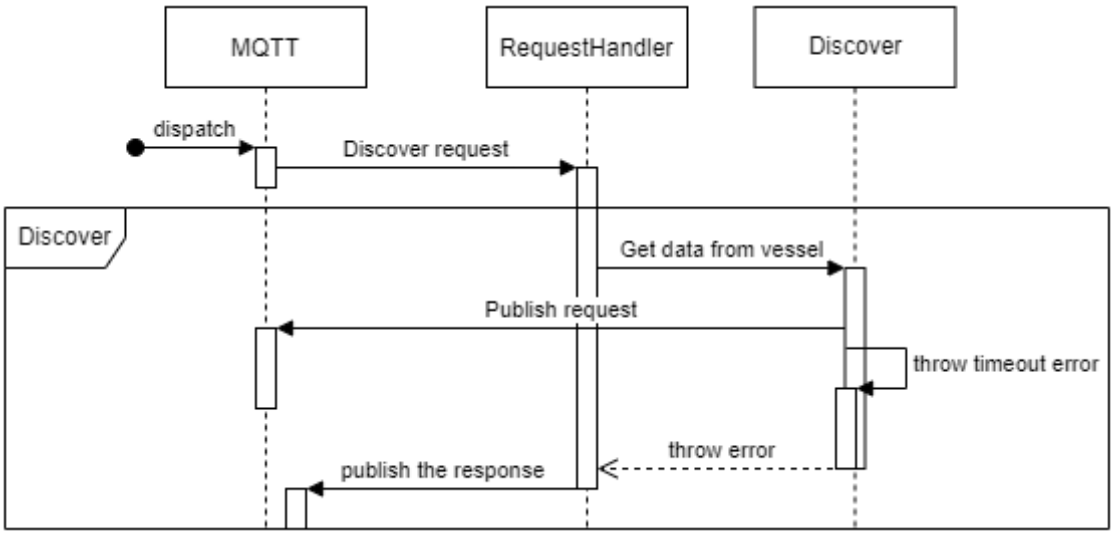


Figure 17: Cache lastn/discover failure sequence

If at any point during a request an error is thrown or a promise is rejected, for example by the 10 second timeout being executed, the error will propagate to the request handler. The request handler will then build a changes object equal to the one in the request, and add an error message and a code, encode it, then publish the encoded changes. This sequence is shown in Figure 17.

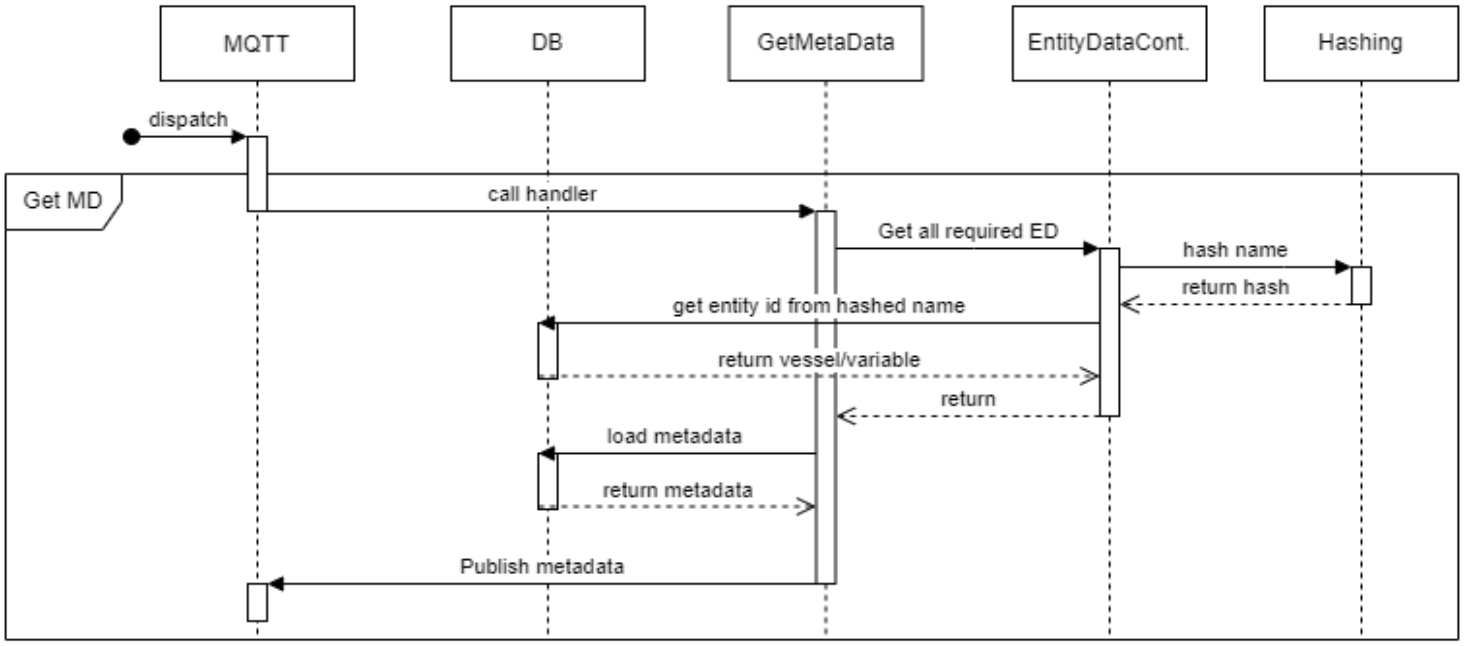


Figure 18: Cache GetMetaData sequence

The cache's GetMetaData handler sequence, shown in Figure 18, is quite similar to the previous endpoints, but since metadata is kept in the database the cache-vessel communication is swapped with cache-database. The GetMetaData responses also tends to contain smaller amounts of data within any given interval, so encoding is performed synchronously to avoid the cost of starting a new worker for every request. The metadata topic also only includes a single request; thus, the intermediate request handler step is omitted. One

thing the metadata handler needs is the id-name-hash mapping of the requested vessel and variables. This responsibility is placed on the EntityData controller. It hashes the name of the given vessel/variable, then retrieves or creates a database record with the information in the vessel table or variable table depending on which type of entity is being requested. This goes for any other request dealing with the database and effectively hides hashing from the rest of the system, allowing other code to use ids or names.
Once the get metadata handler has retrieved the EntityData for all entities, it loads all the relevant data from the metadata table. It maps all the data by variable id, and then goes through each id, and changes the ids out for the names using the EntityData. The end product from this are lists mapped by variable names. It then encodes the data with the MetaDataResponse proto and publishes it on the response topic.

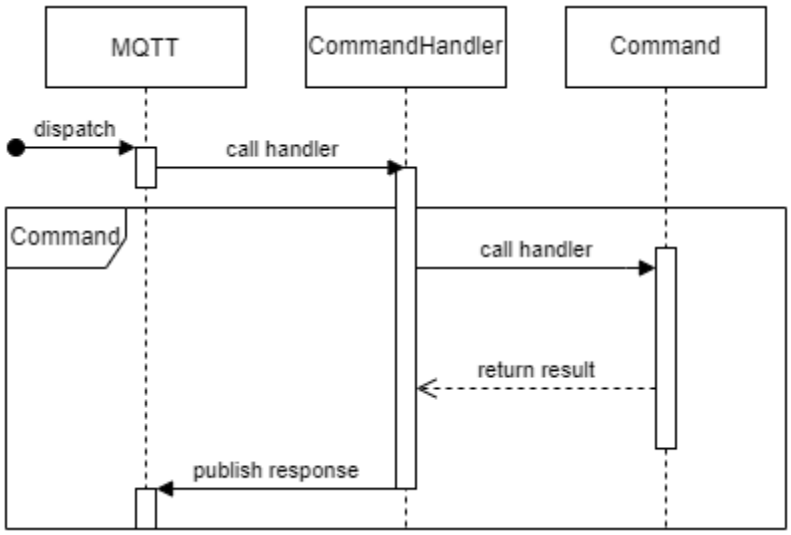


Figure 19: Cache command sequence

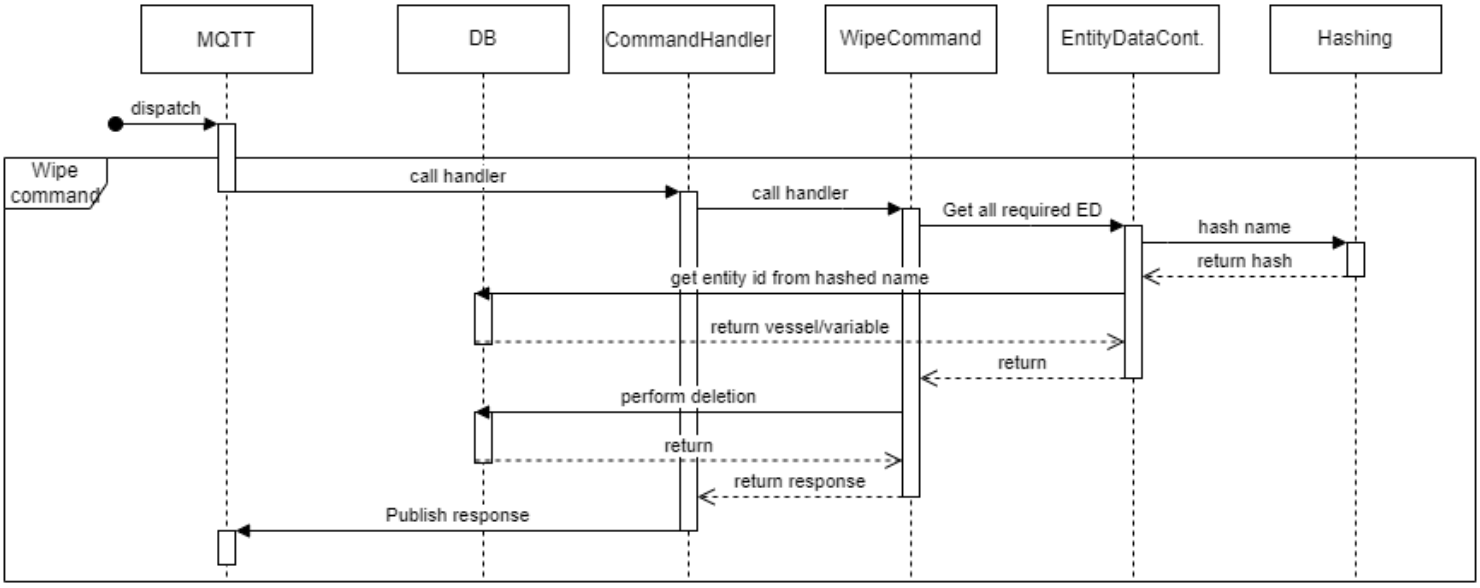


Figure 20: Cache wipe command sequence

The cache also has a skeleton for a command API, with the above figures showing an abstracted overview of how commands are implemented (Figure 19), and the full wipe command sequence diagram (Figure 20).
In general, commands are sent in the command topic, and include an “action” field, which the command handler uses to call the correct handler. Currently, the only command is the wipe command which does the dangerous action of partially or fully wiping the database.
Once the wipe handler has been called, it will use the EntityDataController to get id-hash-name mappings for the given vessel and variables. It will then build a query based on the parameters (For more information, check out the cache service ReadMe). Once the wipe command has completed, it will return a response to the command handler which will encode and publish it. If an error occurs, the error is propagated up to the command handler, and then sent in the encoded response.

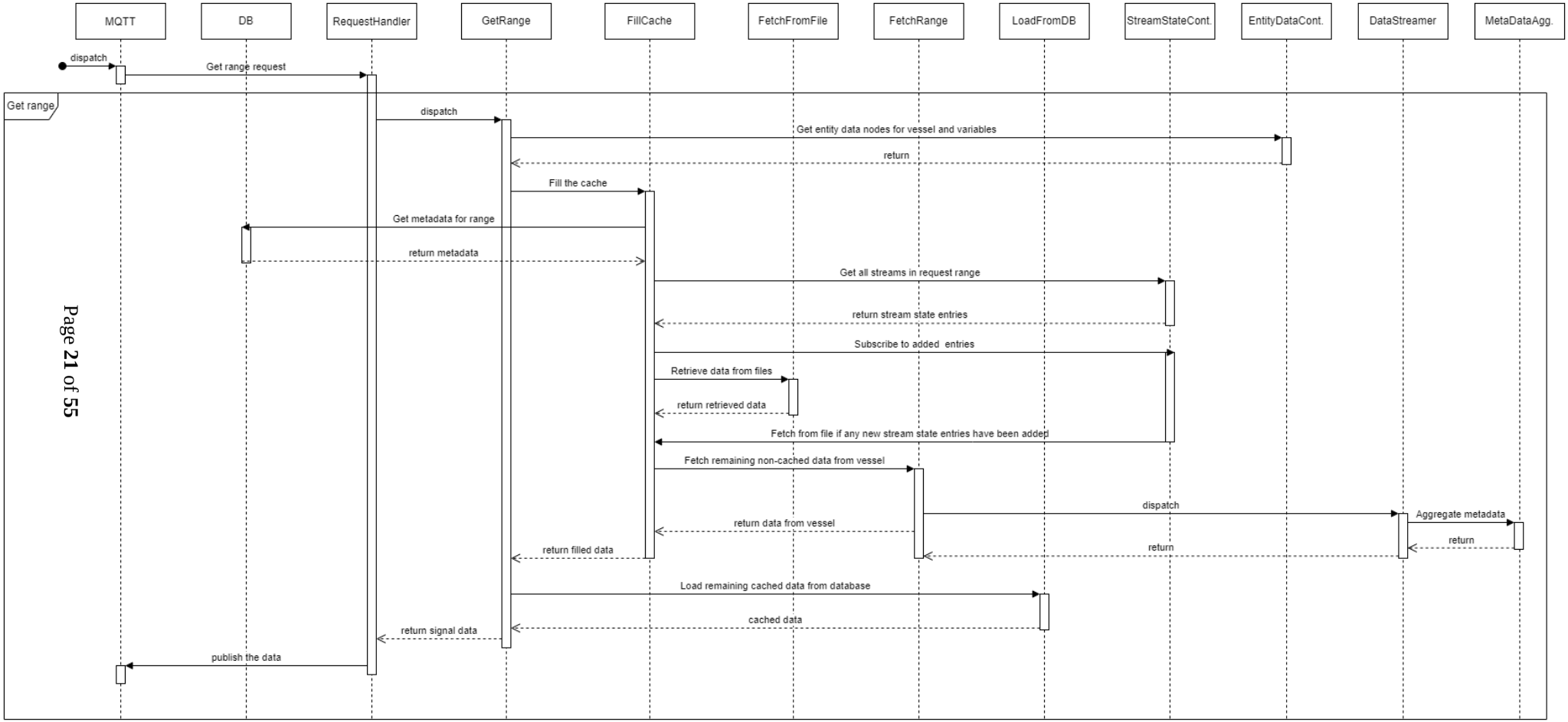


Figure 21: Cache get range sequence

The get range request, shown in Figure 21, is where most of the complexities are used to combine all flows of data in a way that keeps performance high and intensive tasks in the background. The entry point to the get range request is the same as for lastn and discover, but with the fromto and variable fields set (More in cache service ReadMe).
The first thing that the get range handler does is retrieve the entity data it needs. In the above model, some of the component's sequences have been omitted due to the amount of them.
The get range splits a request up in two parts, cached and non-cached data. Fill cache is called after retrieving the entity data.
Fill cache then retrieves all relevant metadata from the database and maps it by variable. It then gets all relevant stream states from the StreamStateController (A stream state represents data in transmission, and a temporary file holding the data). It will then load all data from each file. In addition to loading the data, an observer function is added to the StreamStateController, so if any new streams are added, they will also be loaded to avoid double-fetching when multiple requests of the same data happen at the same time.
After all data in the files has been loaded, fetch range is called for all the remaining intervals (now cached intervals + stream state intervals). Once data is fetched from the vessel, the fetch range function will resolve and start streaming that data into the database using the DataStreamer. Once the DataStreamer has streamed all the data, the metadata accumulated during the fetch request will be inserted into the database and added to the metadata aggregator for aggregation.
Immediately after all fetch ranges are resolved (concurrent to the data streamer and metadata aggregation), the fill cache function resolves. If any cached data exists, it will then be loaded from the database, and all the streams will be reduced into signals-by-name.
When get range resolves, the request handler will asynchronously encode and await the data using the encode worker. This is very important, as without using workers, this could hold up the entire event loop for times above 10 seconds, thus blocking any other requests in the meantime.
After the worker resolves and returns a data buffer of encoded "changes", the request handler publishes the data to the response topic.

In the case where an error is thrown anywhere during a request (depending on whether or not it is recoverable) it will be propagated to the request handler and handled there

REST

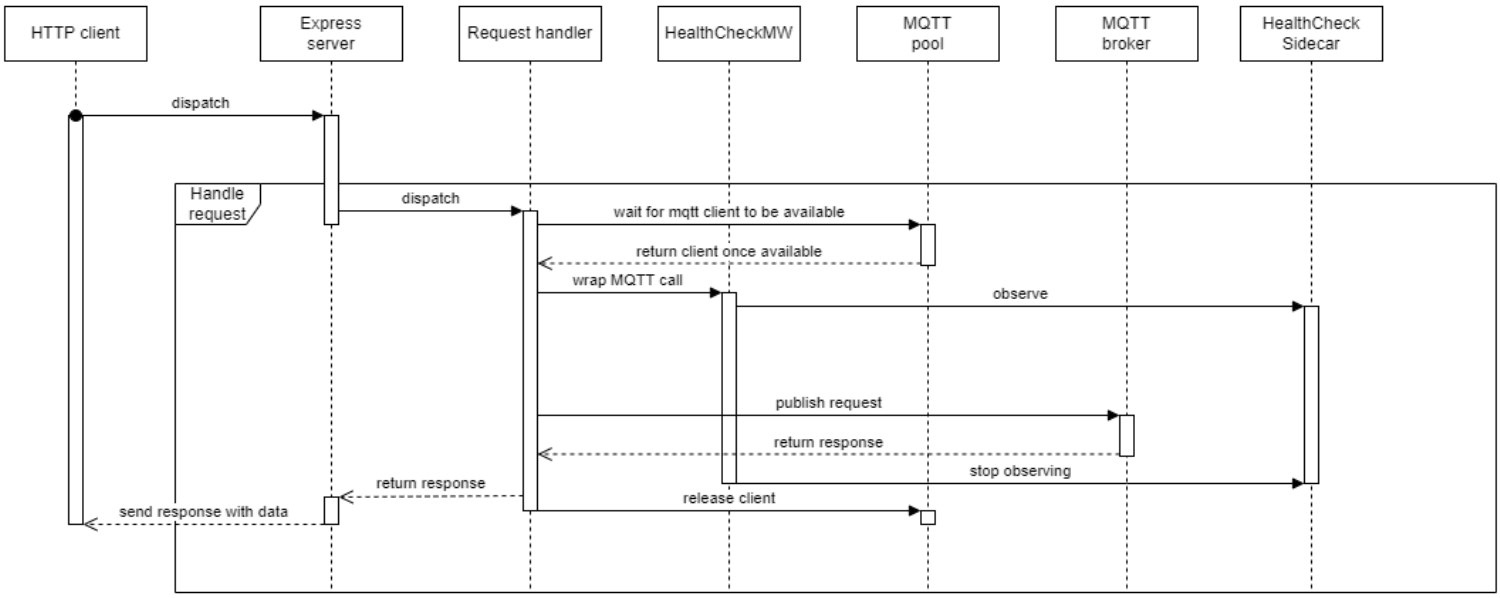


Figure 22: REST service successful request

The sequence of any REST API request, shown in Figure 22, is the same.

1. Perform HTTP request to cache
2. Request handler retrieves MQTT client from the pool
3. Wrap request in HealthCheck middleware
4. Set the request handlers and subscribe to the response topic
5. Publish request to the MQTT broker
6. Get response and disarm healthcheck
7. Return response and release client
8. Send response to HTTP client

If a request fails, for example by the healthcheck throwing, shown in Figure 23, the sequence remains mostly the same.

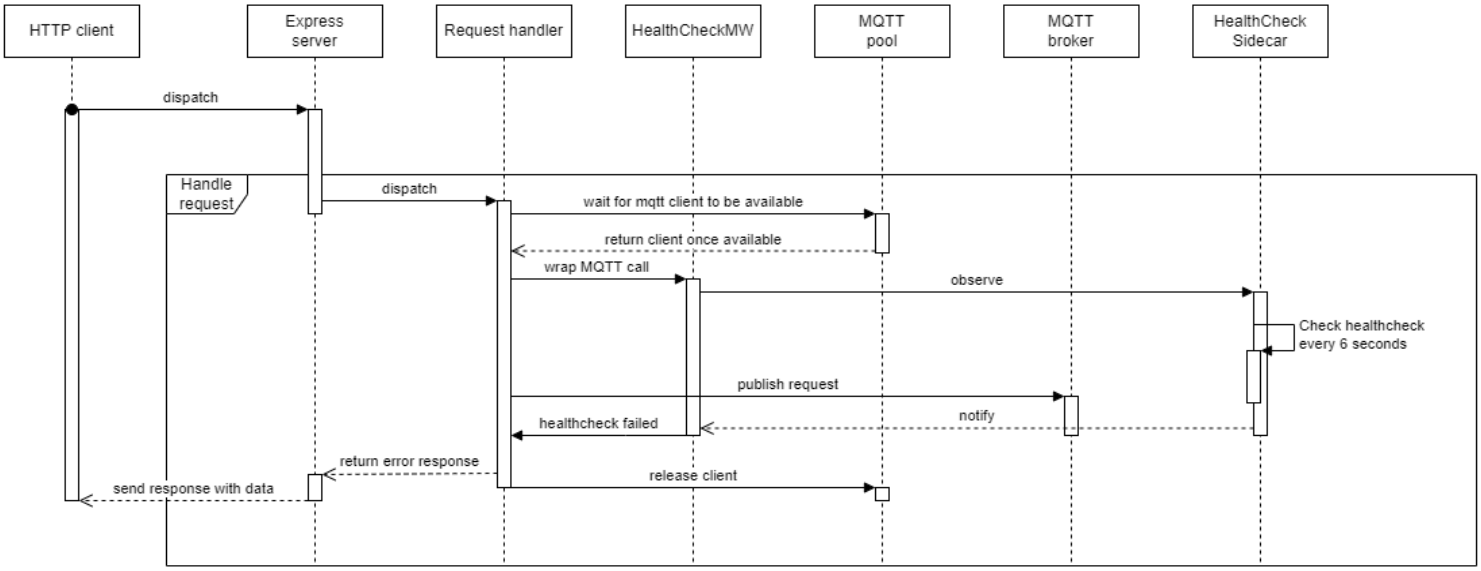


Figure 23: REST service failed healthcheck sequence

If the request for any reason throws before it is resolved, it will cancel all the following sequences and shut down the currently running resources such as the MQTT client.
The Healthcheck sidecar is implemented as a regular class, and not as an actual sidecar process. The reason for this is that through some rough testing, the group could not get any invalid healthcheck errors, so using resources to create a secondary process would just be a waste of resources.

Database model

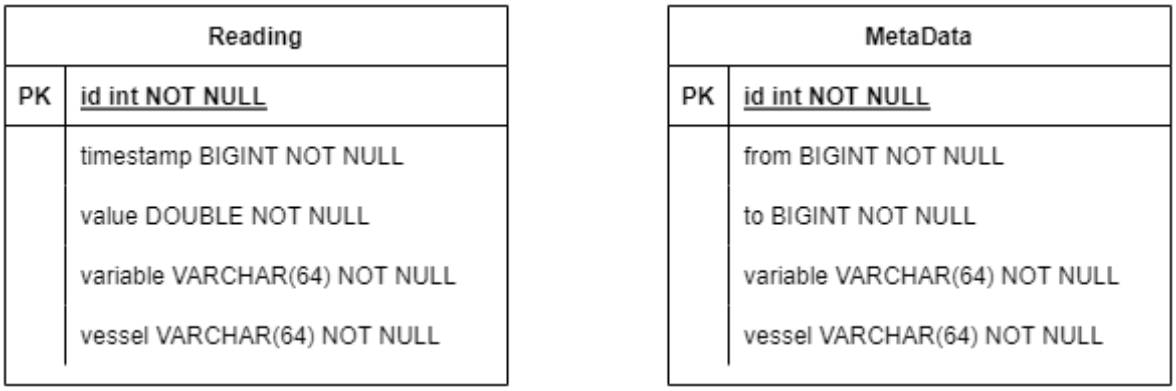


Figure 24: Initial database model

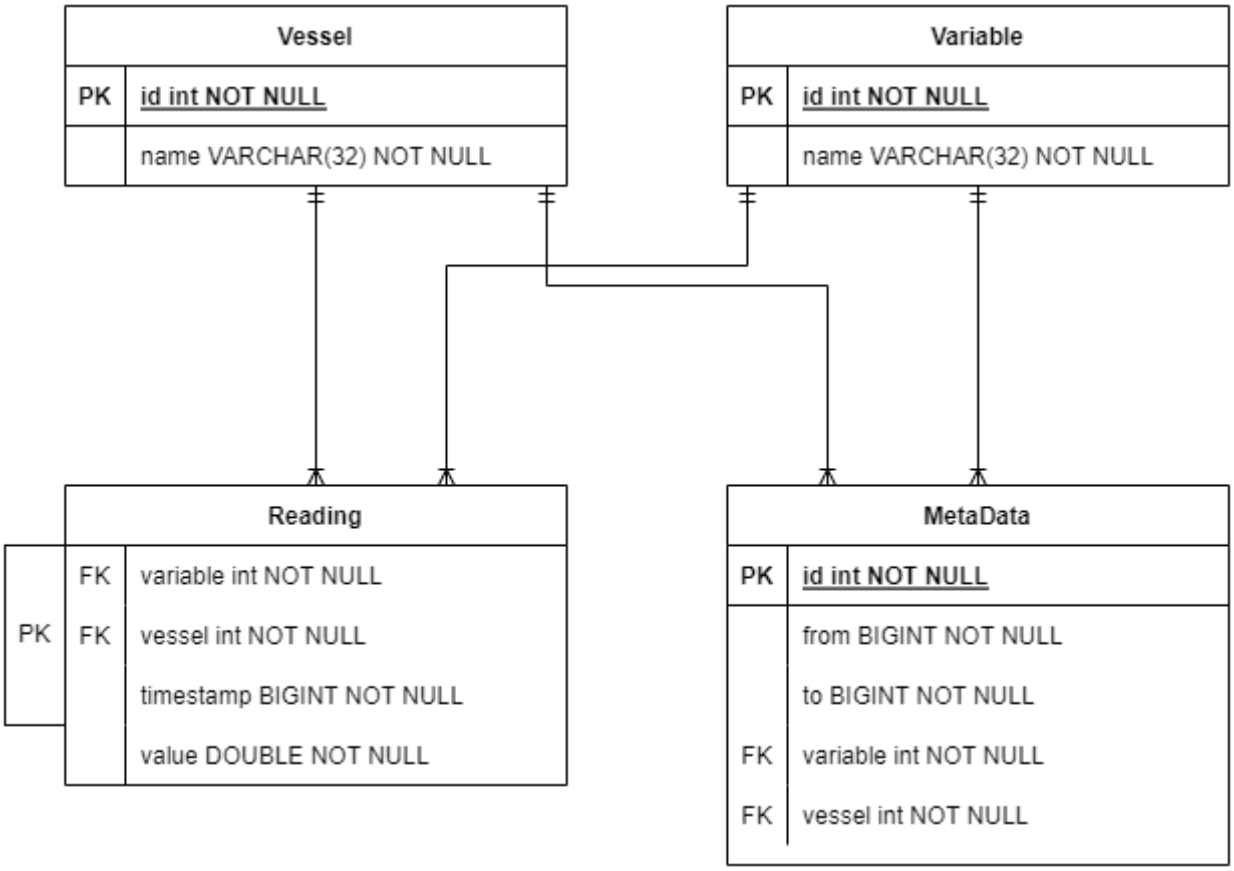


Figure 25: Final database model

The database modelling in this project has been quite simple in nature, since time series data doesn't have many relations, nor is it hard to model. However, the model isn't really the issue when it comes to time series data. Instead, the group has tried optimizing the amount of data used per row of data, as well as optimizing query times, since the number of accrued data entries is likely to grow to a substantial number over time. The team has tried two database models, one straight forward model, shown in Figure 24, and a model where vessel and variable names were taken into their own tables, being replaced by an id to their respective entries, shown in Figure 25

Reading

The reading table represents a single reading from a sensor at a vessel. A reading entry takes up only 24 bytes*. For the sake of lookup performance, especially given the amount of data that will accumulate, the reading table has a tri-column primary key. The primary key is made up of the vessel and value of a record as well as its timestamp. The reason why the key was set up this way is because all database queries to the reading table are based on these three parameters. While the multi-column primary key leads to larger indexes, it also saves 8 bytes per record versus adding a big-int id key.
Additionally, the reading table contains foreign-keys to the Vessel and Variable tables. The amount of unique variable names and vessel names are unlikely to grow to a massive number, and each name is a 64-byte hex-string of a sha256 hashed word. By removing these quite large columns and put them into their own tables, massive amounts of data have been saved, more on how much at the Vessel and Variable subchapter.

MetaData

In the metadata table, a row represents an interval of cached data. This means that by looking at the metadata table alone, you can check what data is cached. This is essential to avoid requesting the same information multiple times, both in terms of how long it takes to find out, but also in terms of certainty. Without the metadata, acquiring that information would involve a lot of guesswork. A metadata record takes up 28 bytes*, but its size is not really important, as the number of metadata that is generated will not be close to the number of readings.

Vessel and Variable

The vessel and variable tables map vessels names and variables names to an id. The purpose of this table is saving data from readings. Since vessel and variable names are up to 64 bytes when hashed, this alone saves 64 bytes per entry, since after any amount of volume, the 68 bytes an entry take up will be insignificant. The addition of these two tables therefore decreases the size of entries by 120 bytes, which is 83% of the original size.

** Counting only raw data, and not any overhead such as indexing, which there will be*

Metadata Aggregation

The number of metadata increases over time and can become quite substantial when many smaller requests are performed. To both maximize the efficiency of storage and querying, the group created a metadata aggregator. The metadata aggregator looks at the recently added metadata and looks for adjacent metadata intervals, if any relevant intervals are found, they are fused into one single interval. If for example, 100 requests spanning 10ms each from 0 to 1 second are executed you would have 100 metadata entries. The metadata aggregator would then look at these recently added intervals, sort them and go through the list fusing one and one interval, with the result being a single interval spanning the whole second.
Both a timed loop and a manually run loop have been tested with the choice falling on the timed run loop since it’s easier to control and the manual loop not showing any benefits. The manually run aggregations would have a tendency to make overlapping requests lead to invalid metadata states with overlapping metadata intervals etc. One compromise was, however, made when going with the timed loop, as by calling the waitForRun method, and awaiting it, would pause execution until aggregation has completed.

Server-services

Each backend services API is defined in their README files. These can be found in their source code which is an attachment to the main report.

Security

The security approach taken by the system is highly passive. Per the request of Seaonics, no user authentication has been implemented. Therefore, the system relies on edge-authentication and edge-authorization, where anyone with access to the network on which the system runs, have full access to its functionality.
The same goes for encryption, the system itself does not provide encryption, but can be run behind TLS termination proxies to encrypt traffic. However, running the systems dependencies (Database and MQTT) on non-TLS connections require the explicit disablement of TLS through the environment variables. The example/test environment running at wirelogger.com, for example is using purely encrypted communication.

The main considerations when setting up the system in terms of authorization are

- Who can access the REST service?
- Who can access the MQTT broker?
- Who can access the database?

Setting up authorization should therefore be fairly easy if wanted, for the MQTT broker and the database, a solution could simply be to require password authentication, whilst utilizing TLS.
For the REST service, one can add authentication middleware using express and some external provider such as Google Firebase if you don’t wish to do it yourself

One security measure found relevant to the services was the hashing of data names, in case of unwanted database access. Therefore, all name values within the vessel and variable tables are hashed using SHA-256 with names only being available in the runtime environment during the request. Once a request has ended, the name is deleted from the runtime environment also. The reason this is relevant is that the signals being logged can contain sensitive data, but also the fact that the names of the variables and vessels being logged may give unwanted insight into how Seaonics operates.

Installation and use

The simplest way to run the time-series system is to use the docker compose in the docker repo, which is a folder in the source code found in the main reports attachments.

The following dependencies chapter goes more in depth on how to set it up without using docker compose.

Dependencies

To run the system, you need either node or Docker to run the actual services. Additionally, to make the system work you need to set up an MQTT broker and a Postgres database with the TimescaleDB extension.

The first thing you need to do is to set up the MQTT broker and the database, as the system will not run without them.
The database is quite easy to set up as Timescale provides free DBaaS, which can be found here at the timescale website [1].
Alternatively, you can use their docker image timescale/timescaledb, which has been used during development. For further instructions on setting up the timescaledb database using docker, you can check out the caching services readme found in the source code.
The MQTT broker can also be set up easily by just using a hosted service with HiveMQ [1].
Optionally, it can be set up with docker, which also is the method used by the group. For further instructions on using docker to set up a broker, check out the previously mentioned cache service readme.

While the actual vessel service is not implemented, there exists a simple mock providing randomized responses according to the vessel services API. For more information on how to set it up, please check out the readme of the cache service

To run the actual services there are two choices, running them with node, or running them with docker. For further information about how to run the individual services and their required environment variables, please check out their respective readmes.

Website use

Intro

The following segment will give insight into how a user can add a vessel to the site, browser for signals, view multiple signals at the same time, panning and offsetting the chart for better visualization, there will also be a short display of how to use the command line interface page.

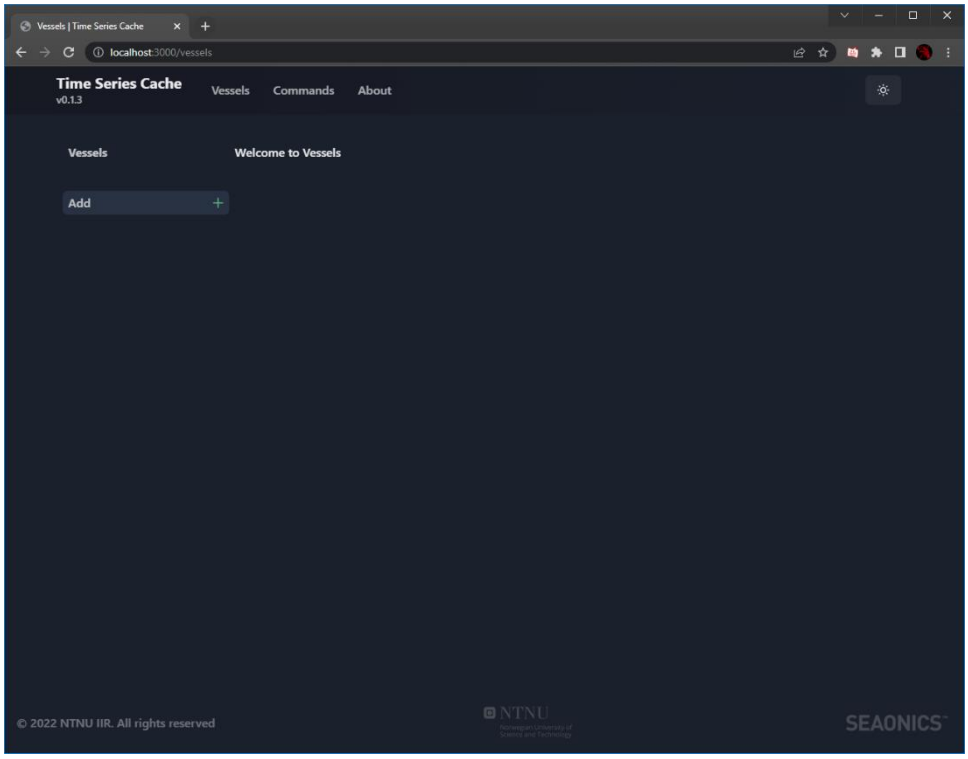


Figure 26: Website after opening

When opening the website for the first time, a user will be sent to the "/vessels" route as seen in Figure 26

Here the user can see a navigation bar at the top of the screen, it has links pointing to the Vessels Page which they are currently viewing, the Commands Page, and the About Page. Lastly it has the toggle for light and dark mode all the way to the right side of the screen, which will be discussed later in this segment.

Continuing down, the user can find the vessels side bare and the main content of the Vessels Page. To the left side the user can add new vessels by clicking the "Add" button. When clicking this button, the user will be prompted with this modal. Here the user can insert the name of the vessel they wish to add to the site.

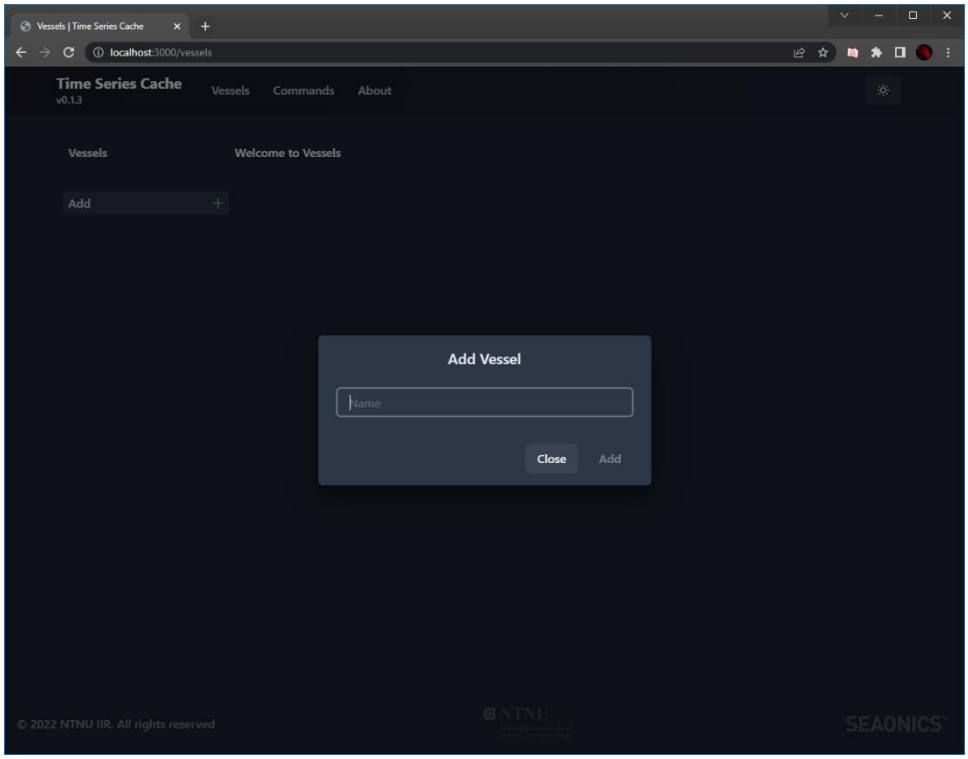


Figure 27: Vessel modal

After submitting a vessel name the user can then exit the modal by clicking “Add” button. As seen in Figure 27.

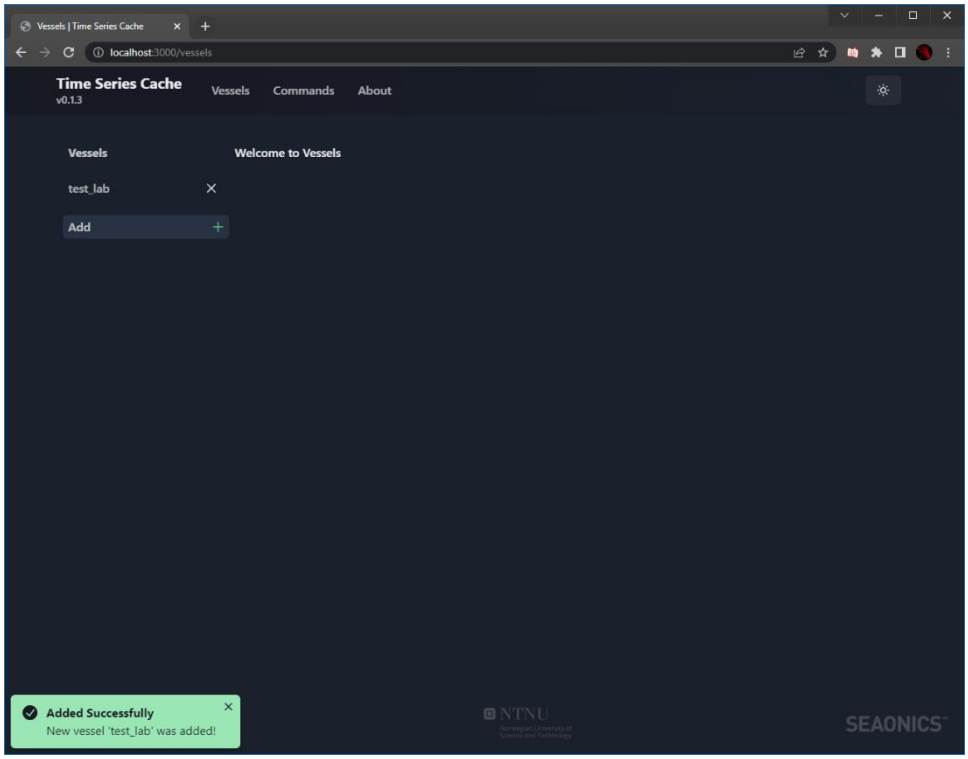


Figure 28: Successfully added vessel

When successfully adding a vessel to the site, the user will be prompted with a success modal. The user can also remove these vessels from the site by clicking the “x” icon next to the signal name, as these are stored in the browser’s local storage for testing purposes. As seen in Figure 28.

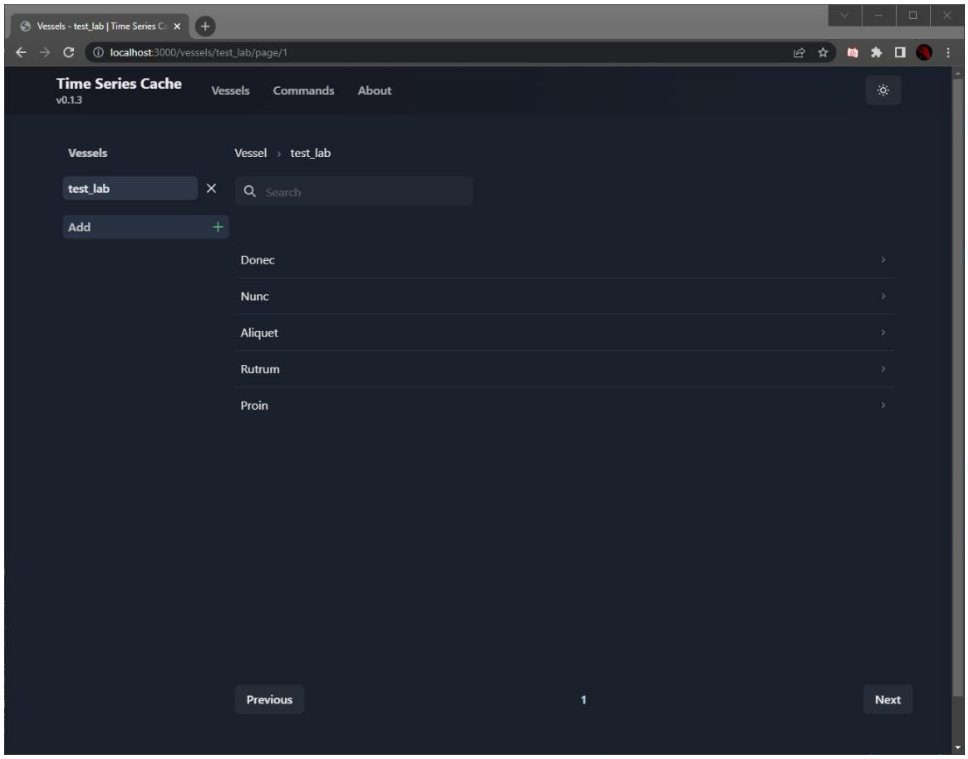


Figure 29: Browsing signals

If the user clicks on the added vessel, a list of available signals for this vessel can be found, shown in Figure 29, for testing purposes there exists some randomly generated signals for any added vessel, for fully integrated version of the Time Series cache a user will navigate Seaonics library of vessels then get all available signals for that vessel. For real life use signals are usually in the hundreds, therefore, a search-field as well as a paged browsing view is important for the user to be able to find what they are looking for without struggle.

Signal Viewer

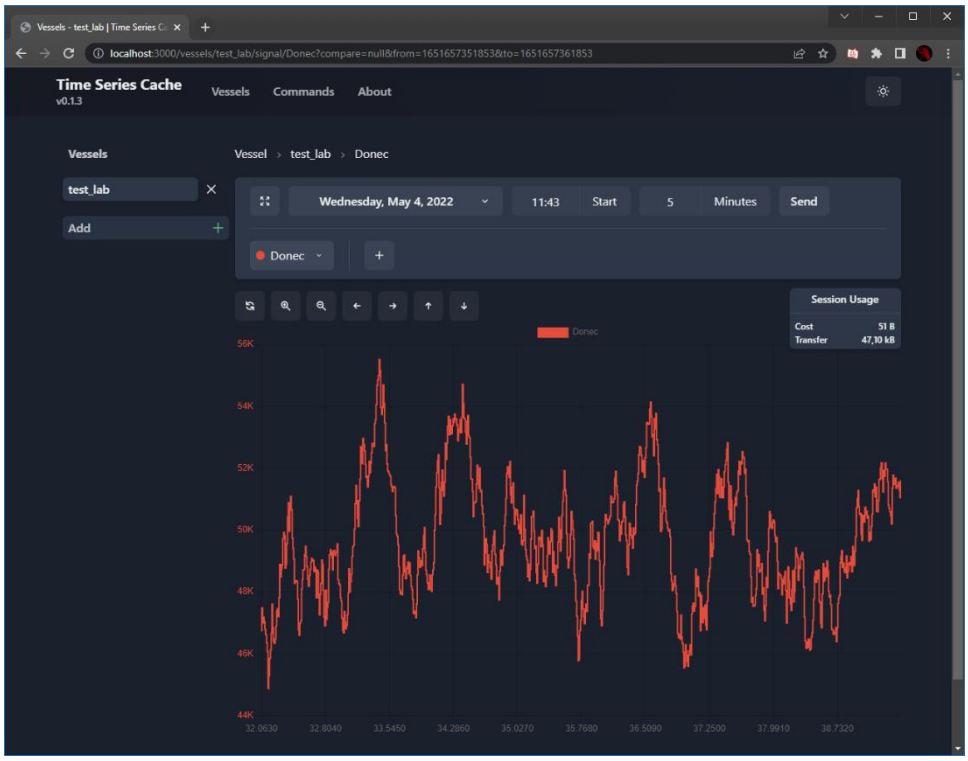


Figure 30: Initial Signal load

After selecting which signal the user wants to review, they will be greeted by the Signal Viewer page, shown in Figure 30. Seen in the chart area are the last 10 seconds of data loaded as a starting point.

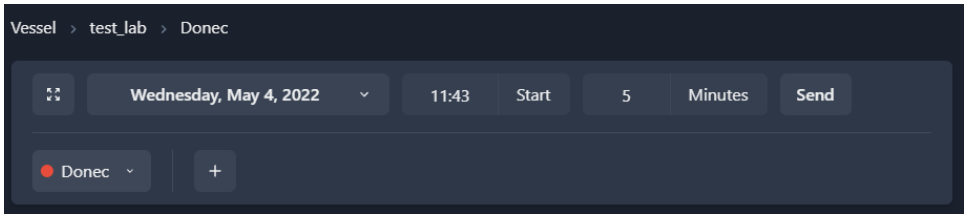


Figure 31: Range controls + Signal controls

At the top of the signal viewer page the user can see the breadcrumbs of their choices as seen in Figure 31. They are currently on the Vessels page, viewing the “test_lab” vessel and reviewing the Donec signal as their focus point. Below this, the user can see the various signal related controls. From the left, there is a full screen mode button, clicking this button will expand the controls and the chart area to take up the entire screen. Next is the date selector, which is by default set to the current date, after this there are the hour and minute offset inputs followed by the submit, or send, button. To use the viewer, do the following:

1. Select a date as a starting point for fetching data
2. Select a start time within the selected date
3. Select an offset time to review from the selected start time, this adds these minutes to the start Unix time of the request. Therefore, if the user were to request 5+ minutes from current time, the data fetched would be minimal as it will only retrieve data up to those milliseconds that have passed since the page was loaded.

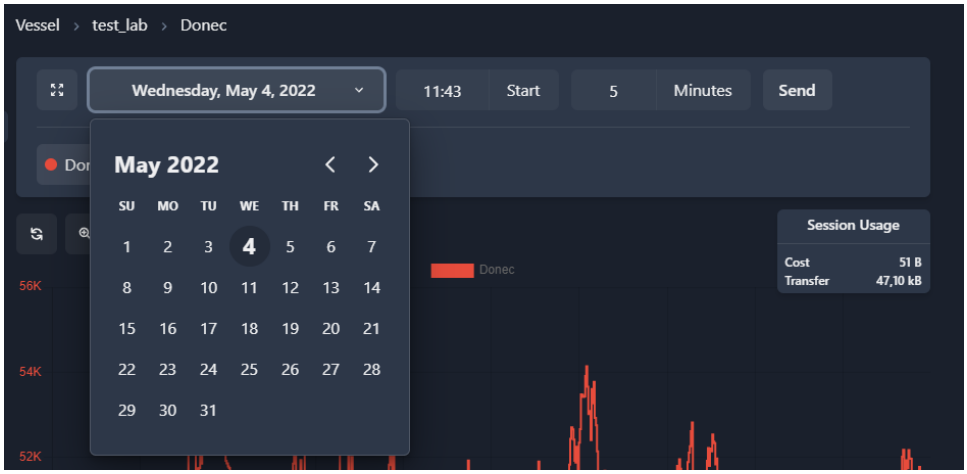


Figure 32: Date picker

Figure 32 is how it looks when the user is selecting a date, the current date will be displayed with a bolder and bigger number than the rest, and the selected time will have a darker background color than the rest.

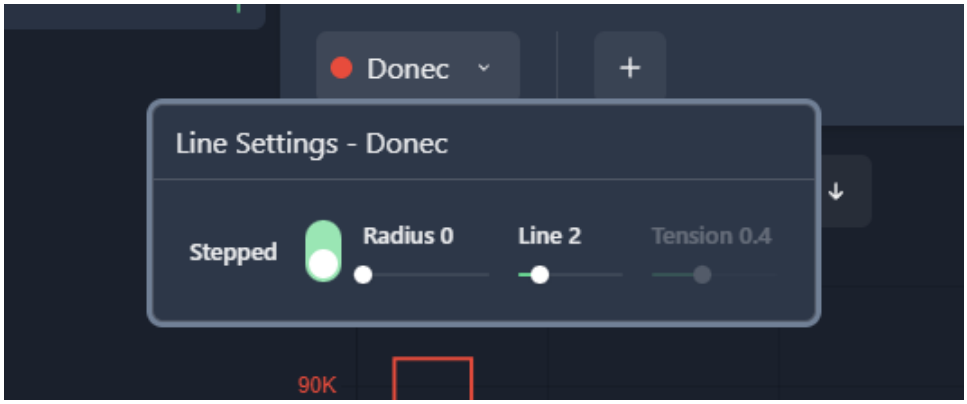


Figure 33: Signal Settings

Next are the signals. To the very left of this row, the base signal can be seen with its own color, separated from any additional signals. Next to the signal there is an add button for adding additional signals for comparison. The user can add up to 5 additional signals, this max capacity was added because of performance. As fetching larger ranges with more than 5 signals performed under the wanted response time wanted by the user.

Each signal has a visual settings dropdown, shown in Figure 33, which handles how the line is presented in the chart. In the dropdown, the user can adjust four different changeable options, The first is a toggle for toggling between stepped and not stepped line. This defaults to stepped as per request by our test users. Next is the radius of each datapoint that makes up the line seen in the chart, these are by default set to 0 as rendering bigger radiuses for each datapoint is performance heavy.
“If you have a lot of data points, it can be more performant to disable rendering of the point for a dataset and only draw line. Doing this means that there is less to draw in the canvas which will improve render performance” [1]

To test this a simple test like drawing 3 datasets in the chart given 500 datapoints each, then drawing the lines and points in the chart, The JavaScript heap result seen in Figure 34 is the result of moving the cursor around to render the tooltip for displaying the data for each point

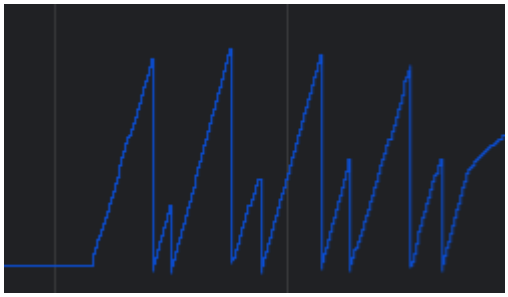


Figure 34: JavaScript Performance Heap with Points render

In Figure 35 when doing the same mouse movement and all datapoints are turned off the heap grows a lot faster and has lower peaks, the browser spends a lot less time drawing, and the site overall feels a lot more responsive

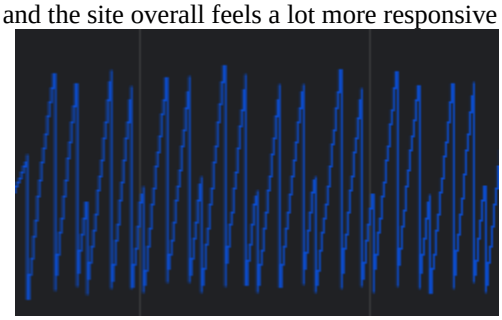


Figure 35: JavaScript Performance Heap without Points render

Then there is the line width which decides how thick the line will be drawn. Lastly there is the tension slider, this dictates the factor of the Bezier curve that is drawn between each data point, this option can only be used when stepping is turned off.

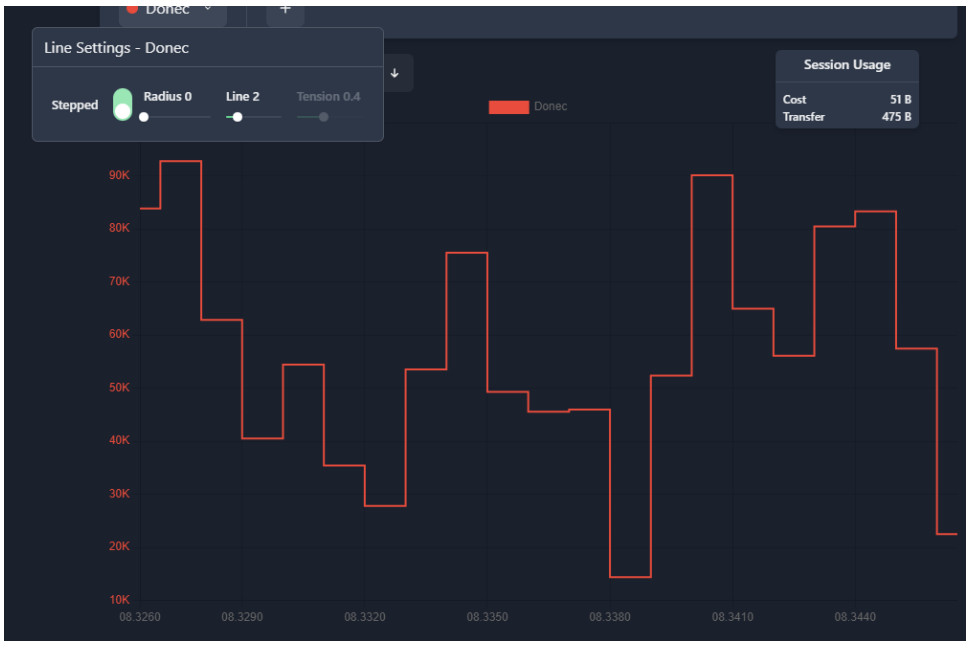


Figure 36: Stepped line example

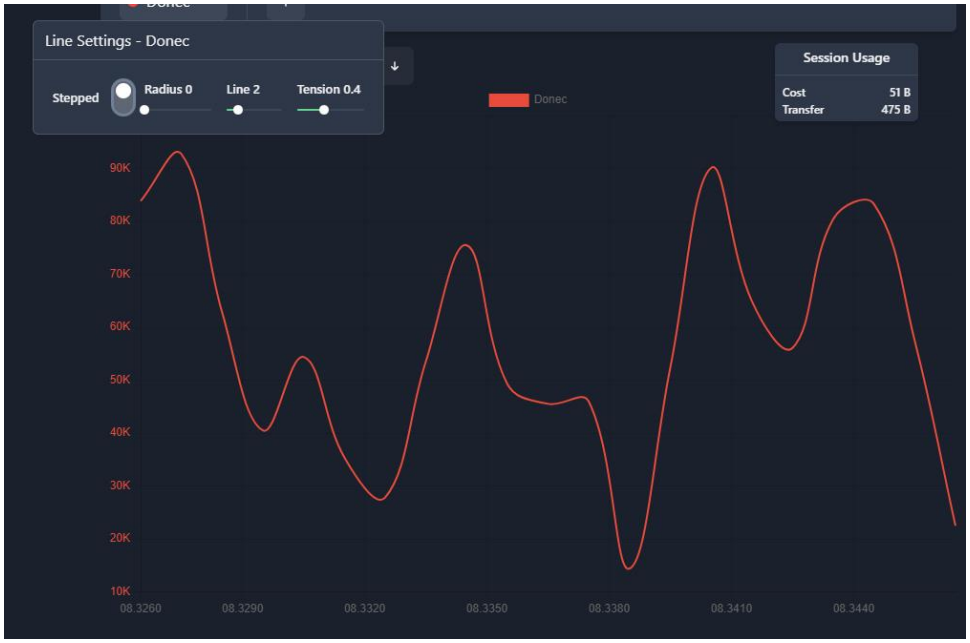


Figure 37: Line Tension example

Figure 36 shows the signal viewer chart drawn with stepping. Figure 37 shows the signal viewer chart drawn with stepping turned off, and tension set to medium, here the user can see the Bezier curves being drawn between each point.

Figure 38: Chart tooltip example

When the user hovers their cursor over any drawn data found in the chart, an animated tooltip containing the exact date, signal, and value of the nearest point is displayed over the graph, this can be seen in Figure 38.

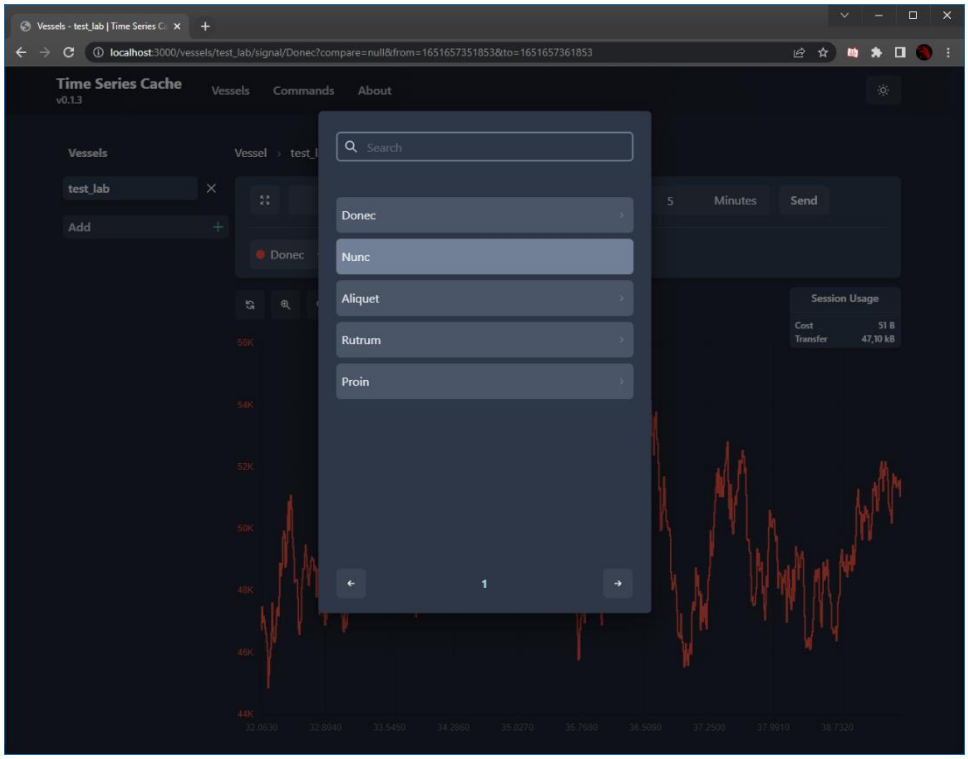


Figure 39: Add signal modal

If the user wants to add any signals for comparison to the chart, they can click the plus / add button found on the signal row. They will then be presented with a modal mirroring the signal browsing page seen earlier. This contains the same functionality as earlier. An example of this can be seen in Figure 39.

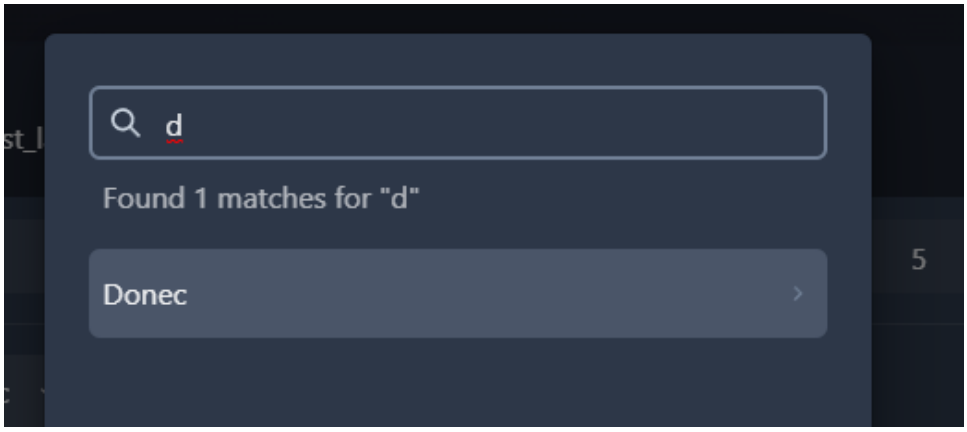


Figure 40: Search for signal to add

The user can search for signals the same as Signal browser, matches for the search key will be displayed as well as a hit counter beneath the search field. After finding the wanted signal, the user simply clicks the list item to add it to the chart area. As seen in Figure 40.

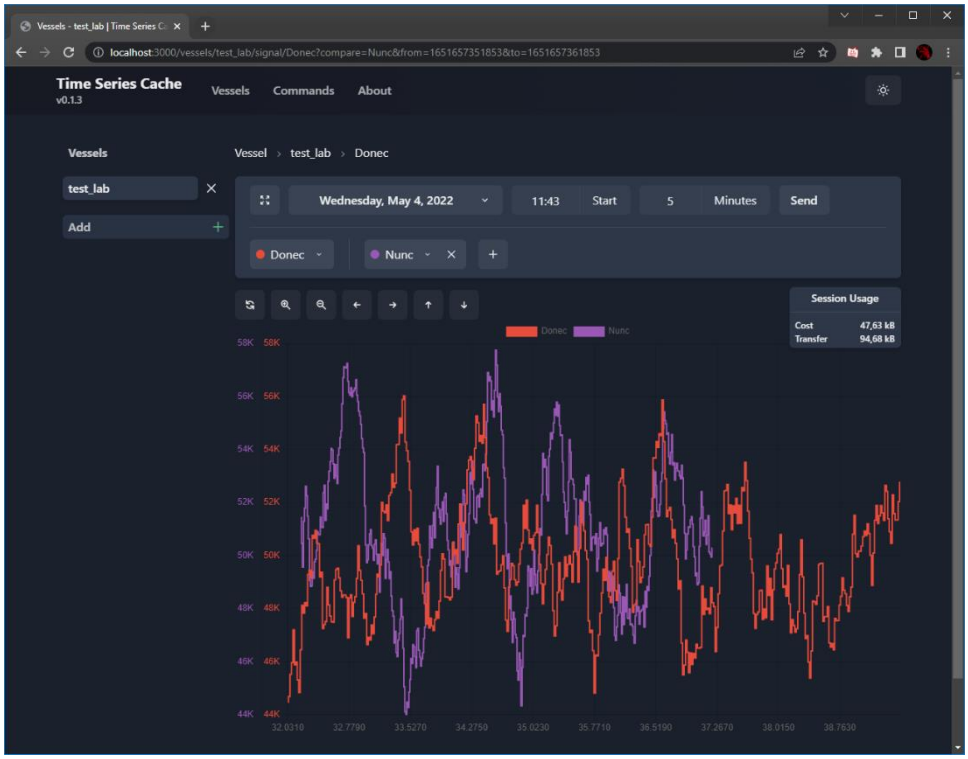


Figure 41: Initial load after adding signal

After adding a new signal, the current range of time being viewed is fetched for the new signal, and the data is then drawn on top of the data that is already being displayed, as seen in Figure 41.
With the help of some customization of the ChartJS library, the different datasets are drawn on top of each other with each dataset getting their own y-axis that matches the color. This comes in handy when viewing data with different ranges of values.

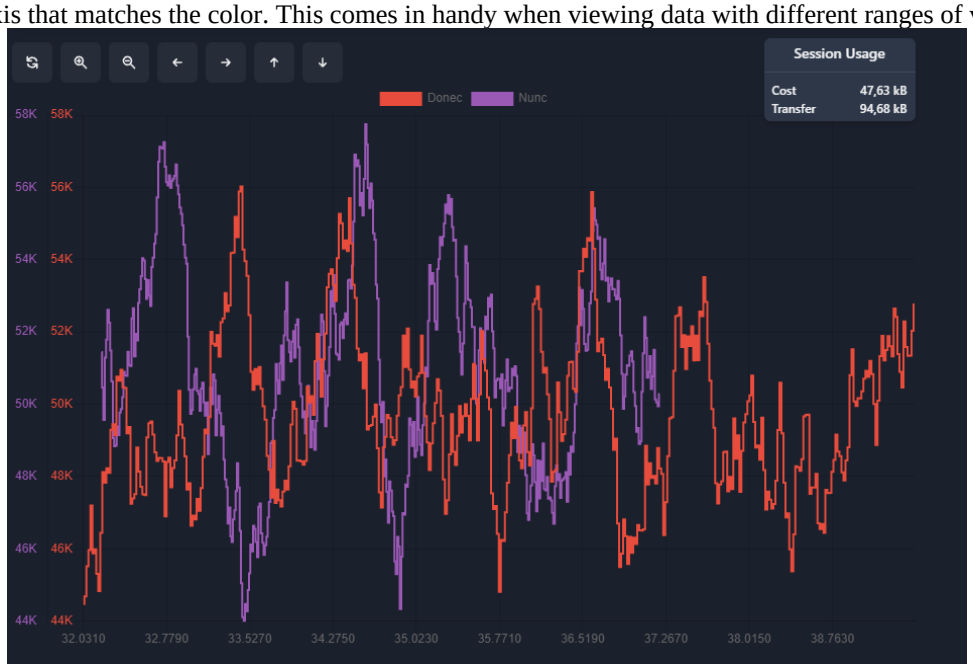


Figure 42: Close-up of signal load

There are multiple ways of manipulating the chart to make the comparison of signals easier. At the top of the chart area, navigational buttons can be found. With these buttons the user can:

1. Reset all the zoom levels, this will make charts span 100% of chart width and height.
2. Zoom in, to focus closer on the current viewing range
3. Zoom out, to view a wider range of data

4. Pan left, to fetch more data into the past
5. Pan right, to fetch more data into the future
6. Pan up, to offset all charts downwards in the chart area
7. Pan down, to offset all charts upwards in the chart area

At the top right of the graph the session usage display can be seen in Figure 42. Here the user can view how much bandwidth is used between the vessel and the cache, in the “Cost” field, and how much bandwidth is used between the website and the cache, in the “Transfer” field. This way, the user can remain conscious of how much data they are pulling from the vessels, as well as how much data the current session has accumulated from the cache.

Another form of chart manipulation is hidden into the chart area with the use of drag events and scroll events:

Manipulation that affects all datasets equally, x-direction:

1. Panning in the x-direction can be done by clicking and dragging on the chart area (all available space to the right of the y-axis), this will offset the datasets in the x-axis and fetch the new range based on the min and max time of the chart area after dragging has stopped.
2. Scaling In / Out in x-direction can be done by having the cursor hover over the chart area and scrolling. This will either zoom in on the area the user is currently hovering or zoom out to fetch more data on both sides of the dataset to see a bigger range.

Manipulation that affects each dataset individually, y-direction:

1. Offsetting the datasets in the y-direction can be achieved by having the cursor above the wanted datasets y-axis column at the left side of the graph. For example, if the user wants to offset the red dataset, they will have to hover above the red y-axis and then click and drag up or down.
2. Scaling up or down in y-direction can be achieved by having the cursor above the wanted datasets y-axis, then scrolling up or down, to scale the overall height of this dataset up or down.

Figure 43: Signals after scaling and offsetting

Figure 43 is an example of the result of manipulating the datasets with the more advanced functionality. Actions used to achieve this result are:

- Scrolling on the general chart area to achieve a larger range than initially fetched after site load
- Scrolling on each dataset's y-axis to make the expanded height of each dataset take up around 20% of the complete chart height.
- Clicking and dragging on each datasets y-axis to offset the datasets from each other for ease of viewing.

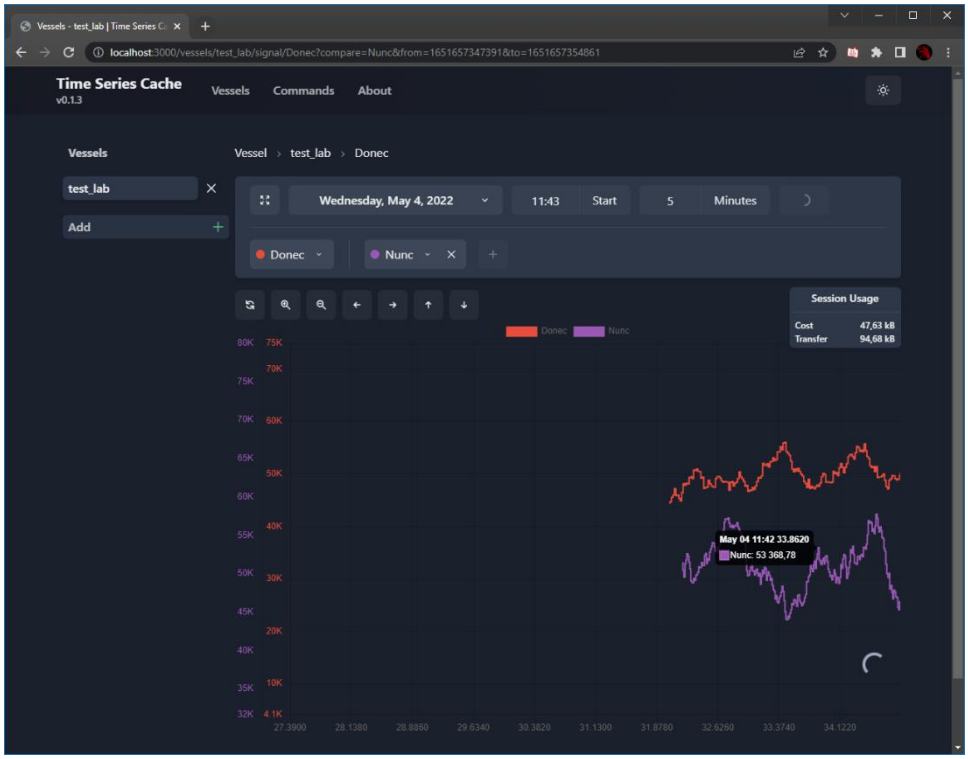


Figure 44: Initial pan into past + fetching new data

Figure 44, example of how the chart reacts to panning into the past. The previous data can be seen pushed to the right of the chart area, and a loading indicator is added to the bottom right of the chart area as well as the submit button to signalize that data is loading.

Usually if the range that is being requested is small enough, the loading will stop, and the chart will be rehydrated with the new range as seen in Figure 45.

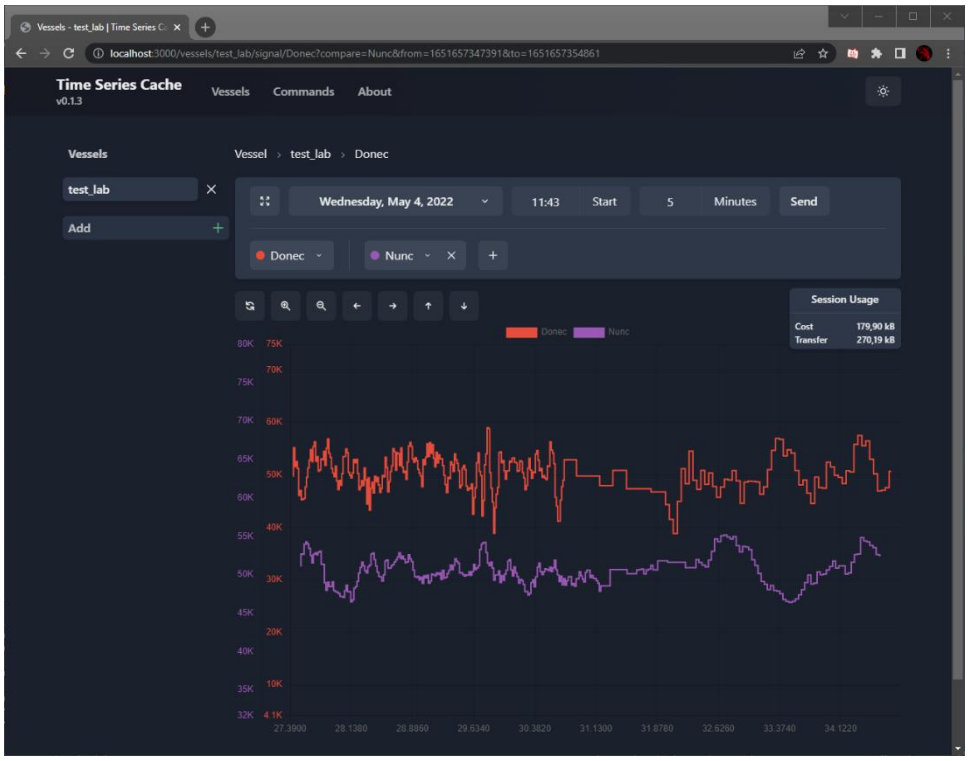


Figure 45: Result of fetching new data

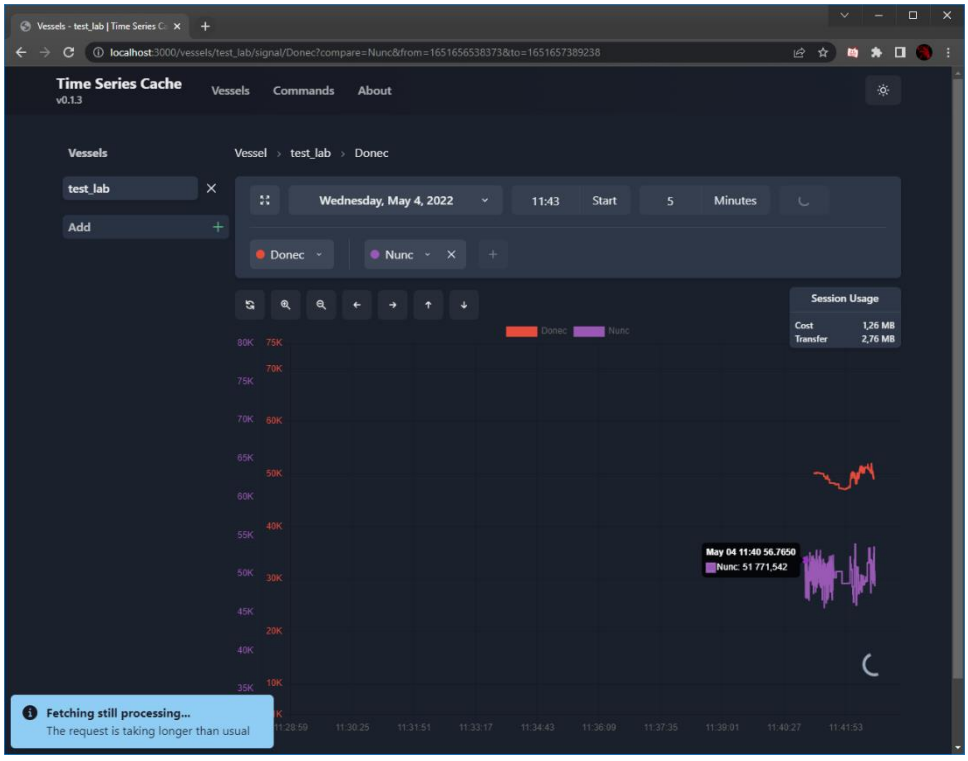


Figure 46: Example of request surpassing 7 seconds

Figure 46 example of how the site reacts to a request taking more than 7 seconds, it will prompt the user with an information box as seen in the bottom left corner of the site.

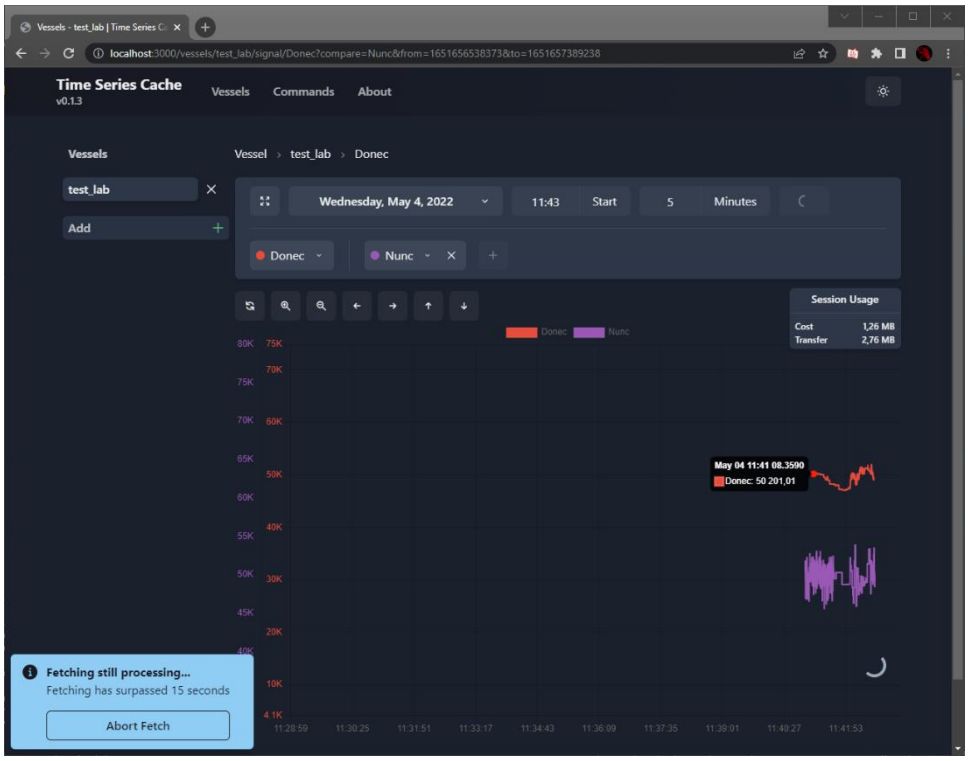


Figure 47: Example of request surpassing 15 seconds

Figure 47, example of how the site reacts to a request taking longer than 15 seconds, the information box will now contain an “Abort Fetch” button that will abort the request if the user is not willing to wait for the data. After aborting the user will be free to fetch a new range.

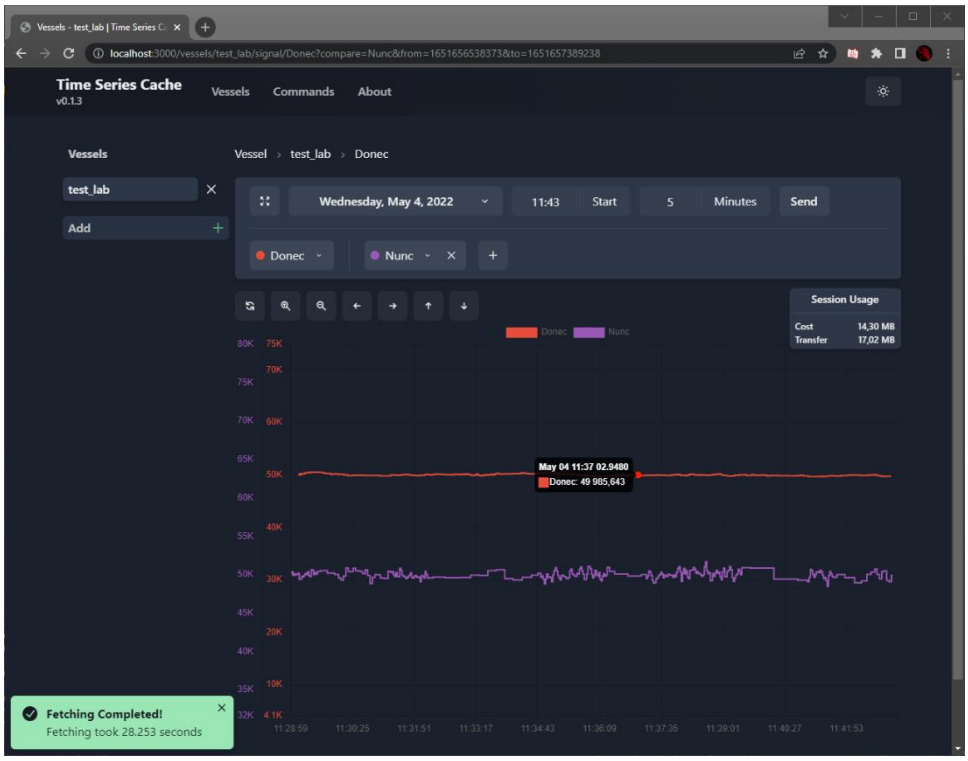


Figure 48: Example of longer request after successful response

When the request finally comes through, the information box will be updated as seen in Figure 48, if the request is taking longer, and the chart area will be rehydrated with the data that was fetched. Although as seen in Figure 48, the scaling the user did previously can affect the drawing of new data, which in this case makes the lines quite slim.

A quick summary of all information boxes:

- Blue: Information, something is being processed and is waiting to be resolved.
- Green: Successful action or resolving process
- Yellow: Warning, something did was requested outside the criteria
- Red: Error, something went wrong, either services are down or internal error (Staus: 500)

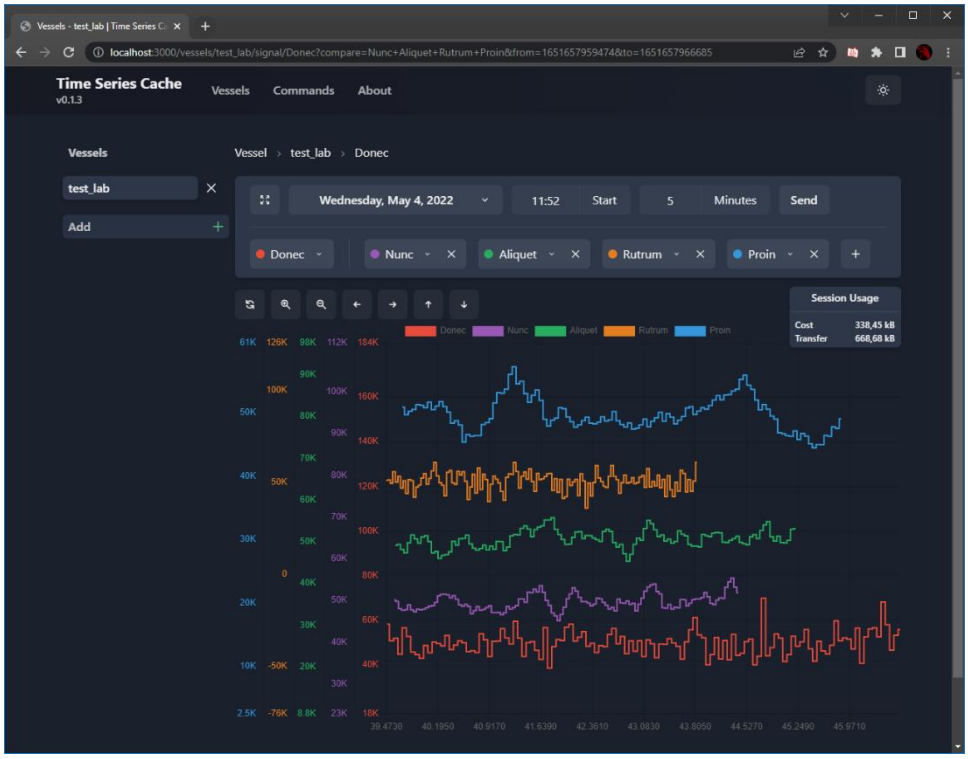


Figure 49: 5 datasets with scaling and offsetting

Figure 49 shows the signal viewer where the user has added 5 signals for viewing. Scaling and offsetting have been done to make comparison easier. In this example the variation in values for each data can be seen by the difference in max value of each individual y-axis.

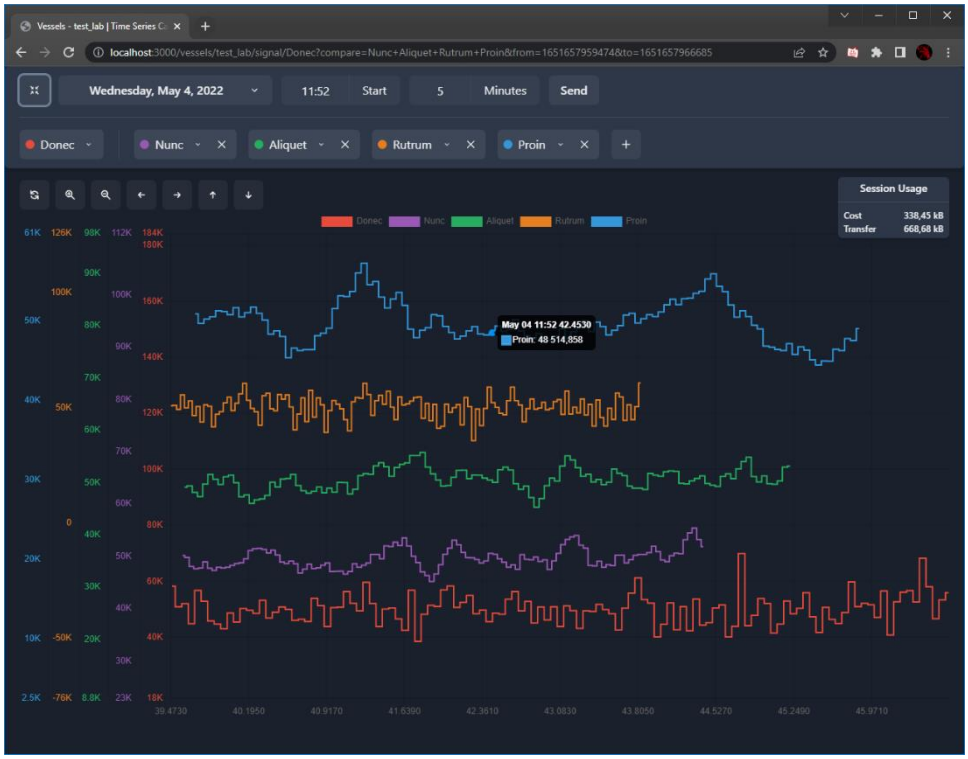


Figure 50: Full screen mode

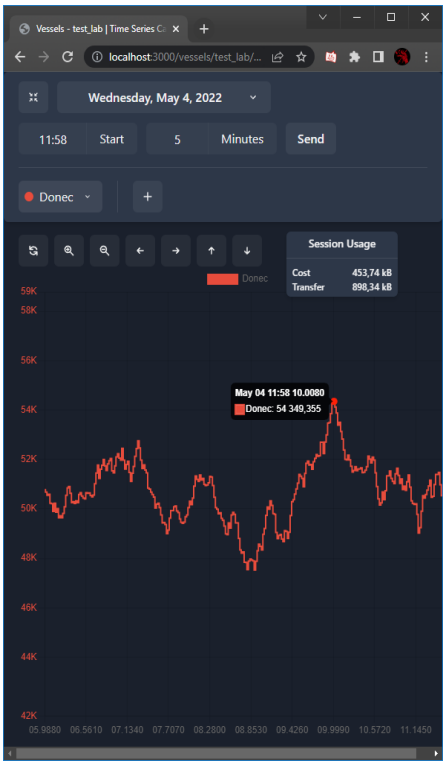


Figure 51: Mobile mode

As mentioned previously, there exists a full screen mode for ease of use. This example shows the same data as seen in Figure 50. Figure 51 is an example of the responsive design in action, here the window has been changed into a more mobile-like shape, resulting in the columns being shifted into rows, as well as the chart area being resized to fit the smaller window size.

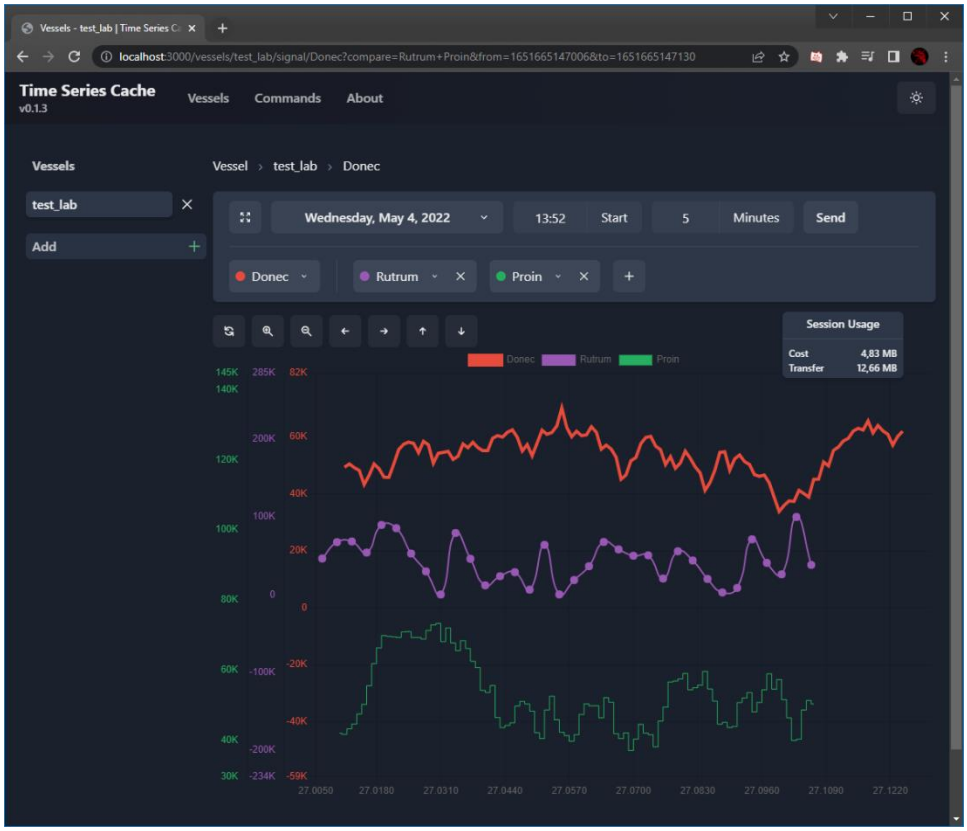


Figure 52: Dataset styling example

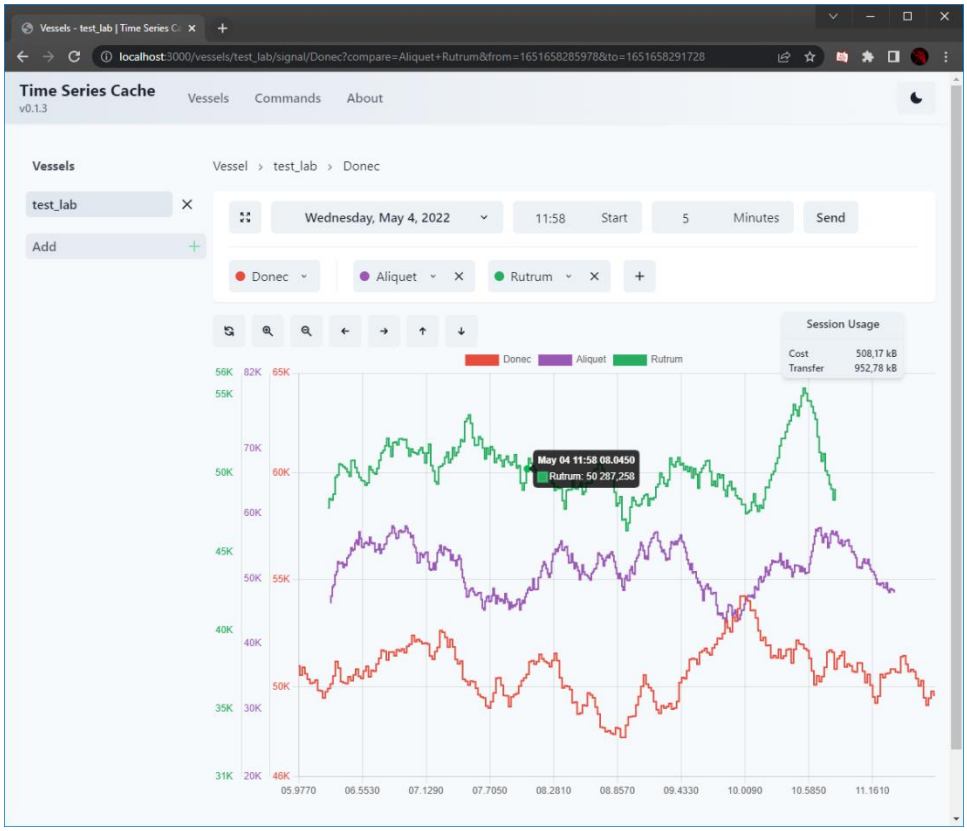


Figure 53: Light mode

As mentioned, each dataset plotted into the chart can be customized to look unique for visualization purposes. An example of this can

be found in Figure 52. Figure 53 is an example of the light mode which can be toggled by the Sun / Moon icon found in the top right corner of. the page. This light mode also works with full screen as seen in Figure 54.

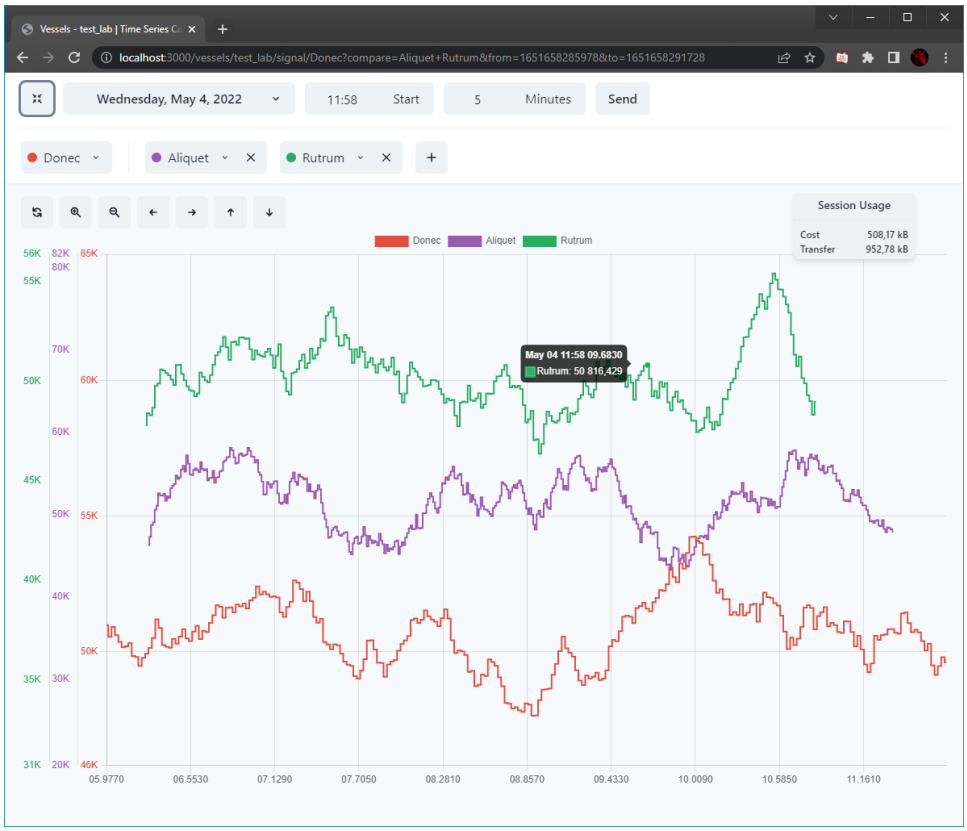


Figure 54: Light mode full screen

Command Line Interface

The command line interface can be found when navigation to the router /commands. This is mainly a dev tool for checking intervals of metadata, checking response data from range call, and wiping the database of signals.

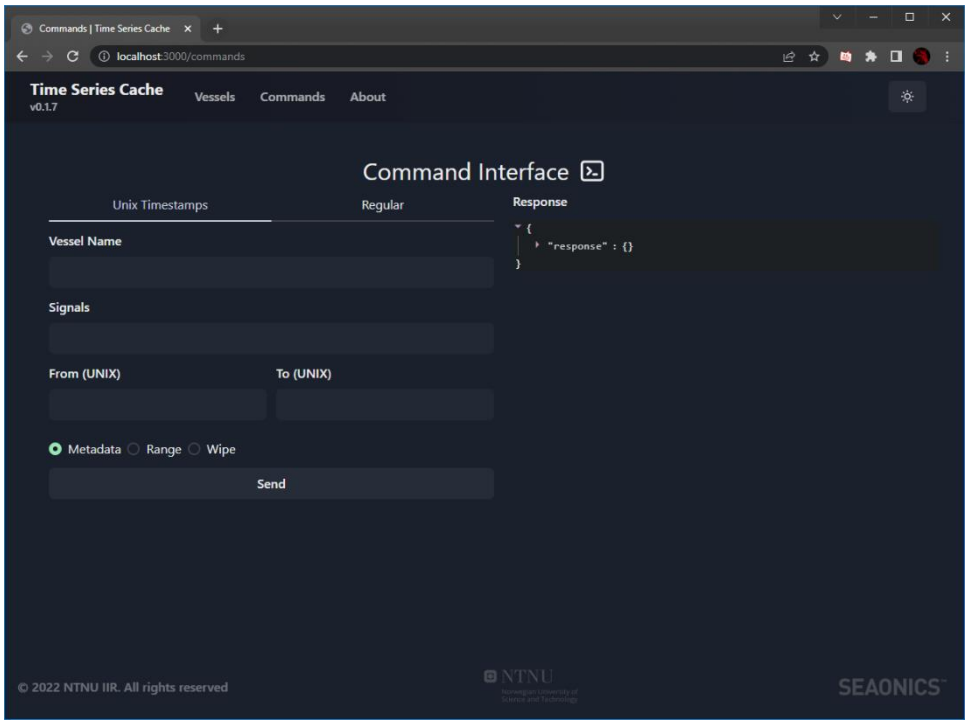


Figure 55: Command Line Page

The main input fields found in Figure 55:

- Vessel Name: here a user can input any available vessel that they have added to the site and cached signals for.
- Signals: here user can input signals they want to add to the request, they need to be inputted in a format where each signal is separated by a comma: “signal_1, signal_2, signal_3 …"
- Range: if using Unix tab, users input a Unix timestamp for “from” and “to”, if using Regular tab, users can select 2 dates and a time input for when in these dates the request should be done with.
- Request Type: Metadata, Range, Wipe

After sending requests the pure JSON data from the response will be displayed to the right of the input fields.

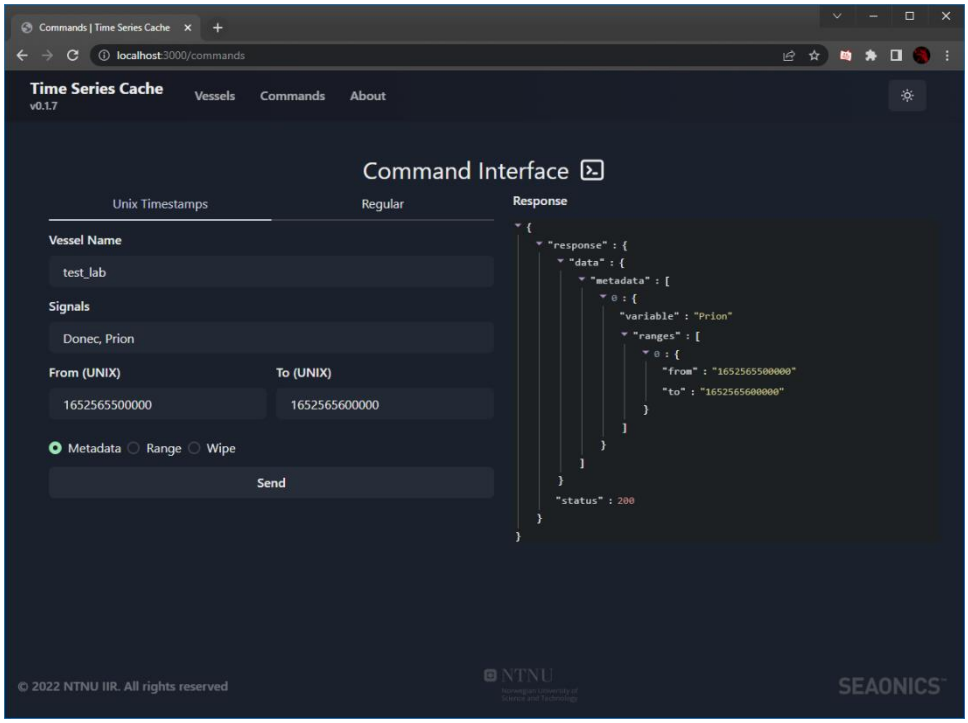


Figure 56: Metadata fetch example

Figure 56 is an example of asking for metadata in a specific range for 2 signals

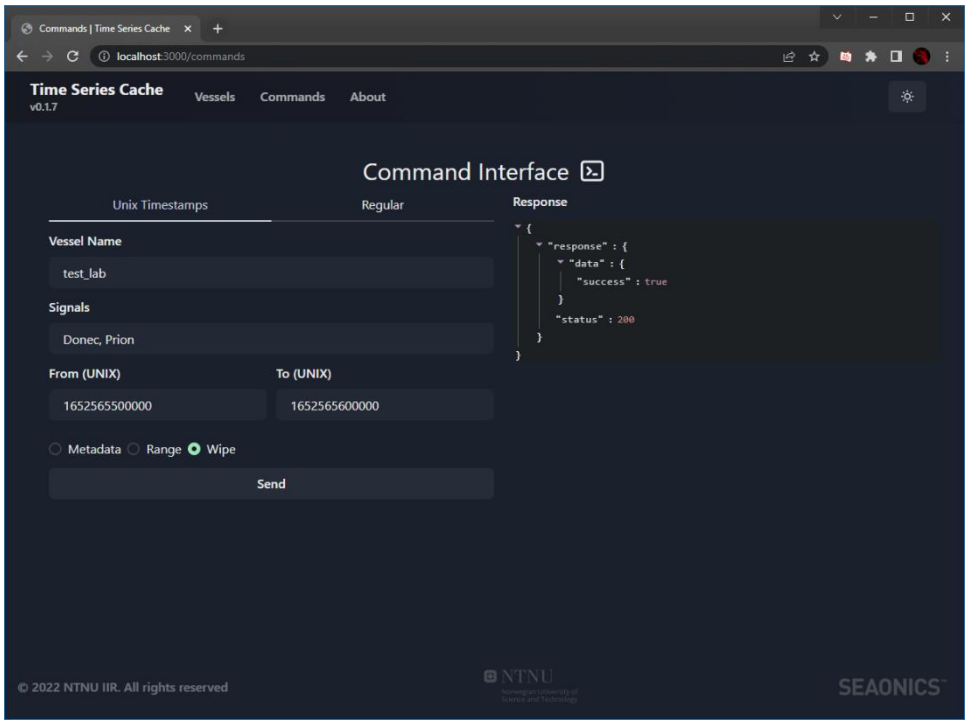


Figure 57: Database wipe example

Figure 57 is an example of deleting the database entries for the metadata discovered above

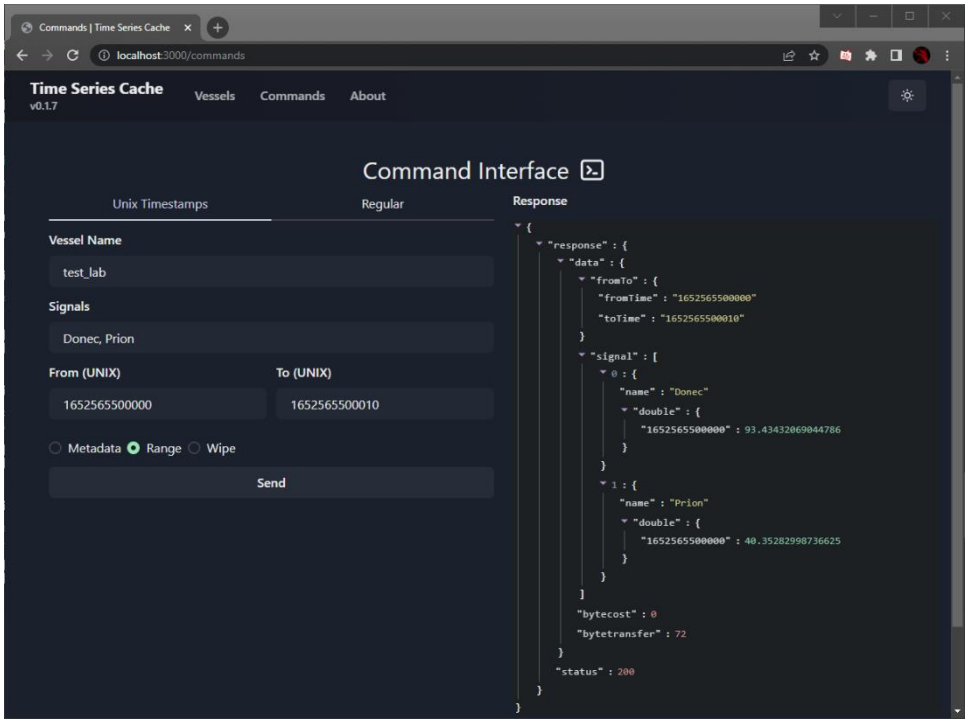


Figure 58: Manual range example

Figure 58 is an example of requesting a range for 2 signals for 10 milliseconds

Documentation of source code

Each codebase is documented with the JSDoc standard and have the TypeDoc dependency. To generate the documentation simply install the TypeDoc package (npm i) and run “typedoc” in the command line whilst in the root directory of one of the services. You should then see a /docs folder with the index.html file which you can open to see the generated documentation.

Testing

The system is tested with unit testing and mocks of dependencies.

Faker

FakerJS is an NPM package for easily creating randomized test data. The types of data faker can create are many and come in the form of simple datatypes like strings, numbers, floating point numbers etc, but also things such as currencies, names and even bitcoin addresses. While the data needed to be faked for testing the services created in this project is quite simple, the group went with faker for the sake of randomization.

Jest

Jest is a versatile testing framework for JS and TS maintained by Facebook, now Meta. The reason the group picked Jest over other testing frameworks, such as Mocha and Chai, is mostly due to the popularity of the different frameworks [1]. They all deliver mostly similar functionality. After reading the documentation of some different testing frameworks, Jest also looks to be the better documented and easier to get into framework. Jest has an expansive API for mocking modules, which was important for the group. Mocking a module means, in essence, to overwrite a modules functionality. For example, instead of doing a legitimate network call and returning the received value, a mock of a module might just generate some data and return it, eliminating the inter-service dependency.
To run the tests, first install the dependencies if you have not already by running npm i, assuming you have Node installed. Then run the test command with npm run test. Once the tests have run, the coverage report can be found in /coverage/lcov-report/index.html. At the time of writing the coverage is around 80-90%

Aedes

Aedes, whose name is a play on Eclipses' Mosquitto, is a lightweight mqtt broker. The broker comes in an NPM package and opens the possibility of creating MQTT brokers straight in your code. In testing, an Aedes broker is used to allow for easy mocking of both the broker and to simplify the mocking of a vessel service. The group initially stuck with an MQTT broker running on a VM, but due to entailing issues from having an instance too weak to run all services under high load, which is usually the case when running back-to-back or even concurrent tests, an in-memory Aedes broker was used for testing.

SQLite3

SQLite3 is a database capable of running in-memory, which means that it can be set up quickly for unit tests, removing the need for a database for testing. While the SQLite DBMS (Database Management System) is different from Postgres, the use of Sequelize does a good job of bridging that.

Cypress

Cypress is a popular end-to-end testing framework for frontend applications. It works by setting up a description of wanted HTML components and lets you write scripts that will click through your web application faster than a tester could. Cypress waits for async calls to external webservers and checks if the behaviour of the result is correct based on the description written in the test file. It comes with its own debug client, that lets the developer see the testing in real-time.

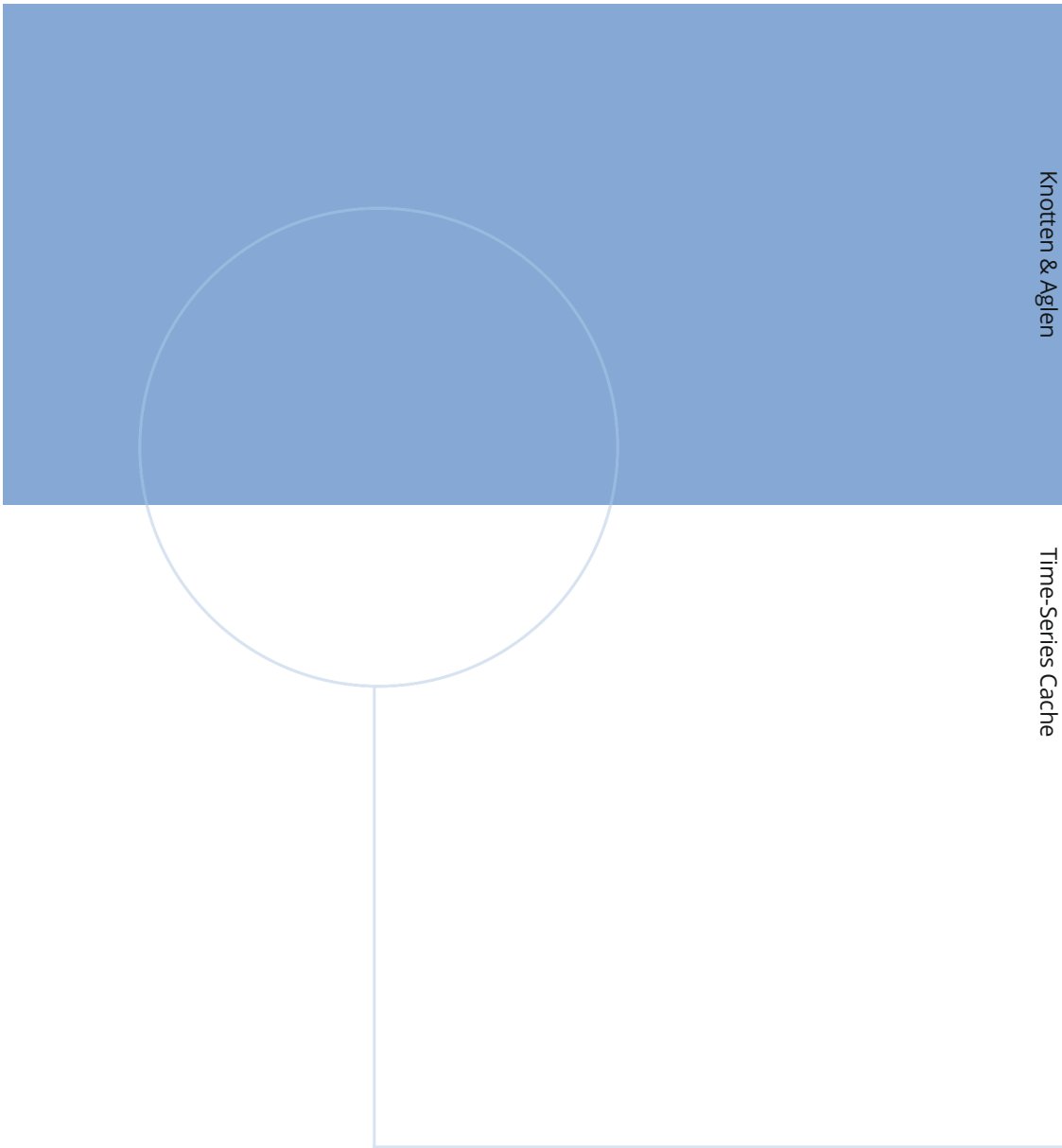
References

[1] J. Kithome, "LogRocket," 12 February 2021. [Online]. Available: https://blog.logrocket.com/the-best-unit-testing-frameworks-for-node-js/. [Accessed 20 March 2022].

[2] Chart.js, "Chart.js | Performance," https://www.chartjs.org/docs/latest/general/performance.html, 2022.

[3] Timescale, "timescale.com," [Online]. Available: https://www.timescale.com/products. [Accessed 18 May 2022].

[4] HiveMQ, "hivemq.com," [Online]. Available: https://www.hivemq.com/mqtt-cloud-broker/. [Accessed 18 May 2022].



Knotten & Aglen

Time-Series Cache

NTNU
Kunnskap for en bedre verden